A Proactive Approach to Application Performance Analysis, Forecast and Fine-Tuning

Siddhartha Kargupta

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University College London

Department of Computer Science
University College London

June 2014

Declaration

I, Siddhartha Kargupta, confirm that the work presented in this thesis is my own. Where information has been obtained from other sources, I confirm that this has been indicated in the thesis.

Some of the work presented in this thesis has previously been published in the following Journal and Conference papers:

- Kargupta, S., Black, S. (2012), A Novel Approach for Service Performance Analysis and Forecast, ACM Digital Library, Journal of Web Engineering, Volume 11 No. 2, pp146 – 176.

- Kargupta, S., Black, S. (2009a), Service Operation Impedance and its role in projecting some key features in Service Contracts, ICWE 2009, 9th International Conference on Web Engineering, San Sebastian, Spain, CEUR-WS.org, Volume 484, pp15 - 23.

- Kargupta, S., Black, S. (2009b), Visualizing the Underlying Trends of Component Latencies affecting Service Operation Performance, IEEE Xplore Digital Library, International Conference on Advances in Computational Tools for Engineering Applications, (ACTEA 2009), pp565 – 570.

Acknowledgements

It has been a privilege and great pleasure to be associated with the Department of Computer Science at the University College London. I felt honoured when David Rosenblum initially reviewed my work and accepted me as a PhD research student at the Department of Computer Science. I am immensely thankful to Antony Hunter for facilitating proper guidance required to complete this thesis. I would like to thank my supervisors Sue Black and Emmanuel Letier for their support and guidance for this thesis. I immensely enjoyed working with Emmanuel who took the time to review this work thoroughly. It was extremely helpful getting his feedback and input to the thesis. Without Emmanuel's advice, this version of the thesis would not have been possible.

It was an honour to know my examiners Andrea Zisman and Wolfgang Emmerich. I thank them from the core of my heart for their invaluable suggestions and advice, all of which have been incorporated in this thesis.

Without the blessings of my late parents, nothing would have been possible in this life of mine. I remembered them all throughout this research.

My in-laws had been a continuous source of encouragement and I thank them for their support.

Finally, I thank my wonderful family – my wife Suvu and children Sami (the human kind), Rino, Tom, Jerry (the dog kind), Timi (the bird kind) and Mew (the fish kind). Without their support (and distraction at times), this work would not have been possible.

Abstract

A major challenge currently faced by the IT industry is the cost, time and resource associated with repetitive performance testing when existing applications undergo evolution. IT organizations are under pressure to reduce the cost of testing, especially given its high percentage of the overall costs of application portfolio management. Previously, to analyse application performance, researchers have proposed techniques requiring complex performance models, non-standard modelling formalisms, use of process algebras or complex mathematical analysis. In Continuous Performance Management (CPM), automated load testing is invoked during the Continuous Integration (CI) process after a build. CPM is *reactive* and raises alarms when performance metrics are violated. The CI process is *repeated* until performance is acceptable. Previous and current work is yet to address the need of an approach to allow software developers *proactively* target a specified performance level while modifying *existing* applications instead of reacting to the performance test results after code modification and build. There is thus a strong need for an approach which does not require repetitive performance testing, resource intensive application profilers, complex software performance models or additional quality assurance experts.

We propose to fill this gap with an innovative relational model associating the operation's Performance with two novel concepts – the operation's *Admittance* and *Load Potential.* To address changes to a single type or multiple types of processing activities of an application operation, we present two *bi-directional* methods, both of which in turn use the relational model. From annotations of *Delay Points* within the code, the methods allow software developers to either fine-tune the operation's algorithm "*targeting*" a specified performance level in a *bottom-up* way or to predict the operation's performance due to code changes in a *top-down* way under a given workload. The methods do not need complex performance models or expensive performance testing of the whole application. We validate our model on a realistic experimentation framework. Our results indicate that it is possible to characterize an application Performance as a function of its Admittance and Load Potential and that the application Admittance can be characterized as a function of the latency of its Delay Points. Applying this method to complex large-scale systems has the potential to significantly reduce the cost of performance testing during system maintenance and evolution.

Key words: CI – Continuous Integration, CPM – Continuous Performance Management, SPE – Software Performance Engineering, PEPA – Performance Evaluation Process Algebra, UML - Unified Modelling Language, QoS – Quality of Service, SA – Software Architecture

Table of Contents

1 Introduction

1.1 Context

Performance experts have a tendency to regurgitate certain performance clichés to each other and to anyone else who will listen. Here is one such cliché:
“Acme Corporation just lost a $40 million sale because their new application cannot meet service level targets under heavy load. How much money do they need to lose before they carry out capacity planning?” (Gunther, 2005).

Information Technology (IT) organizations are under pressure to reduce the cost of testing, especially given its high percentage of the overall costs of application portfolio management (Anderson, 2012). A need to improve performance testing, analysis and monitoring between development, testing and IT operations is driving convergence of the tools (Murphy et al., 2012).

Significant research involving various modelling approaches including Software Performance Engineering (SPE) techniques (Smith et al., 1997), Queuing Networks (Kleinrock, 1975; Menasce et al., 2002) and their various extensions (as Layered Queuing Networks), Stochastic Process Algebra (Gilmore et al., 1994; Hermanns et al., 2002) and Stochastic Petri Nets (Balbo, 2001; Marsan et al., 1995) have been undertaken to analyse and assess the performance of systems. Simulation performance models have been generated from high level UML descriptions of software architecture (Marzolla, 2004). Prediction methods using Kalman Filters (Kalman, 1960), Support Vector Machines (Vapnik, 1998) and other kernel based languages like KLAPER - Kernel Language for Performance and Reliability (Grassi et al., 2005) have been explored to predict states of processes. Various techniques to monitor systems at runtime through the use of instrumentation and profiling have been explored as well (Woodside et al., 2007).

However, the above mentioned approaches mostly require complex SPE techniques, intensive effort to interpret performance models and resource intensive application profilers. Most of the techniques are applied during the early stages of the software development life-cycle and require additional quality assurance experts. The software engineers are required to learn new, non-standard modelling formalisms and notations and to be able to specify the software system using process algebras.

With the advent of Continuous Performance Management (CPM), projects incorporate automated performance tests into the Continuous Integration (CI) process. Performance is assessed by either invoking load tests on the high level business use cases (Haines, 2008) or measuring the performance of the functional units of code during the CI process (Thakkar et al., 2008; Clark, 2009). Automated load tests are run within the CI test harness to baseline and track the application's scalability during development. Integration and load testing requires a different test bed because these are more like business use cases rather than functional tests. The HttpUnit extension of JUnit works well here (Haines, 2008). If performance of functional units of code is tested during the CI process, tools like JUnitPerf are typically used. JUnitPerf is a collection of JUnit test decorators used to measure the performance and scalability of functionality contained within existing JUnit tests (Clark, 2009). There are other performance testing tools used in the industry today like LoadRunner and JMeter (Lloyd, 2012), which do not require unit tests and are often used for CPM. These tools are used to test applications under externally generated load (Hansen, 2011).

However, although performance is assessed continuously in CPM, it still follows the traditional *reactive* and *repetitive* approach to performance analysis and does not provide insight into how the internal processing activities impact the timeliness of responses. To achieve this, code profilers and memory debuggers are needed during CPM (Haines, 2008). Although profilers, debuggers or JUnitPerf are not used in production environments, during testing these tools affect performance by adding overhead to the application being measured and to the machine it is running on (Duggan et al., 2011) and the measured data may get skewed.

1.2 Motivating Example

1.2.1 Context

In order to motivate and illustrate our techniques, we will refer to a fictive application and modification scenario inspired by real systems and events. Our example application is SKGWorld Forex's Trade Confidence Generator (TCG) application. This application is used to generate the real time confidence level for executing trades for the major currency pairs like Pound Sterling (GBP) / US Dollars (USD), Euro / USD, USD / Swiss Franc (CHF) etc. Another Auto-Trader application queries the confidence level at real time from the TCG application and executes Forex trades if the confidence level is above a particular threshold.

The foreign exchange market is the world's largest financial market, accounting for more than $4 trillion in average traded value each day as of 2011 (Folger, 2011). Due to this extreme high liquidity of the Forex market, price movements are also very fast and timely execution of trades is very crucial to the success of the trades. Hence, performance of the TCG application is the key to the success of its consumers. For the purpose of this example, we will focus on the TCG application only. The Auto-Trader application serves as the consumer to the TCG application. Both applications are owned by SKGWorld Forex Ltd and are hosted on the same network at a data centre. Hence no unexpected fluctuation of network bandwidth is foreseen between the Auto-Trader and the TCG application.

1.2.2 Architecture

The TCG application is composed of several key components as shown in Figure 3 below. Some of these components perform “Technical Analysis” of the market data. In Forex, stocks and commodities trading, technical analysis (TA) is the art of examining the currency pair, stock or commodity charts to identify potential trading signals (eHow, 2014). These TA components perform different types of statistical analysis of the real time and historic prices of the currency pairs and then apply further business logic to generate the technical confidence levels for the respective components. All of these TA components fetch the currency pair quotes from a Quotes Distributor component. The Quotes Distributor component retrieves the real time and historic prices from an external system over a dedicated leased network, thus eliminating the chances of unexpected bandwidth fluctuations. There is inter-dependency between the various TA components. Due to the inter-dependency, the TA components process the request sequentially. The Process Orchestrator component, which is implemented as a web service, manages the orchestration sequence between the different TA components and also aggregates the confidence levels generated by the components.

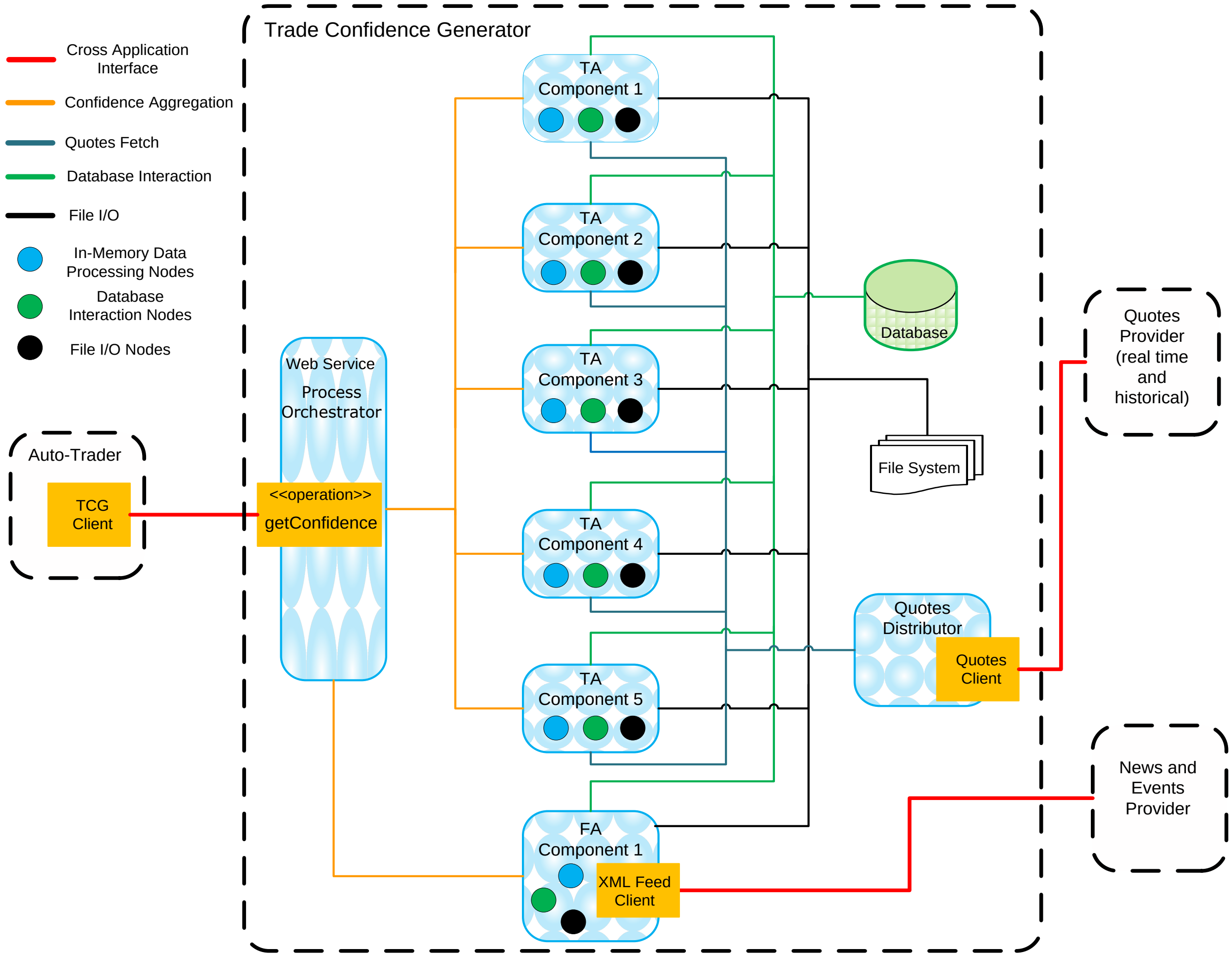


Figure 3: **Component Level View of TCG Application**

There is another component which does Fundamental Analysis of the events happening across the world. In Forex, stocks and commodities trading, fundamental analysis (FA) attempts to study everything that can affect the security's value, including macroeconomic factors like the overall economy and industry conditions (Investopedia, 2014). This FA component fetches all the events and news related to the global financial markets from an external system in the form of XML feeds over a dedicated leased network. The XML data is then processed and business logic applied to generate a fundamental signal.

Depending on their nature, the mode of input and the frequency of updates, the business logic for all the TA and FA components are either stored in an Oracle database or as structured text files on the file system. The components retrieve business logic either from the database or the text files and store the outcome of their analysis back to the database or in text files accordingly for audit trail and any other future requirements. The confidence levels generated by the individual components are aggregated in the Process Orchestrator component by applying a set of rules and logic. This aggregated final confidence level is exposed externally by the TCG application through a getConfidence(...) operation on the web service interface. In our example, the Auto-Trader application makes a call to this operation to retrieve the consolidated confidence level.

The Forex market is generally considered to be very volatile in nature. High volatility means that the price of the currency can change dramatically over a short time period in either direction (Easy Forex, 2014).Volatility of a currency pair changes over time. There are some periods when prices go up and down quickly (high volatility), while during other times they might not seem to move at all (low volatility). Any news or event which influences the global economic condition and the financial markets around the world impacts the volatility of the Forex market. Changes to market conditions affect the way in which the currency prices need to be observed and analysed. Typically, other business applications are architected and designed in a way that possible changes to business logic can be handled through external configurations instead of modifying the application algorithms. However, it is extremely difficult to predict how the global financial market will react to world news and events. Hence, it is usually not possible to pre-configure the algorithms which may possibly be needed to deal with future requirements. Due to this phenomenon, the algorithms and the business logic used by the TA components and the FA component of the TCG application warrants frequent alterations to keep up to the global economic sentiments.

1.2.3 Performance Analysis Scenario

Imagine that during one such change to the global financial market sentiment due to military tensions in the Middle East, the algorithms of the TA components and that of the FA component of the TCG application were required to be changed significantly. The higher management of SKGWorld Forex Ltd mandated that the software development team should implement the necessary changes as soon as possible maintaining the existing QoS of an average response time threshold of 0.5 second for the getConfidence(...) operation under an inbound workload of 10 requests/second.

A project was initiated and the software development team embarked on making changes to the TA and FA components within a CI framework. To ensure continuous monitoring of performance, CPM was implemented with automated load test runs after every nightly build within the CI test harness to baseline and track the application’s scalability during development. SKGWorld Forex Ltd has a team of software engineers but does not have any dedicated performance experts with the knowledge of complex SPE techniques, process algebra, petri-nets, performance models or how to interpret the outputs of application profilers. However, in accordance to the performance mandate from the higher management, the developers set 0.5 second as the higher threshold for the average response time of the getConfidence(...) operation. On several occasions, this threshold for average response time was violated during the automated load tests and due to the unacceptable performance observed, the changes had to be rolled back and redone again applying other variants of the algorithms. The problem was two-fold. As the automated load tests did not provide insight into how the internal processing activities are impacting the timeliness of responses of the components, the software engineers found it difficult to ascertain exactly which processing activity (or group of activities) is causing the bottleneck without the use of resource intensive profilers. Also, this CPM process was reactive in nature. Only after running the automated load tests post every nightly build, it was observed that the performance was unacceptable and the changes had to be rolled back and redone in a “trial and error” manner. This example illustrates how the development team at SKGWorld Forex would benefit from a simple proactive approach to assess the performance of existing software applications undergoing changes without the need of repetitive load testing, complex SPE techniques, performance models and resource intensive application profilers.

1.3 Research Objective

Software applications continuously evolve due to changing market requirements and short innovation cycles. Software performance engineering in its essence is not directly applicable to such scenarios (Westermann et al., 2010). Many approaches focus on early lifecycle phases assuming that the software system is being built from scratch i.e. green-field scenario and all its details are known. These approaches neglect the influences of existing software. Detailed information about the internal structure, which is required for performance prediction, may not be available during the early phases. Many approaches have been published in the context of software performance engineering but none of them has achieved widespread industrial use (Koziolek, 2010). In view of this and all the drawbacks of the various performance assessment techniques mentioned in Section 1.1, the key question this thesis tries to answer is the following:

Is it possible to formulate a proactive approach to software performance analysis and forecast for *existing* software applications undergoing frequent software releases and updates, which will satisfy the following criteria:

a) It will not need repetitive load or performance testing to assess the impact of the changes to an evolving application.

b) It will not require complex SPE technique, effort intensive to interpret performance models or system resource intensive application profilers.

c) It can be used by the software developers without the need of an additional quality assurance expert. It will neither require the software developers to learn new, non-standard modelling formalisms and notations nor to specify the software system using process algebras.

d) It will yield precise and accurate results. The ability of the approach to reveal the true pattern in the collated data may be determined by the statistical power of the forecasts and functions (Wohlin et al., 2012), which shall be as high as possible, ideally 1.

e) It will be applied during the implementation phase or later to ensure more precise performance measures.

f) It will enable a "bottom-up" approach towards achieving a *target* performance criterion. Given a specified performance level, the software engineers will be able to configure the timeliness of the various processing activities of an operation to achieve the pre-specified *target* performance level under a specified inbound workload condition without the need of repetitive load testing as done in CPM. Examples of some typical processing activities are in-memory data processing, file I/O, database interaction etc. *Note*: As the timeliness of a processing activity depends on how its algorithm has been designed and developed, we will refer to this aspect as "algorithm" henceforth.

g) It will also allow the conventional "top-down" approach of performance assessment and enable the software engineers to assess the impact on the performance of an application operation due to changes to the configuration and algorithm of the operation, under a given inbound workload.

h) It will enable the software engineers to ascertain the inbound workload that an application operation may sustain while maintaining a *target* performance level for a given configuration and algorithm of the application.

This thesis sets out to answer the above question affirmatively. It attempts to establish an approach that shall not require repetitive load/performance testing, complex SPE techniques, performance models and resource intensive application profilers. It shall also not require the software engineers to learn new, non-standard modelling formalisms and notations or to be able to specify the software system using process algebras. Without any of the above, the approach will enable the software engineers to probe the latencies of the various transaction processing activities and perform the following:

1. Proactively fine-tune the configuration and timeliness of the algorithm during modification of the application to *retrofit* a pre-specified *target* performance level for an operation or transaction under a given load condition. This will deliver a specified performance level sooner than the current reactive approach of load testing during the CPM cycle, which requires iterations of load testing, performance measurement, code modification and build in the event of any performance issue.
2. Determine the inbound workload that an application operation will be able to sustain while maintaining a specified performance level for any given algorithm and configuration of the application.
3. Analyse and predict the impact of changes to the low level algorithm and configuration of an application operation on the external performance of the operation.

1.4 Research Contributions

This thesis introduces two novel concepts in the context of an application operation's runtime performance, which are Admittance and Load Potential (defined below in this section). Associating the operation's ***P***erformance with its ***A***dmittance and ***L***oad ***P***otential, this thesis establishes an *innovative* relational model (Kargupta et al., 2009a). Using the derived model, referred to as the "*PALP Model*" in this thesis, two bi-directional methods are proposed to enable the software engineers to perform the following *proactively* without requiring repetitive load testing or complex performance models:

1. Fine-tune an application operation's algorithm to *retrofit* a pre-specified "*target*" performance level for a given workload in a *bottom-up* way.

2. Analyse and predict an application operation's performance under a given workload due to changes to the operation's algorithm in a *top-down* way.

3. Analyse and predict the inbound workload that an application operation will be able to sustain in order to maintain a particular performance level given a particular algorithm and configuration.

The *PALP* model and the two methods form a proactive approach that combines model based and measurement based performance engineering techniques to evaluate the performance of *existing* software applications undergoing frequent updates. This approach enables proactive fine-tuning of the application algorithm and configuration by the software engineers upfront during code development and modification through local systems testing to achieve a prescribed ***target*** performance level instead of recourse to the reactive approach currently used in CPM, which requires *repetitive cycles* of load testing, performance measurement, code modification and build to fix any performance issue.

The methodology proposed will allow the software engineers to assess the impact of the changes to an application operation or transaction upfront during application code modification without the need to involve additional quality assurance experts. It attempts to circumvent the unnecessary notational hurdle, which acts as an impediment to the understanding and uptake of modern performance analysis technologies. No system level utilization diagnostics and measurement will be required for which the software developers may not have adequate expertise.

Standard 3 – 5 years technology refresh periods are applied across all IT hardware (Archstone Consulting, 2013). IT hardware technologies typically do not undergo change during application software updates. In view of this, the thesis does not include such technology refreshes. The changes are purely at the application layer. Changes to the environment, which includes hardware infrastructure and networks, are out of scope of this thesis.

The usage of the proposed model and methods is described in details in Chapter 3 under Sections 3.3, 3.4 and 3.6. A brief overview of the model, its attributes and the methods underpinning the proposed approach is presented below:

1. *An innovative relational PALP model associating an application operation's Performance, Admittance and Load Potential*

To analyse the performance of an application operation and to associate the operation's performance to its various internal processing activities and its inbound workload, this thesis introduces two *novel* attribute concepts namely Admittance and Load Potential in the context of runtime consumption of an application operation. With reference to Figures 1 and 2 below, the three attributes of the relational *PALP* model are explained.

- *Application Operation Performance (**'P'**)* – The performance '*P*' of an application operation is the inverse of its response time. The unit of this attribute is (unit of time)$^{-1}$. It is the measure of the application operation's timeliness of response under a given inbound workload. For a typical application operation, the performance of an operation degrades with the increase in the operation's inbound workload.

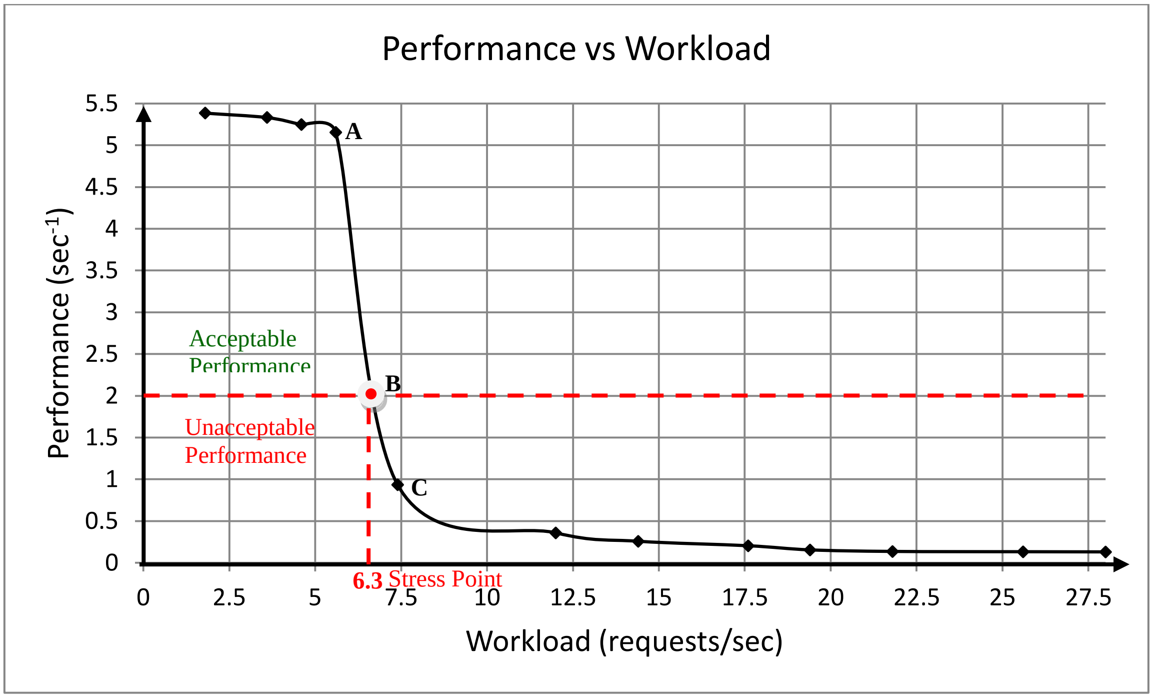


Figure 1: **A sample graph of operation's Performance versus Workload**

Figure 1 above presents an illustrative performance versus inbound workload graph. It shows for a typical application operation, how the performance degrades with the increase of workload. Usually, as shown in Figure 1, performance degrades linearly up to a certain point (Point **A** in Figure 1) and then degrades sharply. This is often due to one or more critical resources in the system becoming overloaded and can no longer work efficiently resulting in the physical system approaching the maximum throughput (Menasce et al., 2002). Again beyond a certain point (Point **C** in Figure 1), as the workload increases even further, the decrease in the performance becomes less abrupt and smoothens out. This often happens as the response time saturates since all the threads tend to be busy all the time and the queue of threads tend to be always full (Menasce et al., 2002; Harbitter et al., 2001).

For the purpose of our example in Figure 1, we assume that the business requires the average response time of the operation to be at the most 0.5 second. Given the definition of performance, the business requires the performance to be at the least 1/0.5 = 2 sec^{-1}. Figure 1 shows that the performance reaches 2 sec^{-1} when the arrival rate of the incoming workload is 6.3 requests/second. Beyond this point (Point **B** in Figure 1), as the workload increases further, the performance degrades below the required Quality of Service (QoS) and becomes unacceptable.

In this thesis, we define *Stress Point* as the workload beyond which the performance becomes unacceptable to the stakeholders. In Figure 1, the Stress Point is shown beyond the knee in the performance curve where the performance degrades sharply. However, it is to be noted that the value of Stress Point is fully dependant on the business requirements. For example, in our scenario, had the business required the average response time of the operation to be at the most 1.0 second, the required performance would have been at the least 1/1.0 = 1 sec^{-1}. The Stress Point in that case would have been 7.4 requests/second instead of 6.3 requests/second.

- *Application Operation Load Potential (‘V’)* – We define an application operation’s Load Potential ‘*V*’ as the difference between the operation’s Stress Point and its current inbound workload:

$$V = \textit{Stress Point} - \textit{Inbound Workload} \qquad (1.1)$$

If the Load Potential is positive, then the application operation has an acceptable performance and the Load Potential measures the headroom between the current inbound workload and the Stress Point.

If the Load Potential is negative, then the application operation's performance is not acceptable and the negative Load Potential indicates how far into the red the current workload is.

With reference to Figure 1, Table 1 shows how the Load Potential is calculated at some of the workload data points for the illustrated Stress Point of 6.3 requests/sec:

Point Reference	**Workload (requests/sec)**	**Stress Point (requests/sec)**	**Load Potential '*V*' (requests/sec)**
A	5.6	6.3	0.7
B	6.3	6.3	**0**
C	7.4	6.3	-1.1

Table 1: **Workload and Load Potential for a given Stress Point**

Hence, for a given Stress Point of an application operation, if the inbound workload decreases, '*V*' increases and vice-versa. The unit of this attribute is the number of requests per unit time. Figure 2 shows the same phenomenon as Figure 1 but plots the performance against the Load Potential for each of the workload conditions shown in Figure 1. It shows that performance increases with the increase in the Load Potential. As evident from Figures 1 and 2, the increase in performance due to the increase in Load Potential mirrors the decrease in performance due to the increase in inbound workload.

One may define performance of a typical application operation as a function of the operation's Load Potential. For example, Figure 2 below shows how the performance varies with the Load

Potential for the same illustrative example depicted in Figure 1. The Load Potentials at Points A, B and C in Figure 2 correspond to the workloads at Points A, B and C in Figure 1.

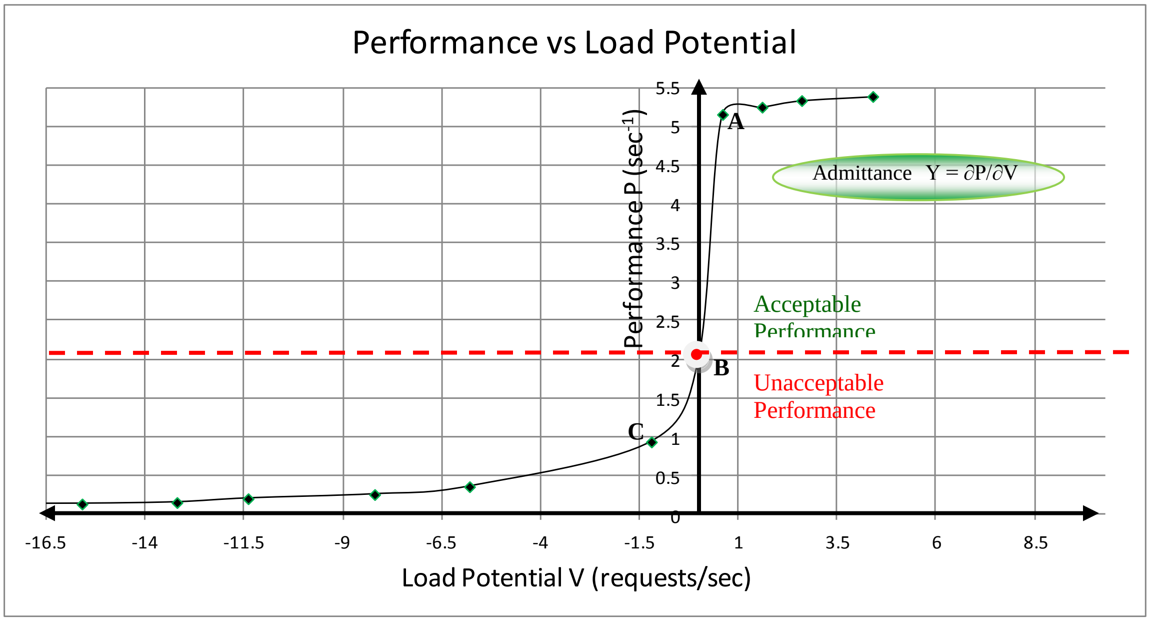


Figure 2: **A sample graph of operation's Performance versus Load Potential**

Application Operation Admittance (*'Y'*) – In this thesis we introduce the novel concept of Admittance *'Y'* for an application operation as the derivative of the operation's performance with respect to its Load Potential:

$$Y = \delta P/\delta V \qquad (1.2)$$

In Electrical Engineering, Admittance is a measure of how easily a circuit or device allows current to flow and is defined as the inverse of the device's Impedance (Southwire, 2013). This concept of Admittance is introduced in software applications by this thesis. It is defined as the measure of ease with which a request to an application's operation is processed and response sent back. All the low level application processing activities required internally to process a request, cumulatively introduce impedance to the application operation's performance due to the time taken to perform the processing activities. Admittance may also be viewed as the inverse of this impedance created. This is a numeric measure only.

The thesis first derives the relational *PALP* model associating the three above mentioned attributes of **P**erformance, **A**dmittance and **L**oad **P**otential of an application operation and subsequently evaluates the model through experimentation. The derivation and evaluation of the model are described in Chapters 3 and 5 respectively.

2. *A bi-directional method to enable software developers fine-tune the Delay Points of a single type to achieve a target Performance level during code modification*

This research introduces the concept of "Delay Points" in an application (Kargupta et al., 2009b). The application components catering to an operation need to perform various types of processing activities to process a request. Each of these processing activities introduces some latency or delay to the response process and is hence referred to as a *Delay Point*. These Delay Points at the application layer process the activities using the underlying system resources. This thesis measures the latencies of the Delay Points at the application layer and does not delve into the underlying system resources. Some examples of Delay Points in a typical application are in-memory data processing, file I/O, database interaction etc.

Using the deduced *PALP* model, this thesis establishes a *bi-directional* method to allow the software developers to probe the latencies of an application operation's Delay Points of a *specific* type and perform the following:

- Proactively fine-tune the response times of the Delay Points during code modification to *retrofit* a pre-specified *target* performance level for a given workload in a *bottom-up* way.
- Predict the changes to the operation's performance under a given workload due to modifications to the Delay Points in a *top-down* way.

The bi-directional (bottom-up and top-down) approaches are described in Chapter 3, under Section 3.3.

Both of the above will be achieved without the need of repetitive performance testing (even if the testing is performed during a CPM cycle), complex SPE techniques, performance models and resource intensive application profilers. As the method is applied during the implementation phase in software development, it addresses the issue highlighted in (Balsamo et al, 2004), which states that most of the modelling approaches try to apply performance analysis very early, typically at the software architecture and design level and hence still require much detailed information from the implementation phase to carry out performance analysis. The thesis establishes that under a given workload condition and hosting environment, the operation's Admittance ('Y') can be expressed as a mathematical function of the total cumulative latency of the Delay Points of a *specific* type across all the supporting application components catering to the operation.

To improve the precision of predictions, calibration of any derived mathematical model is often required (Hill, 1998). In this thesis, some initial measures towards calibration have been undertaken for the derived functions. This thesis assesses the integrity of the derived functions by comparing the *actual* performance measured during experiment runs to that of the *predicted* performance obtained by using the previously derived statistical functions. The delta between the actual measured data and that predicted from the previously derived functions is then used to compute the statistical powers (Wohlin et al., 2012) of the experiments. The computed statistical powers were high and no further detailed calibration of the models in a recursive manner is undertaken in this thesis. However, such detailed recursive calibration using error feedback may form part of the future work to augment this research.

3. *A bi-directional method to enable software developers fine-tune the Delay Points of multiple types simultaneously to achieve a target Performance level during code modification*

Using a combination of the *PALP* model and the above method, this thesis establishes a second *bi-directional* method allowing software developers to probe the latencies of an application operation's Delay Points of *multiple* types and perform the following:

i. Proactively fine-tune the response times of all the Delay Points catering to the operation to achieve a pre-specified *target* performance level without the need of repetitive performance testing (even if the testing is performed during a CPM cycle), complex SPE techniques, performance models and resource intensive application profilers. This enables targeting a specific performance level upfront during development rather than going through a cycle of

Load Testing, Performance Measurement, Code Modification and Build in the event of any performance issue.

ii. Predict the possible change to the operation's performance due to simultaneous modifications to the multiple types of Delay Points supporting the operation. To achieve this, a matrix based technique is used for the assessment of performance due to simultaneous changes to the various types of Delay Points (Kargupta et al., 2011).

The cumulative latency of the Delay Points of each specific type is treated as a predictor variable for the application operation's admittance and hence the performance. As there are various types of Delay Points supporting the application operation, there will be multiple predictor variables. A heuristic method is applied for estimating the best-fit relative weights of the predictor variables in multiple regression. The relative weights determine the proportionate contribution of the respective Delay Point latencies to the overall operation Admittance. The set of relative weights is then used to predict the operation Admittance for any future set of latency values for the different Delay Points. A method of calibrating the matrix aided model has also been explored.

The *PALP* model and the two methods formulated above are very much complementary in nature and in combination support addressing the thesis question affirmatively. Used jointly, the resulting framework facilitates a flexible *three-way* predictive technique involving an application operation's Performance, Admittance and Load Potential. By enabling fine-tuning the response times of an application operation's Delay Points during code modification to *retrofit* a specified *target* performance level, the proposed approach attempts to establish a proactive way to performance analysis instead of recourse to the reactive CPM approach.

1.5 Thesis Outline

This rest of the thesis has been structured into the following chapters:

Chapter 2 (*Background and Motivation*) – Relevant work conducted in the area of software performance engineering towards analysis and forecast is presented in this chapter. Various prediction methods have been reviewed in this chapter. It introduces the context of this research and explains the background of the problem describing the motivation for this research.

Chapter 3 (*Performance Analysis Methodology – the Model and Methods*) – this chapter discusses the rationale behind the performance analysis methodology proposed and states the three hypotheses for a) the relational model between operation ***P***erformance, ***A***dmittance and ***L***oad ***P***otential (*PALP* model), b) the impact on operation Admittance for changes to Delay Points of one type and c) the impact on operation Admittance for changes to Delay Points of multiple types simultaneously. These three hypotheses are evaluated in subsequent chapters. The rationale for this thesis to adopt the technique proposed by (Johnson, 2000) to evaluate the *best-fit* relative weights associated with the cumulative latencies of each type of Delay Points is explained. In this chapter, we discuss and justify the appropriateness of the method adopted for the research problem at hand. The boundary assumptions for the applicability of the research are discussed and at the end it explains when and how in the software development cycle this technique is used.

Chapter 4 (*Experiment Environment*) – this chapter describes the actual Java implementation of the application framework developed to perform the experiments in a controlled environment. The framework resembles a typical application operation provisioning – consuming scenario with various types of backend components accessed. Some of the backend interactions between the facade orchestration web service and the various backend components are detailed in this chapter with the aid of UML Sequence diagrams. The static class hierarchy of the application framework is presented with the aid of UML Class diagrams. Some of the environmental constraints that have been observed in this thesis are discussed.

Chapter 5 (*Evaluating the runtime PALP model*) – this chapter describes the *Controlled Experiments* (Wohlin et al., 2012) performed to evaluate the high-level *runtime* abstract model by applying established mathematical techniques. In these experiments, one or more variables are manipulated and the other variables are controlled at fixed levels. The model facilitates a three-way prediction capability. The concept of application operation admittance is supported in this chapter through the collation of observed data from experiments run in controlled environments. The statistical power (Wohlin et al., 2012) of the experiments has been derived in this chapter to demonstrate the ability of the experiments to reveal the true pattern in the collated data. Finally, an assessment of the threats to the validity of the results that have been derived is discussed.

Chapter 6 (*Single-Type Delay Point Changes: Method Evaluation*) – this chapter describes the steps of combining the derived *PALP* model and measurement data to extract the patterns in which the application operation's Admittance is influenced by the variation of the processing intensities of the Delay Points of a specific type across all the supporting application components. The statistical power of the experiments has been derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data. Finally, an assessment of the threats to the validity of the results that have been derived is discussed.

Chapter 7 (*Multi-Type Delay Points Changes: Method Evaluation*) – in this chapter the method of working on a matrix based predictive model to forecast the application operation's performance for simultaneous changes to multiple types of underlying application component Delay Points is evaluated. An assessment of the threats to the validity of the results that has been derived is discussed.

Chapter 8 (*Related Work*) – this chapter presents a focused discussion of the related work already undertaken. It compares the contributions of this thesis to the findings of the related work.

Chapter 9 (*Conclusion*) – in this chapter we summarize the thesis in terms of the context, motivation, hypotheses, methodology, evaluation objectives and methodology, all the findings and the overall contribution of the research. It presents an evaluation of the proposed approach against the research objectives in a summary table. A discussion of how this research will complement Cloud and Elasticity is presented. Finally, some of the potential future work that may be carried out to improve the precision of the proposed methodology and address real life production systems is discussed.

2 Background and Motivation

Over the last decade and more, research has addressed the importance of integrating quantitative validation in software development processes in order to meet non-functional requirements. Among these, performance is one of the most influential factors to be considered (Balsamo et al, 2004). Performance problems may be so severe that they can require considerable changes to design and also at the software architecture level. In the worst cases, they can even impact the requirements level. Software performance is a pervasive quality difficult to understand because it is affected by every aspect of the design, code and execution environment. By conventional wisdom performance is a serious problem in a significant number of projects. It causes delays, cost overruns, failures on deployment and even abandonment of projects. But such failures are seldom documented. A survey of information technology executives (Compuware, 2006) found that half of them had encountered performance problems with at least 20% of the applications they deployed (Woodside et al., 2007).

Various approaches have been proposed for general methodologies for software performance focusing on early predictive analysis. They refer to different specification languages and performance models and consider different tools and environments for system performance evaluation (Balsamo et al, 2004). To analyse and assess the performance of systems, past research has explored various modelling approaches including SPE techniques, Queuing Networks and their various extensions, Stochastic Process Algebra, Stochastic Petri Nets, simulation based methods and other techniques. With the current CPM, projects incorporate automated performance tests into the CI process. Performance is assessed by either invoking load tests on the high level business use cases or measuring the performance of the functional units of code during the CI process.

The various approaches are discussed in a critical manner in the following sections, which consequently leads to the motivation for this thesis.

2.1 Queuing Network-Based Methodologies

The notion of Queuing Network (QN) as a network of interconnected Queues that represent the computer system has been explored extensively (Kleinrock, 1975; Menasce et al., 2002). A Queue in a QN stands for a resource and the queue of requests waiting to use the resource. Servers where no request is refused and all the arriving requests are queued for service form an Infinite Queue. Servers where arriving requests are rejected based on the volume of existing requests already being processed or queued forms a Finite Queue. QNs influence the software's performance. Different types of QNs such as Single-Class Open QNs, Multiple-Class Open QNs, Single-Class Closed QNs and Multiple-Class Closed QNs have been discussed in details. Significant work has been undertaken on various methodologies which propose transformation techniques to derive QN based models like Extended QN (EQN) or Layered QN (LQN) from Software Architecture (SA) specifications (Kant, 1992; Franks et al., 1995; Rolia et al., 1995; Woodside et al., 1995; Trivedi, 2001; Mania et al., 2002; Petriu et al., 2004). Other proposed methods are based on the SPE methodology introduced by Smith in her pioneering work (Smith, 1990).

2.2 Methodologies Based on the SPE Approach

The SPE methodology (Smith, 1990; Smith et al., 2002) was the first comprehensive approach to integrate performance analysis into the software development process, from the earliest stages to the end. The use of software execution models and system execution models has been described in this methodology. The first one represents software execution behaviour in the form of Execution Graphs (EG). The second one is based on Queuing Network models and represents the system platform including hardware and software components. A Layered Queuing Network model is derived from a Software Architecture specified by means of a Class Diagram and a set of Sequence Diagrams. (Smith et al., 1997) describe the SPE performance modelling tool, SPE.ED and its use for performance engineering of object-oriented software. The software execution models and system execution models are combined to derive knowledge of the performance. The SPE.ED approach is embedded into a general method called Performance Assessment of Software Architectures (PASA). It gives guidelines and methods to determine whether an SA can meet the required performance objectives (Williams et al., 2002). SPE is extended to component based (CB) applications through the CB-SPE framework (Bertolino et al., 2004). It is a compositional methodology for CB performance engineering and its supporting tool. CB-SPE adapts to a CB framework, the concepts and steps of SPE technique. The methodology consists of a component layer and an application layer in which the documented component properties are instantiated and composed to predict the performance properties of the assembled system. XML based interchange format Software Performance Model Interchange Format (S-PMIF) is formulated to exchange information between (UML-based) software design tools and software performance engineering tools while Performance Model Interchange Format (PMIF 2.0) is a common representation for system performance model data that can be used to move models among system performance modelling tools that use a QN model paradigm (Smith et al., 2005).

However, despite all its merits, applying SPE techniques in practice is still very challenging (Happe et al., 2010). The SPE analytical models can usually be built only by imposing some structural restrictions on the original system model, depending on the specific modelling formalism as analytical models have often a limited expressiveness. There are many cases in which the significant aspects of the system cannot be effectively represented into the performance model. The SPE techniques are not widespread since they require the software engineers to learn new and non-standard modelling formalisms and notations (Marzolla, 2004). In SPE integration between the early calculations (e.g. by models) and the later measurements is elusive. SPE is constrained by the tight project schedules, poorly defined requirements and over-optimism about meeting those (Woodside et al., 2007).

2.3 Architecture and Pattern Based Methodologies

Performance analysis methods have been derived based on the specific classes of systems, identified by their architectural patterns. Architectural patterns characterize frequently used architectural solutions ((Balsamo et al, 2004). The first approach to deal with CB SA investigated the design and performance modelling of component interconnection patterns, which define and encapsulate the way client and server components communicate via connectors (Gomaa et al., 2000; Gomaa et al., 2001). Instead of proposing a transformational methodology, the pattern is described through Class and Collaboration diagrams and directly shows their corresponding EQN models. Three conceptually similar approaches are proposed where SA is described through architectural patterns such as pipe and filters, client-server, broker, layers, critical section and master-slave, whose structure is specified by UML Collaboration Diagrams and whose behaviour is described by Sequence or Activity Diagrams (Petriu et al., 2000; Petriu et al., 2002; Gu et al., 2002).

An integrated coverage of performance modelling, workload forecasting, load testing and benchmarking of web applications has been carried out (Menasce et al., 2002). The perception of performance from web infrastructure, server architecture and network point of view has been explained. As partitioning the workload is key to the precision of performance analysis and prediction, steps to characterize and partition the workload are described in details. The motivation for partitioning the workload is two-fold - improve representativeness of the characterization of the class of workload and increase the predictive power of the model.

Partitioning techniques divide the workload into a series of classes such that their populations are formed by quite homogeneous components. The aim is to group similar components, which improves the precision of the analysis to a great extent. (Menasce et al., 2002) discusses some of the attributes used for partitioning the workload such as:

- *Resource Usage* – types of resources used can be used effectively to classify transactions
- *Applications* – a workload may have its components grouped according to the application they belong to. However, the existence of very heterogeneous components within the same application poses problem with the choice of this attribute.
- *Objects* – a workload may be divided according to the type of objects handled by the applications
- *Geographical Orientation* – Due to inherent delays involved in networks, it is important to distinguish between local and remote requests or transactions.
- *Functional* – The components of a workload may be grouped into classes according to the functions being serviced.
- *Organizational Units* – A workload can be divided into classes taking the organizational structure as an attribute of similarity, such as finance, marketing and manufacturing
- *Mode* – The mode of processing or the type of interaction with the system may be used to categorize the components of a workload.

High performance website design techniques involving redundant hardware, load balancing, web server acceleration and efficient management of dynamic data have been explored to address the issue of performance degradation under heavy load (Iyengar et al, 2000). Different types of caching techniques (server side and client side) to boost performance under increased request load have been explained.

A method to automate the extraction of architecture level performance models of distributed CB systems using run-time monitoring data is devised by combining existing techniques such as call path tracing and resource demand estimation to an end-to-end model extraction method (Brosig et al., 2011). The method is validated with the representative, industry-standard SPECjEnterprise2010 benchmark application (Spec, 2011) in realistically scaled deployment environments. The method uses Palladio Component Model (PCM) as an architecture-level performance modelling formalism to describe extracted models. PCM is a domain specific modelling language for describing performance relevant aspects of CB software architectures (Becker et al., 2009). PCM provides clear separation between i) the implementation of software components, ii) the external services used by components, iii) the component execution environment and iv) the system usage profile. Separation of these concerns is a key advantage of PCM over other architecture-level performance models (Koziolek, 2010) such as SPT and MARTE profiles (OMG, 2006), CSM (Petriu et al., 2007) or KLAPER (Grassi et al., 2008).

PCM instances are mapped to a prototype implementation executable on a Java EE application server in (Becker et al., 2008). The results demonstrated that the effects, which are hard to predict on the model level can be revealed using the generated prototype. The mapping facilitated software architects to assess the performance of created architecture designs under more realistic conditions compared to performance analysis models which have to rely on simplifying assumptions.

An approach to automatically improve software architectures with respect to performance, reliability and cost is presented by (Martens et al., 2010). Using this approach, the design space spanned by different design options (e.g. available components and configuration options) can be systematically explored using meta-heuristic search techniques. Based on an initial architectural model of a system, new candidates are automatically generated and evaluated for the quality criteria. However, the approach is time consuming and doesn’t guarantee globally optimal solution. For the results, uncertainty of estimations, uncertainty of the workload and the resulting risks are not taken into account.

For distributed applications, a novel approach to performance testing based on selecting performance relevant use-cases from the architecture designs and instantiating and executing them as test cases on the early available software is described (Emmerich et al., 2004). The core hypothesis of the approach is that the performance of a distributed application can be successfully tested based on the middleware and/or off-the-shelf components that are available in the early stages of the software process. It indicated important directions towards engineering such approach like classification of performance-relevant distributed interactions as a base to select architecture use-cases and the investigation of software connectors as a mean to instantiate abstract use-cases on actual deployment platforms. The approach reported on experiments that showed that the actual performance of a sample distributed application is well approximated by measurements based only on its early available components.

For large scale component oriented enterprise applications, the problem of automatic compositional analysis of non-functional properties of a component assembly, with a focus on performance properties is dealt in (Grassi et al., 2004). Building on some of the existing methodologies and techniques, it suggested a path toward the automatic analysis of performance properties of a component assembly and its integration in the process of automatic components discovery and composition. A definition of a machine-processable and interoperable notation for representing performance-related properties of components and of their composition, expressed in a way that supports compositional performance analysis is identified. To give it performance semantics, definition of a mapping of this notation to a performance model is formulated as well. Finally, a definition of algorithmic methods for the prediction of performance properties of a component assembly on the basis of the previously defined component notation and semantics is proposed.

An assembler tool and a methodology to automatically generate performance models for component based systems have been explored in (Wu et al., 2003). For component based software system's performance prediction or evaluation, components are expressed as a type of LQN sub-model. The binding definition for the overall system is expressed as another LQN, with additional information to define parameters and bindings. The component sub-model is incorporated into a software product in the form of a sub-system. The various ways in which an LQN component that corresponds to a software component or sub-system may be derived is explained. In order to assemble these sub-models, a high level assembly model, which can be derived from the software architecture of the system, is defined. The system performance model is created from the component assembly model and the component sub-models. This is done in an automated process by a software tool called the "Component Assembler" which generates the task (object) instances and their parameters, guided by the binding section of the assembly model.

A framework integrating three modules for monitoring, modelling and performance prediction of component based enterprise systems is proposed to automate the discovery of performance problems (Mos et al., 2002). The monitoring module is responsible for obtaining information from a running component-oriented application. The technology used by the monitoring module is Java Management Extensions (JMX), which offers a lightweight, standardized way for managing Java objects (Fleury, 2002). The information is used by the modelling module to generate models of the application with associated performance information. The models can be traversed in a Model Driven Architecture (MDA) (Object Management Group, 2001) manner at different levels of abstraction. The performance prediction module uses the generated models to perform simulations that are used to predict the performance of the system when workload or design changes.

A component based mark-up language (CBML) is proposed to describe the performance models of software components and component-based systems (Wu et al., 2004). The language enables capturing the performance related features of the software components, their integration and deployment in the system and variations between alternative components in a product line. A model assembler tool is used to generate performance models automatically using a library of component sub-models. The performance components are reusable like the software components themselves. The combination of the performance modelling and components described at an architectural level, as in CBML, was deemed to be well suited for the analysis of software product lines and other kinds of component-based systems.

Reusing an existing component on different execution platforms requires repeated measurements of the concerned component for each relevant combination of execution platform and usage profile, leading to high effort. A novel integrated approach is presented that overcomes this limitation by reconstructing behaviour models with platform-independent resource demands of byte-code components (Kuperberg, 2008). The reconstructed models are parameterised over input parameter values. Using platform specific results of bytecode benchmarking, the method is able to translate the platform-independent resource demands into predictions for execution durations on a specific platform. To address the problem of anticipating the performance of the eventual large scale enterprise solution built on component technology, a method is proposed to determine the performance characteristics of component based applications by benchmarking and profiling (Chen et al., 2005). A model is constructed to act as a performance predictor for a class of applications based on the specific component technology.

Focussing on Message Oriented Middleware (MOM), a formal relationship is established between generated performance models and the generated code, a design and application process for parametric performance completions is introduced and a parametric performance completion is developed according to the proposed method (Happe et al., 2010). Coupled transformations formalise the relation between generated code and performance models and limit the design space to a restricted set of features specified in the transformation’s mark model. Chains of transformations realise these options deterministically. Using the knowledge about the deterministically generated code, coupled transformations can generate performance models based on the same restricted set of features.

An approach is proposed to support software architects to make early architectural choices during the design phase for a component-based, container hosted application to achieve performance goals (Fekete et al., 2005). It derives a quantitative performance model for the design, with parameters reflecting properties of the component container and platform. These parameters can be measured by running a simple benchmark application on the platform. The models developed can be applied to different EJB containers and a performance profile can be used to predict performance of different applications being executed on the same container. The infrastructure model, the architecture model and the performance profile of the platform are all extensible, which can be developed to capture future changes in container architecture.

2.4 Methodologies Based on Trace Analysis

An approach to performance analysis from requirements to architectural phases of the software life cycle is presented in (Petriu et al., 2002b). LQN performance models are derived from system scenarios described by means of Use Case Maps (UCM) (Buhr et al., 1996). The UCM specification is enriched with performance annotation. The derivation of LQN models from annotated UCM is defined on a path by path basis, starting from the UCM start points. The approach describes an algorithm which derives the component interaction types from the UCM paths by maintaining the unresolved message history while traversing a UCM path. A UCM2LQN tool automates the method and it is integrated into a general framework called UCM Navigator. This allows the creation and editing of UCM, supports scenario definitions, generates LQN models and exports UCM specifications as XML files. Further work is carried out to create performance models from scenarios to permit the earliest possible analysis of potential performance issues (Petriu et al., 2005). Scenario models like UMLs, Activity or Sequence Diagrams (SD) and UCMs, which capture the causal flow of intended execution and the operations, activities or responsibilities which may be allocated to components with their expected resource demands, are automatically transformed into performance models by the Scenario to Performance (S2P) algorithm. The LQNGenerator tool implements S2P to convert UCM and other scenario models to layered queuing performance models.

2.5 UML for performance

A methodology called PRIMA-UML is proposed, which makes use of information from different UML diagrams to incrementally generate a performance model representing a specified system (Cortellessa et al., 2000). PRIMA-UML is incremental in that it combines information extracted from and annotated into different UML diagrams to piecewise build the performance model. The methodology is open to embed information coming from other UML diagrams (possibly in late lifecycle phases) for detailing, refining or domain tailoring the performance model. Hence, it is not a black-box approach. Another formal approach is proposed to building LQN performance models from UML descriptions of the high level architecture of a system and more exactly from the architectural patterns used in the system (Petriu et al., 2000). The transformation from UML architectural description of a given system to its LQN model is based on PROGRES, a well known visual language and environment for programming with graph rewriting systems. An extension of UML to best model the possible adoption of mobility based paradigms in the software architecture of an application is introduced in (Grassi et al., 2001). A complete methodology is proposed that, starting from a software architecture described using the extended notation, generates a performance model (Markov Reward or Decision Process) that allows the designer to evaluate the convenience of introducing logical mobility into a software application.

A Core Scenario Model (CSM) is described (Petriu et al., 2004), which integrates the scenario and resource elements defined in a UML model with performance annotations, preparatory to generating performance models. It is based on and aligned with the UML Profile for Schedulability, Performance and Time (Object Management Group, 2002), and supports the generation of predictive performance models using queuing networks, layered queuing or timed Petri nets.

The work on CSM is extended to create a Performance by Unified Model Analysis (PUMA) interface, which provides a unified interface between different kinds of design information and different kinds of performance models like Markov models, stochastic Petri Nets and process algebras, queues and layered queues (Woodside et al., 2005). A CSM is described in (Petriu et al., 2007) that provides a meta-model for an intermediate form which correlates multiple UML diagrams, extracts the behaviour elements with the performance annotations, attaches important resource information that is obtained from the UML and facilitates the creation of many different kinds of performance models. The CSM can be transformed into various types of performance models like LQNs, QNs, stochastic Petri nets and process algebras.

A framework is introduced to transform source software models into target performance models (D’Ambrogio, 2005). The transformation requires a clear understanding of the abstract syntax and semantics of both the source and target models, which is obtained by use of meta-modeling techniques for defining the abstract syntax of models, the interrelationships between model elements and the model transformation rules. The method is founded on the precepts introduced by MDA and makes use of the set of related standards like Meta Object Facility (MOF), Query, Views and Transformation

(QVT) and XML Meta Interchange (XMI). The framework allows obtaining a high degree of automation, so that interoperable model transformation tools can be implemented in a timely and efficient way, leading to improvements in terms of software designer's productivity and system quality.

2.6 Process Algebra Based Approaches

Several Stochastic extensions of Process Algebras (SPA) (Hermanns et al., 2002) have been proposed in order to describe and analyze both functional and performance properties of software specifications within the same framework (Balsamo et al, 2004). All of these associate exponentially distributed random variables to actions and provide the generation of a Markov chain out of the semantic model of a system. Besides exponential actions, passive and immediate actions are also considered. PEPA nets – coloured stochastic Petri nets, which is a modelling formalism to clearly capture important features such as location, synchronisation and message passing and a platform support for software performance modelling using the PEPA nets is described in (Gilmore et al., 2004). Design decisions such as the placement of software components on hosts, the separation of client functions from server functions and movement of code and data across different hosts are typically needed to improve system reliability and performance. Software performance modelling is challenging for different reasons (Marzolla, 2004). It is difficult to derive meaningful performance measures from static analysis of code. Moreover, software performance modelling cannot be performed on one component at a time, as critical issues may arise only when different components interact. The drawback of many effective state-of-the-art performance analysis tools which do not differentiate between simple local communication and the migration of processes which may change the allowable pattern of communication is addressed by PEPA nets modelling language (Gilmore et al., 2003). PEPA nets extend the PEPA stochastic process algebra (Hillston, 1996) by allowing PEPA process algebra components to be used as the tokens of a coloured stochastic Petri net.

Work has been done to introduce stochastic probes as a means of measuring the soft performance characteristics over software systems (Argent-Katwala et al., 2004). Soft performance bounds like passage-time quantile, transient constraint and steady-state measure are an integral part of software and system performance validation. A regular expression language is presented, which specifies the stochastic probe and is then itself converted into a stochastic process algebra component. This is combined with the original SPA model (PEPA used) and analysed to provide soft performance and reliability bounds. Probes in effect partition the model state space into states where the probe measure has been started and states where it is stopped. This convenient partition allows the three types of soft performance analysis – transient, passage-time and steady-state to be expressed in a unified manner. As an extension of this work, a functional performance specification language (FPS) is developed (Bradley et al., 2006). It allows the modeller to derive quantitative performance functions using stochastic probes.

In the context of PEPA, a mechanism is defined for specifying performance queries which combine instantaneous observations of model states and finite sequences of observations of model activities (Clark et al., 2008). These queries are realised by composing the state-aware observers called eXtended Stochastic Probes (XSP) with a model expressed in stochastically-timed process algebra. The approach allows the modeller to refer to the states of components which are located in a hierarchically-structured performance model expressed in the stochastic process algebra PEPA.

PEPA is used as an intermediate language in a performance modelling approach, which facilitates the efficient solution of models being extracted from high-level system descriptions (Gilmore et al., 2005). UML is used to describe the high-level design. The technology which provides the efficient representation capability for the underlying performance model is the Multi-Terminal Binary Decision Diagram (MTBDD) based PRISM probabilistic model checker. The UML models are compiled through PEPA before translation into MTBDDs for solution. A component-based method of linking a collection of software tools to facilitate automated processing of UML performance models is described. The connectors in this method are the extractors and reflectors which have been developed. This approach is an attempt in some way to circumvent the unnecessary notational hurdle acting as an impediment to the understanding and uptake of modern performance analysis technologies. However, process algebra based approaches have their own problems. From the performance evaluation viewpoint, the analysis usually refers to the numerical solution of the underlying Markov chain which can easily lead to numerical problems due to state space explosion. On the software side, the software designer is required to be able to specify the software system using process algebras and to associate the appropriate performance parameters to actions (Balsamo et al, 2004). One significant practical problem with this approach is that an inexperienced modeller will not be able to use the system to compute any performance measure that they wish without any understanding of the abstraction, modelling and mathematical analysis at work in performance prediction and estimation (Gilmore et al., 2005).

This thesis tries to address this very crucial issue of software engineers and developers requiring understanding of the abstraction, complex modelling, mathematical analysis and unnecessary notational languages for performance evaluation.

2.7 Petri-Net Based Approaches

Like SPA, Stochastic Petri Nets (SPN) are usually proposed as a unifying formal specification framework, allowing the analysis of both functional and non-functional properties of systems (Balsamo et al, 2004). A tool called ArgoSPE is introduced in (Gomez-Martinez et al., 2006). It implements a performance evaluation process that builds on the principles of the SPE. It translates some performance annotated UML diagrams into SPN models and therefore prevents software engineers to model with SPN since they are obtained as a by-product of their UML models. The design of the tool follows the architecture proposed by OMG in the UML Profile for Schedulability, Performance and Time specification.

Queuing Petri Net (QPN) is a general-purpose modelling formalism at a lower level of abstraction compared to PCM that has lent itself well to modelling and analyzing the performance of distributed component-based systems (Kounev et al., 2003). A formal mapping approach from PCM to QPN models implemented by means of an automated model-to-model transformation as part of a new PCM solution method based on simulation of QPNs is presented in ((Meier et al., 2011). An automatic transformation from PCM to QPNs in the form of a new PCM solver tool is implemented. A practical performance modelling methodology is presented in (Kounev, 2006), which helps to construct models that accurately reflect the system performance and scalability characteristics. Taking advantage of the modelling power and expressiveness of QPN, the approach makes it possible to model the system at a higher degree of accuracy.

As UML2 activities are based on Petri net like semantics, an approach to formally map UML2 activities into Petri nets or Petri net semantics from a theoretical, practical and operational points of view is formulated (Staines, 2008; Staines, 2010). Petri Net diagrams are complex, contain more nodes and edges than UML 2 activities and are unsuitable for visualization by stakeholders. To address this problem, the UML Activity diagram is translated into a Fundamental Modeling Concepts Petri net diagram compact notation. This is then converted to a coloured Petri net for execution and validation.

A timed Petri Net is derived by creating subnets for the individual activity steps with labels that allowed composition of the fragments of scenarios, bottom-up, to arrive at a model for the entire behaviour (Lo’pez-Grao et al., 2004). However, this approach is suitable only for the particular style of Petri Nets and does not address processor contention.

2.8 Methodologies Based on Simulation Methods

Simulation performance models allow for unconstrained representations of software models (Marzolla, 2004). This is not always easy to achieve with analytical models. In QN models it is difficult to handle situations arising from finite capacity of queues and subsequent blocking behaviour. Only approximate techniques can be used in some cases and simulation is the only approach in general (Balsamo et al., 2003). Other difficulties arise when analyzing simultaneous resource possession, fork and join systems, synchronous versus asynchronous communications and many queuing disciplines. An automatic generation of simulation performance models from high-level UML descriptions of SA is presented in (Marzolla, 2004). The approach considers UML diagrams annotated with a subset of the UML Performance Profile (Object Management Group, 2002). An almost one-to-one mapping between UML elements and simulation processes is defined. As a result, the structure of the simulation model is very similar to the structure of the software model. The drawback of this approach is its dependency on UML modelling. UML is only informally defined, so that software modellers may use different diagrams for the same purpose, or use the same notation with different implicit meaning (Marzolla, 2004). Also, approaches which try to apply performance analysis very early, typically at the software architecture and design level still require much detailed information from the implementation phase to carry out performance analysis (Balsamo et al, 2004).

This thesis attempts to address this issue and focuses on the implementation phase directly.

2.9 Kalman Filter, Support Vector Machine and Kernel methods

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error (Kalman, 1960). The filter estimates a process by using a form of feedback control. It estimates the process state at some time and then obtains feedback in the form of noisy measurements. There are two types of equations that are used by the filter – *time update* equations and *measurement update* equations. The time update equations project forward in time the current state and error covariance estimates to obtain the *a priori* estimates for the next time step. The measurement update equations provide feedback for incorporating a new measurement into the *a priori* estimate to obtain an improved *a posteriori* estimate. The time update equations are a kind of predictor equations for the process state while the measurement equations are the corrector equations. However, the Kalman filter is an optimal filter and limited to linear and Gaussian assumptions (Di et al., 2008). Thus the system dynamics and measurement model must be linear in order to use the Kalman filter to track a target. Kalman Filters are optimal filters and are limited to linear and Gaussian assumptions only. This linearity constraint limits the usefulness of the Kalman filter since most of the real systems are non-linear in nature.

In supervised learning a set of examples of input vectors along with corresponding targets are given, which may be real values (in regression) or class labels (in classification). From this training set, we learn a model of the dependency of the targets on the inputs with the objective of making predictions of the target for previously unseen values of the input vectors (Tipping, 2000). In real-world data, the presence of noise (in regression) and class overlap (in classification) implies that the principal modelling challenge is to avoid 'over-fitting' of the training set. An approach to supervised learning is the Support Vector Machine (SVM) (Vapnik, 1998). It makes predictions based on a function of the form:

$$y(x) = \sum_{n=1}^{N} \mathrm{w_n K(x, x_n) + w_0} \qquad (2.1)$$

where $\mathrm{w_n}$ are the model weights and K(.,.) is a *kernel* function. The key feature of the SVM is that in the case of classification, its target function attempts to minimise the number of errors made on the training set while simultaneously maximising the margin between the two classes in the feature space implicitly defined by the kernel. However, the support vector machine exhibits significant disadvantages (Tipping 2000): 1) The predictions are not probabilistic, 2) SVMs make liberal use of kernel functions, the requisite number of which grows steeply with the size of the training set, 3) The cross-validation procedure that is necessary to estimate the error/margin trade-off parameter (in regression the insensitivity parameter) is wasteful both of data and computation and 4) The kernel function K (.,.) must satisfy Mercer's condition (Burges, 1998).

A kernel based language KLAPER (**K**ernel **LA**nguage for **PE**rformance and **R**eliability analysis), whose main goal is to act as a bridge between design models of component-based applications based on heterogeneous notations and performance or reliability analysis models is presented in (Grassi et al., 2005). KLAPER is defined as a Meta Object Facility (MOF) metamodel to exploit MOF-based model transformation frameworks. It facilitates the integration of KLAPER based analysis tools within the MDA approach to software design. KLAPER captures the relevant information for the analysis of non-functional attributes of component-based systems with a focus on performance and reliability, abstracting away unnecessary details. The direct transformation from source to target models is split in a two-step transformation: 1) from the source model to an intermediate model expressed using KLAPER and then 2) from the intermediate model to the target model. With conventional model transformation, if there are N different notations used to model components and their composition, and M different target notations, N·M different model transformations are needed. With KLAPER, only N+M transformations are needed instead of N·M transformations. The transformations are also simpler to devise than direct transformations as the intermediate model is likely to be closer to both the source and target models. However, kernel based algorithms are computationally very demanding and in certain cases working with explicit feature space with the standard learning algorithms may be beneficial over kernels (Grassi et al., 2005).

2.10 Standard Performance Measurement Tools and Techniques

The tools used by performance analysts range from load generators for supplying the workload to a system under test, to monitors for gathering data as the system executes (Woodside et al., 2007). Monitoring itself can be performed through hardware, software or a combination of the two. Monitoring can be broken down into two broad categories:

Instrumentation – it is the insertion of probes into a system to measure some sort of events. Some instrumentation is usually built into the host operating system and minimally indicates the utilization of the various devices including the CPU. Other instrumentations are added manually to applications. Frameworks such as the Application Response Measurement (ARM) (Johnson, M. W., 1998) are beneficial as they form a common platform to which disparate programs can gather performance information. Instrumentation can be added automatically as well. Aspect-Oriented programming can be used to automatically insert instrumentation code into applications (Debusmann et al., 2003). Quantify (IBM, 2002) adds probes at the beginning and end of basic blocks of object code to count the number of cycles executed. The Paradyn tool (Merson et al., 2005) carries this one step further by instrumenting the actual executables dynamically. Monitoring introduces system overheads. However, to reduce the overhead of monitoring system requests, two orthogonal concepts exist: (i) *quantitative throttling*: throttling the number of requests that are actually monitored (Gilly et al., 2009) and (ii) *qualitative throttling*: throttling the level of details for the requests that are being monitored (Ehlers et al., 2011).

Developers and testers use instrumentation tools to help them find problems with systems. However, users depend on experience to use the results. Better methods and tools are a future requirement for interpreting the results and diagnosing performance problems (Woodside et al., 2007). This requirement is one of the aspects addressed by this thesis.

Profiling – A program profile is a histogram of the execution of a program (Knuth, D. E., 1971). Profiles can be generated using instrumentation through statistical sampling or by running the program on a virtual machine and counting the execution of the actual instructions (Nethercote et al., 2003).
One key problem with these performance tools is that these are not well established at earlier stages in the software life cycle. Some of the obstacles to the adoption of these tools are (Malony et al., 2001):

a) A lack of theoretical justification for the methods for improving performance that work and why they do so. The measurement data collected requires expert interpretation to fix the problems.

b) A conflict between automation and adaptability. Systems which are highly automated but are difficult to change and vice versa. As a result no tool does the job the user needs. So the user invents one. Also, various tools have various forms of output which makes interoperability quite challenging.

This thesis attempts to alleviate the problem of leaving the collected measurement data to subsequent expert interpretation by attempting to formulate an approach applicable during the implementation phase of the development life cycle.

A combination of model configuration, automated measurements, statistical inference and model based performance prediction in order to support performance evaluations during the evolution of the software system is proposed in (Westermann et al., 2010). The run-time monitoring approach uses systematic measurements to build mathematical models or models obtained with genetic optimization. The models serve as interpolation of the measurements. The main idea of the approach is to abstract from system internals by applying a combination of systematic goal-oriented measurements, statistical model inference and model integration. The measured functional dependencies are integrated in the PCM. However, this approach intrinsically has a few drawbacks. One of the major problems in the area of software system measurement is a lack of discussing the meaning of the numbers produced by measures and the inappropriate statistical operations on those systematic measures (Zuse, 1998).

Intuitive interpretation of the measure values is applied. If there do not exist more information about how the numbers are produced, it is difficult to interpret them. Behind every software measure (resource, process or product measures), a qualitative model is hidden. Doing software measurement we have to quantify qualitative properties of objects in reality. Behind every measure, models of qualitative attributes are hidden (Zuse, 1998). In a software application it can be quite challenging to ascertain how the various processing activities are contributing to the measured systematic data. It takes a lot of effort, time and most importantly the right technical insight to identify these hidden attributes. The approach does not consider the internal structure of the underlying system but focuses on the observable data. A "black-box" specification of performance characteristics is used i.e. the performance of the system is captured by a function of its usage. These black-box performance models do not contain any information about the system's internal structure. The other significant limitation of this approach is that it integrates into the PCM. Depending on the approach, it may not follow an industry standard and therefore widespread use may be a long term goal. Existing UML tools cannot be used to create PCM instances. The learning curve for developers familiar with UML is potentially higher. Existing UML models are not supported by the PCM analysis tools and have to be transformed to PCM instances to carry out performance predictions. Implementing such transformations is complicated (Becker et al., 2009).

2.11 Continuous Performance Management

Typically, preliminary performance profiling of an application is done by using synthetic workloads or benchmarks which are created to reflect a "typical application behaviour" for "typical client transactions" (Cherkasova et al., 2007). While such performance profiling can be useful at the initial stages of design and development of a future system, it may not be adequate for analysis of performance issues and observed application behaviour in existing production systems. Frequent software releases and application updates make it extremely difficult and challenging to perform a thorough and detailed performance evaluation of an updated application. When poorly performing code slips into production and an application responds slowly, the organization inevitably loses productivity and experiences increased Opex. The traditional *reactive* approach to performance analysis is to set thresholds for observed performance metrics and raise alarms when these thresholds are violated. It is acknowledged that this approach is not adequate for understanding the performance changes between application updates. Instead, a *proactive* approach that is based on upfront continuous application performance evaluation may assist enterprises in reducing loss of productivity by time-consuming diagnosis of essential changes in application performance (Cherkasova et al., 2007).

CPM implements performance and scalability testing within a CI environment. This allows software engineers to test beyond the component level, such as testing integration of components into a working solution by tracing a request as it passes between multiple Java Virtual Machines (JVMs). Automated load tests can be run within the CI test harness to baseline and track the application's scalability during development (Haines, 2008). JUnitPerf covers performance analysis of component level unit tests. Load testing requires a different test bed because these are more like business use cases rather than functional tests. The HttpUnit extension of JUnit is suitable here.

However, although in CPM, performance is assessed continuously during CI, it still follows the traditional *reactive* approach to performance analysis by setting thresholds for observed performance metrics and raise alarms when these thresholds are violated. These types of tests do not provide insight into how the internal processing activities impact the timeliness of responses of the components or operations supported by multiple components. To achieve this and take fullest advantage of CPM, scriptable performance analysis tools like code profilers and memory debuggers are needed (Haines, 2008). That is, an engine that will run the performance, integration, scalability tests and capture the results is needed. Profilers and debuggers are not used in production environments. However, while measuring the application, profiling and monitoring tools affect performance (Duggan et al., 2011) by adding overhead to the application being measured and to the machine it is running on. The amount of overhead depends on the type of the profiler. In the case of a performance profiler, the act of measurement may itself impact the performance being measured. This is particularly true for an instrumentation profiler, which has to modify the application binary to insert its own timing probes to every function. As a result there is more code to execute, requiring additional CPU and memory, causing increased overhead. The profiler also has to deal with lots of data and for a detailed analysis it may require a lot of memory and processor time just to cope. If the application is already memory and CPU intensive, things will only get worse and it could be that it is just not possible to analyse the application properly (Farrell, 2010). To test performance of functional units of code, tools like JUnitPerf are used. It is a collection of JUnit test decorators used to measure the performance and scalability of functionality contained within existing JUnit tests (Clark, 2009). These tests verify the performance of the functional units as a black box and performance of any operation involving multiple units of code cannot be ascertained. Also, in JUnitPerf, the decoration on top of JUnit tests introduces significant overheads. The elapsed time measured is not reflective of the actual elapsed time of the method (Clark, 2009). These tests are also not intended to be a full-fledged load testing or performance profiling tool. Tools like LoadRunner and JMeter are used to test applications under externally generated load (Hansen, 2011). But these tests are reactive in nature as these set observed performance thresholds and flag alerts when these thresholds are violated. These neither help to understand how each of the low level granular application processing activities impacts the overall performance of the higher level operation nor do they facilitate proactive fine-tuning of the various application processing activities to achieve a pre-specified target performance level.

To address the issue of continuous performance assessment, a novel framework for automated anomaly detection and application change analysis is presented by (Cherkasova et al., 2007). It is based on a regression-based transaction model that reflects a resource consumption model of the application and an application performance signature that provides a compact model of run-time behaviour of the application. Through the resource consumption model, the framework detects anomaly in CPU utilization across different transactions. The application performance signature facilitates identification of unusual variations to transaction service times with changes to the application. A very linear, sequential approach is used to compute the overall latency of a transaction, which relates to the transaction's performance:

$$R_i = R_i^{front} + R_i^{DB} \qquad (2.2)$$

where R_i is the observed overall latency of the i-th transaction, R_i^{front} and R_i^{DB} are the observed average latencies for the i-th transaction at the front application server and the database server. However, this doesn't delve into the different application processing activities that may be running inside the application server. Also, the approach is very sequential. It does not take into account the possibility of parallel application processing where the overall transaction latency of the operation may not be a linear sum of all the individual latencies introduced by different processing activities. Also, depending on the location of the servers, there may be some non-negligible network latency, which needs to be factored in. The focus of their work is on server level (App Server and DB server) transaction latencies and identifying unusual variations to overall transaction service times with changes to the application. It does not delve into the details of how the various lower level transaction processing activities impact the service time of the transaction. Application programs introduce load on the system resources and introduce performance problems if they make poor use of system resources, generate undue network traffic or create unnecessary contention in the system resources (IBM, 2011). The approach proposed does not address this issue.

2.12 Chapter Summary

This chapter provided a critical overview of the research that have been undertaken towards various modelling approaches including SPE techniques, Queuing Networks and their various extensions, Stochastic Process Algebra, Stochastic Petri Nets, simulation based methods and other techniques.

It is acknowledged that the traditional *reactive* approach to performance analysis by setting thresholds for observed performance metrics and raise alarms when these thresholds are violated is not adequate for understanding the performance changes between application updates. Instead, a *proactive* approach that is based on upfront continuous application performance evaluation may assist enterprises in reducing loss of productivity by time-consuming diagnosis of essential changes in application performance.

Many weaknesses in the current performance modelling processes are highlighted. They require heavy effort, which limits what can be attempted. Measurements lack standards. There is a semantic gap between performance concerns and functional concerns, which prevents many developers from addressing performance at all. For the same reason many developers do not trust or understand performance models, even if such models are available. Performance modelling is costly and approximate. They leave out details that may be important and are difficult to validate (Woodside et al., 2007). Building models that accurately capture the different aspects of system behaviour is a challenging task and requires a lot of time and effort when applied to large real-world systems (Cortellessa et al., 2005). Given the costs of building performance models, techniques for model extraction based on observation of the system at run-time are highly desirable. Current performance analysis tools mostly focus on profiling and monitoring transaction response times and resource consumption. They often provide large amounts of low-level data while important information about end-to-end performance behaviour is missing (Brosig et al., 2011). Modern performance analysis technologies often introduce unnecessary notational hurdle, which act as an impediment to the understanding and uptake of those. Inexperienced modellers are not be able to use most of the methods to compute any performance measure that they wish without any understanding of the abstraction, modelling and mathematical analysis at work in performance prediction and estimation (Gilmore et al., 2005).

The citations above corroborate my industry experience where I have witnessed application developers often find it more convenient to monitor and analyze application level outputs rather than system resource or service level diagnostics.

Automatic compositional analysis of non-functional properties of a component assembly, with a focus on performance properties has been dealt. However, analysis of the impact of the various lower level processing activities of the components on the service time of an application operation remains to be explored. Profilers may be used for this purpose but deeply affect performance by adding overhead to the executing application it is measuring and to the machine it is running on. If the application is already memory and CPU intensive, things will only get worse and it could be that it is just not possible to analyse the application properly. It is highlighted that preliminary performance profiling of an application by synthetic workloads or benchmarks created to reflect a "typical application behaviour" for "typical client transactions" can be useful at the initial stages of design and development of a future system, but may not be adequate for analysis of performance issues and observed application behaviour in existing production systems. It is an acknowledged fact that the lack of performance requirement validation in current software practice is mostly due to the knowledge gap between software engineers/architects and quality assurance experts. To add to the complexity, frequent software releases and application updates make it extremely difficult and challenging to perform a thorough and detailed performance evaluation of an updated application (Cherkasova et al., 2007). Slippage of poorly performing code into production will result in the organization losing productivity and experiencing increased Opex.

Review of the merits and demerits of the approaches highlights the need of a simple, proactive approach to evaluate the performance of existing software applications undergoing frequent updates, which will not require repetitive load/performance testing, complex SPE techniques, performance models and resource intensive application profilers. The approach will not require the software engineers to learn new, non-standard modelling formalisms and notations or to specify the software system using process algebras and complex mathematical analysis. Overall, the approach will not act as an impediment to the understanding and uptake of performance analysis technologies as highlighted by (Gilmore et al., 2005).

3 Performance Analysis Methodology – the Model and Methods

In this chapter, we further motivate our research problem, revisit the need for our approach and discuss the three hypotheses leading to the performance analysis methodology proposed in this thesis. We do so based on our simple example application, which allows us to characterise the various drawbacks with the prevalent approaches, which we attempt to address in this thesis. We discuss and justify the appropriateness of the method adopted for the research problem at hand and review the boundary assumptions for the applicability of the research. This chapter concludes by explaining how the proposed model and methods work in combination and the usage of the model and the methods during the Software Development Life Cycle (SDLC).

3.1 The need for a simple Methodology revisited

As highlighted in Chapter 2, many approaches focus on early lifecycle phases and assume that the software system is being built from scratch i.e. green-field scenario and all its details are known. These approaches neglect the influences of existing software. Detailed information about the internal structure, which is required for performance prediction, may not be available during the early phases. The review of the current performance engineering techniques in Chapter 2 emphasizes the need of a simple approach to evaluate the performance of *existing* software applications undergoing frequent updates, which shall not require repetitive load/performance testing, complex SPE techniques, performance models and resource intensive application profilers. The need to learn new, non-standard modelling formalisms and notations or to specify the software systems using process algebras and complex mathematical analysis acts as an impediment to the understanding and uptake of modern performance analysis techniques. A technique, simple enough to address these hurdles, needs to be proposed and evaluated.

As discussed in Section 2.11, poorly designed algorithms in application programs make poor use of system resources, generate undue network traffic or create unnecessary contention in the system resources. This introduces load on the system resources resulting in performance problems. In our example TCG application, the underlying TA and FA component algorithms catering to the getConfidence(…) operation of the existing TCG application were optimal enough to conform to the business requirement of an average response time of 0.5 second under an inbound workload of 10 requests/second. However, as the algorithms were changed, those became less optimal and couldn’t respond within the required 0.5 second threshold. Consequently, the automated nightly load tests failed resulting in rolling back of the changes and applying other variants of the algorithms. Neither of the approaches proposed by (Westermann et al., 2010) or (Cherkasova et al., 2007) attempts to analyse which low level application processing activity is causing the observed anomaly in the systematic data. The approaches also require additional system analyst resources to execute the performance assessment work.

While adopting the above mentioned approaches, this thesis aligns with the future direction as proposed by (Woodside et al., 2007). It states that complex software systems are usually built from components and the novelty will be to constitute a “horizontal” composition at the application level as opposed to a “vertical” composition of an application with its supporting platform. In our example of the TCG application, this translates to an approach which will enable SKGWorld Forex’s software developers to analyse the performance of the TA and FA components at the application layer without the need to delve into the underlying system resources, which most often will warrant more complex analysis techniques as discussed earlier.

This thesis proposes a combination of a model based with a measurement based approach, focussing primarily on the application layer of the system. Other than the application layer, the approach uses “black-box” performance models, which do not contain any information about the system’s internal structure and the performance of the system is captured by a function of its usage. The reason for this approach is manifold. Typically, the application components are often modified due to changes in business requirements while the underlying system comprising the operating system, hardware infrastructure and network remain unchanged. As illustrated in our example above, with the change in the global financial market’s sentiment, the algorithms and the associated business logic of the TA and FA components were required to be changed. No other aspect of the TCG application like the software containers, hardware, hosting environment etc. needed any change.

The IT department at SKGWorld Forex Ltd has a team of software engineers but does not have any dedicated performance experts with the knowledge of complex SPE techniques, process algebra, petri-nets, performance models or how to interpret the outputs of application profilers. The proposed approach will allow the software developers to assess the impact of the changes to an application operation or transaction upfront without the need to involve additional quality assurance experts. It will in some way attempt to circumvent the unnecessary notational hurdle, which acts as an impediment to the understanding and uptake of modern performance analysis technologies. No system level utilization diagnostics and measurement will be required for which the software developers may not have adequate expertise (Marz, 2005). This thesis addresses the need to associate the internal processing activities of the transaction of an application operation to its performance. The inbound workload is also taken into consideration as it is an important factor impacting the performance and the internal processing activities.

3.2 Admittance, Load Potential and Delay Points revisited

Conventionally there are two types of operations – synchronous and asynchronous (MSDN, 2014a). In a synchronous operation, the request to the operation blocks till such time the transaction is processed and response sent back to the consumer. In an asynchronous operation there is no such blocking. The response is sent back to the consumer at a later point in time and linked to the original request through a correlation identifier. In an asynchronous operation call, the consumer doesn't wait for the response to come back from the operation and continues with its other activities after posting the request to the operation. In synchronous operation calls, the consumer waits till such time a response is received from the operation called. Hence, performance in terms of timeliness of response is extremely critical for synchronous operation transactions.

This thesis focuses on synchronous operation transactions only. As the overall transaction latency of the operation may not be a linear sum of all the individual latencies introduced by different processing activities, this thesis attempts to decouple the *external* performance of an application operation from the latencies introduced by its *internal* transaction processing activities. This is achieved through the Application Operation Admittance ('Y'). The individual latencies of all the low level application processing activities required to process an operation (or transaction), cumulatively introduce an overall "Impedance" for that operation. This may or may not be a linear sum of the individual latencies. Admittance is the inverse of this Impedance created. It is defined as the measure of ease with which a request to an application's operation is processed and a response sent back.

In Section 1.4, we have discussed the concept of Stress Point and the significance of understanding how near the current workload is to this threshold from performance perspective. To enable taking into account both the inbound workload and the Stress Point, this thesis introduced the concept of Load Potential (‘V’) for an operation as explained in the same section.

The concept of Delay Point is also explained in Section 1.4. It is defined as any processing activity node of an application operation which introduces some latency (or delay) to the overall transaction. If we consider an application operation which warrants reading some data from the File System, performing some in-memory calculations, writing back some data to the disk before sending the response back, there are 2 types of Delay Points in action, which are File I/O and In-Memory Data Processing. In our example TCG application, the TA components read the business logic either from the files on the file system or from the database. Technical analysis is then performed in memory on the currency quotes applying the business logic and algorithms. Alongside passing the output to the Process Orchestrator or the next TA component as applicable, the output is also either stored back in the database or written to files on the file system. In this case, each of the TA components is performing some in-memory data processing, some file I/O and some database interactions. The nodes within each TA component where these processing activities are being performed are the Delay Points. In this example the types of the Delay Points are In-Memory Data Processing, File I/O and Database Interaction.

In production systems, application components can be huge and complex. However, every component supporting an operation can be decomposed into a collection of Delay Points each performing specific activities. The internal activities of these Delay Points interface with the system resources. One key objective of this thesis is to enable software developers to assess the impact of their changes at the application layer without the need to monitor system resources. That is why the Delay Points are treated as the boundary between the application layer and the systems/platform layer and the variations in their latencies are measured.

3.3 Performance, Admittance, Load Potential – the *PALP* Model

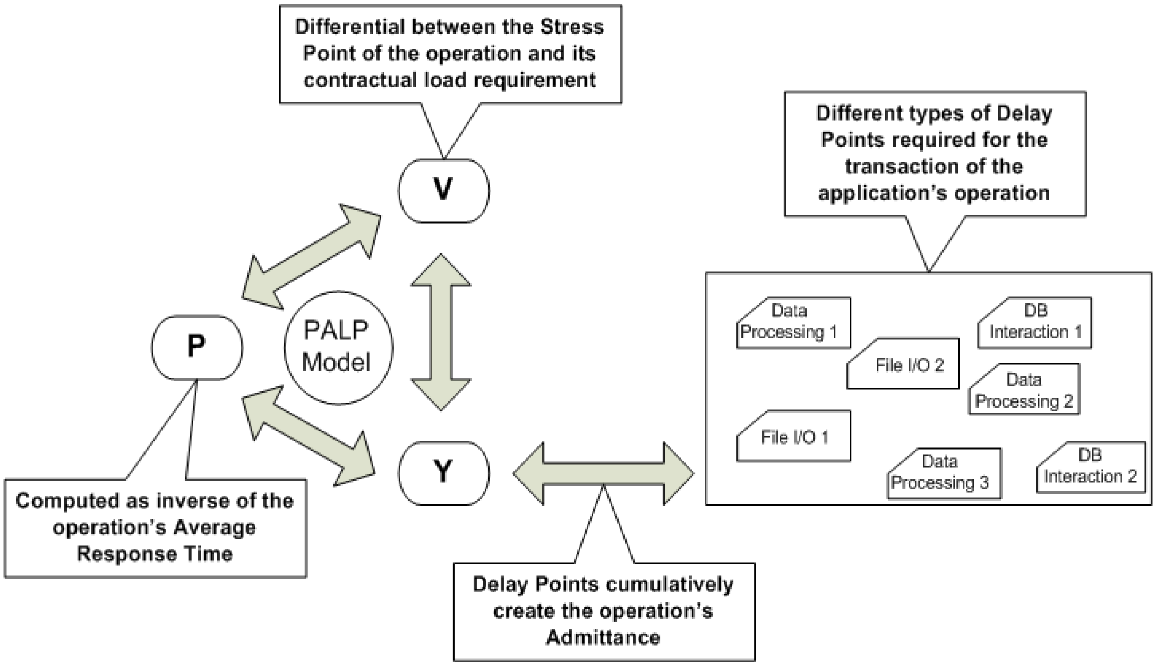

Differential between the Stress Point of the operation and its contractual load requirement
V
Different types of Delay Points required for the transaction of the application's operation
P
PALP Model
Y
Data Processing 1
DB Interaction 1
File I/O 2
Data Processing 2
File I/O 1
Data Processing 3
DB Interaction 2
Computed as inverse of the operation's Average Response Time
Delay Points cumulatively create the operation's Admittance

Figure 4: **Logical view of the PALP model**

As shown in Figure 4, to analyse an operation's performance from within an application layer, this thesis attempts to associate the three attributes namely the operation's Performance ('P'), its Admittance ('Y') and its Load Potential ('V') through a relational model. In a separate step, using this model and measuring the latencies of the different Delay Points of the operation through instrumentation probes, the thesis attempts to extract the mathematical functions in which the Delay Point latencies influence the operation's Admittance. The advantages of this two-stepped model and measurement based approach are manifold. At a high level (in a "black box" way), it will enable a flexible *three-way* predictive mechanism between the operation's Performance, Admittance and Load Potential in the following way:

a) To achieve a specified *target* Performance level of the operation under a given Load Potential (i.e. workload condition specified) and hosting environment, predict what should be the operation's Admittance and hence the Delay Point latencies? In this way, the application algorithm and configuration can be fine-tuned proactively during code development and modification instead of the traditional reactive approach to performance analysis by setting thresholds for observed performance metrics and raise alarms when these thresholds are violated. This reactive approach is followed in the current CPM paradigm as well. In the event of violation of performance thresholds, the application algorithm and configuration is revisited again and the cycle of code change, build, load testing and performance measurement follows again.

b) To achieve a specified target Performance level of the operation with a given operation Admittance (i.e. specific algorithm and configuration), predict what should be the ideal Load Potential (i.e. inbound workload assessment) on a given hosting environment?

c) For a given Load Potential and Admittance, predict what will be the expected Performance of the operation on a given hosting environment?

Caching may potentially influence the latencies of the Delay Points. Two types of caching are commonly used in applications – Memory Caching and Disk Caching (ITBusinessEdge, 2014). Different types of caching strategies may be adopted to improve the performance of application operations. Two methods commonly used are proactive data caching and reactive data caching (MSDN, 2014b). In proactive caching, the required data is loaded upfront when the system starts and maintained in the caches for the lifetime of the application or process. In reactive caching, data is retrieved and loaded to the cache when it is requested for the first time. Depending on the caching strategy it usually takes some time initially to load the caches. Due to this phenomenon, the timeliness of response to any particular application operation improves after the initial few requests. This thesis does not deal with any caching strategy explicitly. However, to address any caching in the background, the controlled experiments are run repeatedly for a number of times (optimal number determined through Standard Deviation of the response times as explained in Chapters 5 and 6) with the same application configuration and the average latency of each type of Delay Point is recorded. Details of the experiments are discussed in Chapters 5, 6 and 7.

In a decoupled way, as the operation’s Admittance is linked to the latencies of the various Delay Points, the approach advocated in this thesis will provide a *bi-directional,* bottom-up and top-down technique to performance analysis as shown in Figures 3 and 4.

Figure 5 shows how the model will enable *proactive* fine tuning of the algorithm and configuration of the application operation to attain a pre-specified *target* performance level during software development and modification instead of *reacting* to the observed data from repetitive performance testing and modifying the application code if required during CPM cycles. This is a *bottom-up* approach *targeting* a specific performance level.

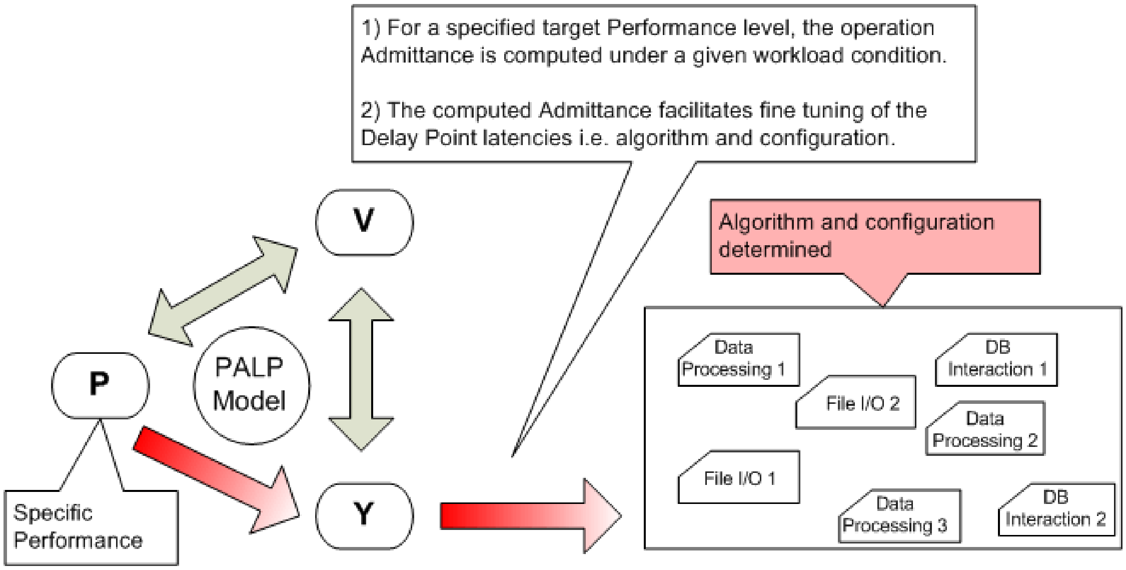
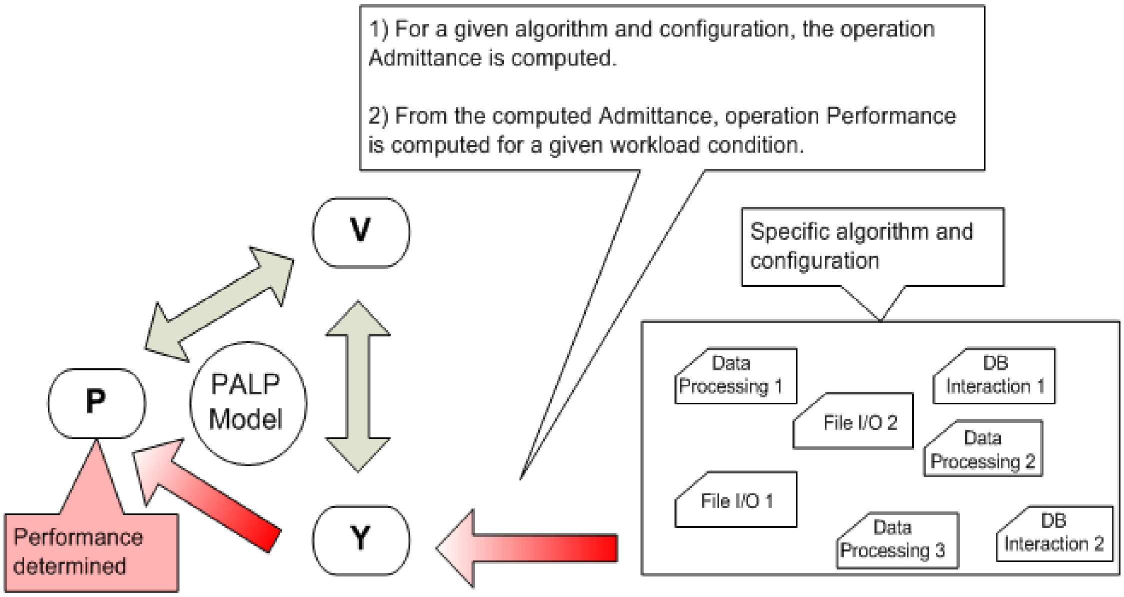

1) For a specified target Performance level, the operation Admittance is computed under a given workload condition.
2) The computed Admittance facilitates fine tuning of the Delay Point latencies i.e. algorithm and configuration.
Algorithm and configuration determined
V
P
PALP Model
Y
Specific Performance
Data Processing 1
DB Interaction 1
File I/O 2
Data Processing 2
File I/O 1
Data Processing 3
DB Interaction 2

Figure 5: Fine tuning the Algorithm for a specific Performance level

Consider the above capability in the context of our example TCG application. The software developers will not have to react to the failed automated nightly load tests by rolling back the changes and trying to fine tune the algorithms of the TA and FA components in a trial and error manner. In our example, for the sake of simplicity, we have shown only one operation namely getConfidence(...). However, in real life, the web service interface may have multiple operations exposed resulting in a lot more overhead for the build procedure. Applying this method, the software engineers will be able to determine the 'P' and 'V' specific to the getConfidence(...) operation from the mandated response time and the inbound workload respectively as explained previously. Then, using the PALP model, they can compute the 'Y' for the operation upfront from 'P' and 'V'. The computed 'Y' can then be used to determine what the latency (or latencies) of the modified Delay Points should be in order to achieve the mandated 'P' for the getConfidence(...) operation. This calculated latency will serve as the reference point for the actual measured latency of the modified Delay Points. The software developers will be able to fine tune the Delay Points so that the measured latency of the modified Delay Points is at the least equal to the calculated latency. This may be done through local systems testing by the software developers themselves without the need of the nightly build of the whole application, automated load tests, verification of the results and if failed, going through the cycle of undoing and redoing the algorithms. Detailed description of how the PALP model and the two proposed methods complement each other and work in combination is provided in Section 3.6 with the aid of Figure 7.

Figure 6 below shows how the model will allow assessing the Performance level under a given workload condition for a specific algorithm and configuration of the application operation. This can be achieved by only probing all the relevant Delay Points catering to the operation instead of execution of the full build of the application and subsequent load testing, which has its own overhead from a time, resource and cost perspective. This is a *top-down* approach.


1) For a given algorithm and configuration, the operation Admittance is computed.

2) From the computed Admittance, operation Performance is computed for a given workload condition.

Specific algorithm and configuration

Performance determined

Figure 6: Assessing Performance level for a specific Algorithm

In our example TCG application, if only the in-memory data processing Delay Points are modified and everything else remain as-is, the software developers will be required to probe the latencies of the data processing nodes only through local systems testing under the given inbound workload. Full build of the whole application and subsequent load tests will not be required. From the measured Delay Point latencies for the specific algorithm and configuration, the resultant ‘Y’ for the getConfidence(...) operation is calculated. This ‘Y’ is then fed into the PALP model to project the ‘P’ for the operation under the given conditions. This method is the reverse of the process explained above with the aid of Figure 5.

In both the scenarios as shown in Figures 5 and 6, the operation Admittance (‘Y’) acts as the bridge between the external Performance (‘P’) and the latencies of the internal Delay Points of the application operation.

3.3.1 Constancy of runtime Operation Admittance ‘Y’

Real workloads can be viewed as a collection of heterogeneous components (Menasce et al., 2002). Concerning the type and level of resource usage, a request involving a complex database query will differ significantly from a request involving a quick and simple in-memory data processing. Partitioning the workload and classifying the request types increases the predictive power of the model.

As highlighted in (Menasce et al., 2002), some of the attributes used to partition and classify request workloads are:

- *Resource Usage*
- *Applications*
- *Objects*
- *Geographical Orientation*
- *Functional*
- *Organizational Units*
- *Mode*

To classify the request type, we restrict our model to operations of an application. At a given time T_1, for a particular type of request to the same application operation, the request type, the process logic that is followed to serve the request, the system configuration, the resource requirements and the contract request load condition will be ideally the same as that at another time T_2. Below, we assess the classification of a request to the same application operation in light of the attributes prescribed above:

- *Resource Usage* – for a request to a particular application operation, the types of resources used will be the same

- *Applications* – the components used to process the request will be the same. Hence, the applications containing the components should be the same as well for all the requests.
- *Objects* – the type of objects handled by the applications needed to support the transaction for the operation will be the same.
- *Geographical Orientation* – the assessment of performance will be from the point of view of a particular client. Hence the geographical location of the request initiation should be the same. It should be either local or remote but should not vary during runtime.
- *Functional* – as the same application components will be used to support the application operation, their functionalities will be the same.
- *Organizational Units* – the organizational units supporting a particular application operation should be ideally the same.
- *Mode* – for the same application operation, the mode of processing or the type of interaction with the system will be typically the same.

Hence, requests to a particular application operation is categorised as a "Single Class" in this thesis.

In view of the above and the fact that applications are run on multi-core, multi CPU servers today, for simplicity, we assumed Multi-Processor Single Class Queuing Network (open or closed) model approximation (Menasce et al., 2002). With *m* resources and *D* service demand at each resource, the service demand at the single resource queue will be *D*/*m* and for the delay resource will be *D*(*m*-1)/*m*. Under light load, the Residence time (R_i') is *D* (proven) and under heavier load, it will be dominated by the single resource queue:

$$R_i' = V_i W_i + D_i \qquad (3.1)$$

where V_i is the average no. of visits, W_i is the average waiting time and D_i is the service demand for a request at queue *i* (Menasce et al., 2002). As the requests are to the *same* application operation, applying all the above constraints, D_i and V_i will ideally be same for all requests. As we used the Average Response Time of responses in test runs, the variability of W_i is averaged out. Considering all the above, R_i' is assumed consistent for all requests at queue *i*.

Technology refresh for IT systems, which are typically undertaken every 3-5 years (Archstone Consulting, 2013) are outside of the scope of this thesis. Applications are frequently updated with new software releases. To comply with the changes in business requirements, the application components are modified while the type of the underlying system (i.e. operating system, type of hardware infrastructure and network) remain unchanged. The changes referred to in this thesis are purely at the application layer and no changes to the type of the environment, which includes hardware infrastructure and networks are in scope of this thesis.

The paradigm of Cloud and Elasticity enables *runtime* extension or contraction of the hardware and the application being hosted on those depending on increased or decreased inbound workload. During this process, the type of the hardware does not change. Either additional server instances are provisioned or excess instances are switched off depending on the variation in the workload. However, the approach proposed in this thesis will facilitate *build time* detection and remedy of any possible performance bottleneck due to changes to the application and may not necessarily be due to any increase in the inbound workload condition. Any negative impact may be addressed upfront and potential performance issues can be averted *before* rolling the application to production. The method established will complement the paradigm of Cloud and Elasticity. A detailed discussion of this is presented in Section 9.3.

Various types of optimisation techniques may be applied to boost performance. Just-In-Time (JIT) compilers may be used for faster execution of programs. A JIT compiler runs after a program is started and compiles the code (bytecode or any other type of Virtual Machine instructions) to machine code (Cidade, 2008). Intelligent query optimisation may be adopted with the use of data mining and other techniques (Soliman, A. F., 2007). This thesis does not delve into the details of performance optimisation techniques. It assumes that any optimisation if adopted is applied upfront and focuses on the post optimised state of the application. Prior to optimisation, the initial few requests will result in relatively slower responses and then there will be improved performance post optimisation for the rest of the application’s life. For example, if JIT compiler is used, the *first time* the application operation is invoked, the JIT compiler will convert the bytecode to machine code, which will then be used as long as the application is running. Hence, there will not be any further variation in the application operation’s performance due to JIT compilation. However, as the controlled experiments are run repeatedly for a number of times (optimal number determined through Standard Deviation of the response times as explained in Chapters 5 and 6) with the same application configuration and the average latency of each type of Delay Point is recorded, the impact of JIT compilation (if applied) should be smoothened out. Finite resource pools like thread pools minimise the overhead due to thread creation for every request. An important advantage of the fixed thread pool is that applications using it *degrade gracefully* (Oracle, 2014). When the number of requests being processed reaches the maximum pool size, subsequent requests are queued and only serviced when worker threads are available again. This thesis does not deal with any such resource pools in the application layer explicitly. However, to reduce any possibility of resource contention as described above, the application container is configured in a way so that the maximum inbound workload remains within the limit of the maximum number of requests that can be handled concurrently by the container.

Finally, network latencies (inter-component and Provider to Consumer) impact overall latency as well. Our example TCG application has co-located components with local calls between them. The client Auto-Trader application is hosted on the same network as the TCG application at a data centre. Hence no unexpected fluctuation of network bandwidth is foreseen between the Auto-Trader and the TCG application as explained in Section 1.2. Also, only *formal* application operation consumption contracts are in scope with dedicated, controlled network traffic and *not* any random application access over public network. For example, the Quotes Distributor component retrieves the real time and historic prices from an external system over a dedicated leased network. Hence, at *runtime*, no unpredictable fluctuation of network bandwidth or latency is assumed. Average resource usage effect of other application operations on requests of the tested operation is assumed.

With all the above boundary conditions in place, we logically deduce *constancy* of overall runtime Impedance for processing requests to the same application operation for a particular operation Load Potential (‘V’). As Admittance (‘Y’) is the inverse of Impedance, constant overall runtime Impedance implies constant overall runtime Admittance ‘Y’ for processing requests to the *same* application operation for a particular value of ‘V’ as well.

The work done in (Cherkasova et al., 2007) corroborates the above logical deduction. A Cumulative Distribution Function (CDF) is generated for the service time S_i of a typical server transaction for different utilization points U_k over time. By solving:

$$S_i = R_{i,k} * (1 - U_k / 100) \qquad (3.2)$$

a set of solutions $S_{i,k}$ is obtained with a large number of similar points in the middle and some outliers in the beginning and the tail of the curve. The 50th percentile value as the solution for S_i worked very well for all the transactions of that type across time. An application operation involves a particular type of transaction. Hence, this deduction also implies that an application operation's Admittance, which is the measure of ease with which a request to the operation is processed and response sent back, is the same for all the transactions of the same type under the same workload condition and hosting environment across time.

3.3.2 The *PALP* Model

Given the deduction of constancy of the runtime operation Admittance 'Y' as described in Section 3.3.1, the ***first hypothesis*** of this thesis is that for a given class of workload, a given hosting environment and a given range of Load Potential *'V'*, an application operation's Performance *'P'* can be expressed as a linear function of its Load Potential *'V'* at runtime and the gradient constant is the runtime Admittance *'Y'* of that operation. The hosting environment includes the connecting network between the application and the operation consumer. Let's consider our example TCG application. Given our definition of 'V', with the increase of inbound workload, 'V' will decrease and vice-versa. If the inbound workload to the getConfidence(...) operation decreases (i.e. the operation's 'V' increases), the operation's 'P' will also increase as a result of this. This implies 'P' is directly proportional to 'V'. For a given range of inbound workload to the getConfidence(...) operation (i.e. for a given range of 'V'), the operation's 'Y' is the proportionality constant, which implies that for the given range of 'V', if the getConfidence(...) operation's 'Y' increases, its 'P' will also increase.

3.4 Operation Admittance and Delay Point Latencies

As shown in Figures 5 and 6, an application's operation Admittance serves as a bridge between the *PALP* model and the internal Delay Points catering to the operation. In order to use the knowledge derived out of the *PALP* model to fine tune the Delay Point latencies (i.e. modifying application algorithm and configuration) and vice-versa, it needs to be determined how the operation Admittance is influenced by the variation in the Delay Point latencies. There can be two scenarios by which the Admittance may be affected as described below:

3.4.1 Operation Admittance and changes to Delay Points of a specific type

An operation's Admittance may be affected due to changes to Delay Points of a particular type. It may so happen that due to the changes to the business requirements, the Delay Points of just one particular type are required to be modified with all the other processing activities remaining unchanged. Changes to the Delay Points of that particular type will potentially change the cumulative latency of the Delay Points of that type, which in turn will impact the overall operation Admittance. In our example TCG application, consider a scenario when due to some unforeseen events in the global financial markets, the nature of movements of the GBP/USD currency pair is affected greatly. As a consequence, the currency pair temporarily stopped following the historical trends and instead started reacting very abruptly to the current news and events happening around the world. In order to adapt to this change to the market sentiment, only the algorithms processing the logic in-memory in the TA and FA components were required to be modified. All the other processing activities remained intact. However, these changes to the in-memory data processing activities in the TA and FA components will result in a change to the latencies caused by these activities across the components catering to the operation. This change to the overall latency will in turn impact the Admittance ('Y') of the getConfidence(…) operation. Once the new 'Y' is calculated, the corresponding new 'P' for the operation can be calculated using the PALP model.

This thesis attempts to address the need of a simple, pragmatic approach to evaluate the performance of existing software applications undergoing frequent updates, which will not require repetitive load/performance testing, complex SPE techniques, performance models and resource intensive application profilers. The approach will not require the software engineers to learn new, non-standard modelling formalisms and notations or to specify the software system using process algebras and complex mathematical analysis. To predict the impact on operation Admittance due to changes to the latencies of the Delay Points of a particular type, the "black-box" approach to the underlying system as proposed by (Westermann et al., 2010) is adopted. The ***second hypothesis*** states that under a given workload condition and hosting environment, the operation Admittance ('Y') can be expressed as a statistical *function* of the total cumulative latency of the Delay Points of a particular type across all the supporting application components catering to the operation i.e.

$$\mathrm{Y} = f\,(\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{L_{DLPi}}) \qquad (3.3)$$

where $\mathrm{L_{DLP1}}$ is the latency (or delay) introduced by the Delay Points of a particular type in Component1. This function represents the distinct *pattern* by which the total cumulative latency introduced by the Delay Points of one particular type across all the components 1 to n $(\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{L_{DLPi}})$ influence 'Y' and hence 'P' and potentially 'V' in case a target performance level needs to be attained. In our example TCG application, this implies that the Admittance (‘Y’) of the getConfidence(…) operation can be expressed as a statistical function of the total cumulative latency of any of the following:

a. All the in-memory data processing activities across TA Component1 to TA Component5 and the FA Component1.
b. All the file I/O activities across TA Component1 to TA Component 5 and FA Component1.
c. All the database interactions across TA Component1 to TA Component5 and FA Component1.

Extending from the first working hypothesis, this implies that for a given range of ‘V’, the performance of the operation may be expressed as:

$$P = (f (\sum_{i=1}^{n} L_{DLPi})) V + c \qquad (3.4)$$

Atomicity of Delay Points is very important as the type of Delay Point determines its nature of system resource usage, which in turn influences the pattern of impact of the Delay Point on the operation’s performance. Hence, Delay Points should be granular (or atomic) and should not span across different types of processing activities. For example in-memory data processing may be of different types – pure numeric data processing, pure textual data processing, combination of numeric and textual data processing and other types. Numeric data processing will use the Arithmetic and Logic Unit (ALU) (Arithmetic Logic Unit, 2013) but pure textual data processing will not need the ALU. In this example, due to the difference in the types of system resources used, we need to have two categories of atomic Delay Points – Numeric Data Processing Delay Point and Textual Data Processing Delay Point. Atomicity of the Delay Points and workload partitioning is discussed in detail in Section 9.4.3.

3.4.2 Operation Admittance and changes to Delay Points of multiple types

An operation’s Admittance may be affected due to simultaneous changes to Delay Points of multiple types. It may so happen that changes to business requirements warrant the in-memory data processing activity, some File I/O activities and a database interaction to be modified simultaneously. Changes to these different types of Delay Points will potentially change the latencies of the Delay Points, which in turn will impact the overall operation Admittance. The second working hypothesis is extended to the next level for changes to Delay Points of multiple types simultaneously.

The ***third hypothesis*** of this thesis states that under a given workload condition and hosting environment, in the event of simultaneous changes to multiple types of Delay Points, the total cumulative latency of Delay Points of each type has an implicit relative weight associated to it, which determines the degree of its impact on the operation Admittance. Determining the *best-fit* relative weights will facilitate predicting the operation Admittance for any future set of latency values of the Delay Points. To illustrate this, we consider an application operation which is supported by 'n' number of components. If there are 4 different types of Delay Points (DLP_type1, DLP_type2, DLP_type3 and DLP_type4) designed across the components, the total cumulative latency introduced by the Delay Points of type DLP_type1 can be denoted by $\sum_{i=1}^{n} \text{DLP\_type1}_i$, the total cumulative latency introduced by the Delay Points of type DLP_type2 can be denoted by $\sum_{i=1}^{n} \text{DLP\_type2}_i$ and so on. Hence, the third working hypothesis states that:

$$Y = f\,(w1 * \sum_{i=1}^{n} \text{DLP\_type1}_i + w2 * \sum_{i=1}^{n} \text{DLP\_type2}_i + w3 * \sum_{i=1}^{n} \text{DLP\_type3}_i + w4 * \sum_{i=1}^{n} \text{DLP\_type4}_i\,) \qquad (3.5)$$

where *w*1, *w*2, *w*3 and *w*4 are the relative weights by which the latencies of the Delay Points of type DLP_type1, DLP_type2, DLP_type3 and DLP_type4 impact the overall operation Admittance ('Y'). The formula in (3.5) can also be expressed in the form:

$$Y = f\,(\sum_{j=1}^{4} (w_j * \sum_{i=1}^{n} \text{DLP\_type}_{j,\ i})\,) \qquad (3.6)$$

This thesis considers the total cumulative latency introduced by each type of Delay Points (i.e. $\sum_{i=1}^{n} \text{DLP\_type1}_i$, $\sum_{i=1}^{n} \text{DLP\_type2}_i$ etc) as a predictor variable and w1, w2,...w4 as the relative weights by which each of these latencies influences proportionately the operation Admittance multiple regression. The thesis proposes a technique to determine the best-fit values of the relative weights for the predictor variables (i.e. $\sum_{i=1}^{n} \text{DLP\_type1}_i$, $\sum_{i=1}^{n} \text{DLP\_type2}_i$ and so on) in a multiple regression. These relative weights will then facilitate prediction of the resultant Admittance of an operation caused by any future set of latency values of the Delay Points. Let's consider our example TCG application to illustrate this third hypothesis. Unlike the scenario cited in Section 3.4.1, suppose for a particular change in business requirements, the in-memory data processing activities, the file I/O activities and the database interactions – all of these Delay Points are needed to be modified across the TA Components (1 to 5) and the FA Component. Given this scenario, the hypothesis implies the following:

a. The total cumulative data processing latency across all the six components is considered as a predictor variable. The degree of impact of this predictor variable on the Admittance 'Y' of the getConfidence(...) operation is determined by a relative weight which is implicitly associated to this predictor variable.
b. The total cumulative file I/O latency across all the six components is considered as the second predictor variable. The degree of impact of this predictor variable on the Admittance 'Y' of the getConfidence(...) operation is determined by a second relative weight which is implicitly associated to this second predictor variable.
c. The total cumulative database interactions latency across all the six components is considered as the third predictor variable. The degree of impact of this third predictor variable on the Admittance 'Y' of the getConfidence(...) operation is determined by a third relative weight which is implicitly associated to this third predictor variable.

Although there are no unambiguous measures of relative weight when the predictor variables are correlated, some measures have been shown to provide meaningful results (Budescu, 1993; Lindeman et al., 1980). However, these measures are very difficult to implement when the number of predictors is greater than about five (Johnson, 2000). Applying a heuristic method for estimating the relative weights of predictor variables in multiple regression, a very effective technique to predict the best-fit weights is proposed in (Johnson, 2000). This method is computationally efficient with any number of predictors and is shown to produce results that are very similar to those produced by more complex methods. The method proposes that the set of predictor variables, the set of corresponding relative weights and the set of transformed regressed values may be expressed in the form:

$$AX = B \qquad (3.7)$$

where:

- A is the [m x n] matrix containing rows of predictor variables from different test runs
- B is the single column [m x 1] matrix of the transformed value for each row in A
- X is the single column [n x 1] matrix of the relative weights

The *best fit* value of *X* can be calculated by the formula:

$$X = (A^T A)^{-1} A^T B \qquad (3.8)$$

This thesis adopts the technique proposed by Johnson to evaluate the relative weights associated with the latencies of each type of Delay Points, which determine the proportionate contribution of those respective latencies to the overall operation Admittance. The set of relative weights is then used to predict the operation Admittance ('Y') for any future set of latency values for the different Delay Points. The Performance ('P') corresponding to the predicted 'Y' under the given Load Potential ('V') can then be computed either through extrapolation or interpolation of the 'P' versus 'Y' function generated previously using the PALP model. The way in which the *PALP* model and the two methods proposed by this thesis complement each other and work in conjunction is illustrated in Section 3.6 with the aid of Figure 7.

3.5 The Empirical Approach to Evaluation - Rationale

Depending on the purpose of the evaluation, whether it is techniques, methods or tools, and depending on the conditions of the empirical investigation, there are three major types of investigation strategies that may be carried out: *Survey*, *Case Study* and *Experiment* (Wohlin et al., 2012; Robson, 2002).

Survey – In scientific evaluation, a survey is often an investigation, which is performed in retrospect, when for example, a tool or technique, has been in use for a while (Pfleeger, 1994). The primary means of gathering qualitative or quantitative data are interviews or questionnaires.

Case Study - In software engineering, case study is an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real life context, especially when the boundary between phenomenon and context cannot be clearly specified (Runeson et al., 2012).

Experiment (or controlled experiment) – In software engineering, it is an empirical enquiry that manipulates one factor or variable of the studied setting (Wohlin et al., 2012). Based in randomization, different treatments are applied to or by different subjects, while keeping other variables constant, and measuring the effects on outcome variables. In technology oriented experiments, different technical treatments are applied to different objects. Experiments are mostly done in a laboratory environment, which provides a high level of control. When experimenting, subjects are assigned to different treatments at random. The objective is to manipulate one or more variables and control all the other variables at fixed levels. The effect of the manipulation is measured and based on this a statistical analysis can be performed.

Ideally, the three hypotheses stated in Sections 3.3.2, 3.4.1 and 3.4.2 above should be evaluated on more than one subject to gain statistical significance. However, for the purpose of the controlled experiments to be performed for this thesis, an application framework is needed, which will resemble our example TCG application of SKGWorld Forex Ltd. The framework will expose an operation (similar to the getConfidence(…) operation in our example) processing a particular transaction involving various types of Delay Points. This operation will be consumed by an application client (similar to the Auto-Trader application in our example), which will invoke calls on the application operation. To resemble real life scenarios, the framework is required to comprise several components which will be used to cater to the higher level operation. The different types of Delay Points need to be spread across the components in the way the various Delay Points are spread across the TA and FA components of the TCG application.

A *black-box* approach is taken with a Web Server in (Menasce et al., 2002) to model a finite queue system for determining the variation in response time against increase of arrival rate of requests per second. To evaluate the *PALP* model as stated in the first working hypothesis, a *black-box* approach similar to Menasce et al needs to be taken towards the application operation under observation. To determine the nature of proportionality of the operation's Performance (timeliness of response) against the operation's Load Potential and map the operation's Admittance to the proportionality constant for a given class of workload and a hosting environment, a controlled environment is required. Using the environment, while maintaining the same application algorithm and configuration (i.e. operation Admittance), the Load Potential can be varied by varying the inbound workload at random. The impact of this variation on the operation's Performance can then be measured.

To evaluate the second working hypothesis, the approach proposed in (Westermann et al., 2010) is adopted. While this thesis's approach does not consider the internal structure of the underlying system and takes a *black-box* view to it, it focuses on the observable data and adopts a *white-box* approach for the application operation. Firstly, the application components supporting the operation need to be decomposed into Delay Points. To determine the pattern in which the variation to the total cumulative latency of a particular type of Delay Point influences the overall Admittance (and hence the Performance) of the operation, the processing intensities of that type of Delay Point(s) need to be varied while keeping the algorithms and processing activities of all the other types of Delay Points constant and maintaining a steady Load Potential. Across the application components, the *PALP* model based indicative values of 'Y' need to be computed against the variability of the actual measured Delay Point latencies. Using appropriate statistical regression functions like the Least Square Fitting (LSF) (Weisstein, 2013) the "best-fit" functions from the collected data set need to be derived. The function graphs will provide the graphical patterns to enable *bi-directional* (i.e. bottom-up and top-down) analysis of the operation's Performance, Admittance (related to the application algorithm and configuration) and Load Potential (related to the inbound workload).

To evaluate the third working hypothesis, a similar *white-box* approach as above needs to be taken towards the application operation. Maintaining a constant Load Potential, the processing intensities of all the different types of Delay Points across the various backend components need to be varied simultaneously and the *PALP* model based indicative values of 'Y' be computed against this variability. From the gathered data set, in line with the method as proposed in (Johnson, 2000), the relative weights by which the different types of Delay Points contribute to the overall operation Admittance will then be computed. The set of relative weights will then facilitate prediction of the operation Admittance for any future set of latencies of the different Delay Points.

All the above evaluation procedures suggest that the approach will entail manipulation of one or more variables and control of all the other variables at fixed levels for particular configurations. The effect of the manipulation will be measured and statistical analysis will be performed. Model based indicative values of some features will be computed against variability of the Delay Point activities of the underlying application components supporting the application operation. Lookup datasets against different system configurations will be created to associate these computed values to the actual measured values of other features. Established mathematical techniques will then be applied with appropriate statistical regression types to enable trend extrapolation and interpolation. This research method aligns itself with the "Controlled Experiment" empirical strategy as described in (Wohlin et al., 2012).

The research attempts to allow the software developers to undertake performance analysis for modifications to the application layer. The target users of the methodology are the software developers who have knowledge of the application codebase and its behaviour. Applying the methods to create the initial reference data set from the existing application will warrant modifying the Delay Points supporting the application operation and probing their respective latencies through instrumentation. The current research does not provide tool support for automated instrumentation and recourses to manual insertion of probes. Modifying the Delay Points may impact the behaviour of the operation in some way. Hence, knowledge of the application operation's codebase, its behaviour and implication of behavioural changes is required for the current version of this thesis.

Due to all of the above mentioned constraints, an application framework resembling our example TCG application is designed and developed to conduct an *experimental* research for the evaluation purposes in this thesis. The framework exposes an application operation (similar to the getConfidence(…) operation) through a web service, which is supported by several backend components and is consumed by a multi-threaded client (to generate workload). The client resembles the Auto-Trader application in our example.

3.6 Use of the Model and Methods in conjunction

The approach proposed in this thesis combines the *PALP* model and the two methods to analyse and forecast software performance for changes to a single type or multiple types of Delay Points respectively. It is formulated for ***existing*** software applications undergoing frequent software releases and updates. Software engineers will be able to analyse the performance, application algorithm/configuration and workload aspects without the need to learn new, non-standard modelling formalisms and notations or to specify the software system using process algebras and complex mathematical analysis. The approach requires matrix manipulation. However, these matrix calculations can be easily achieved either by using readily available online resources (as used in this thesis) or programmatically.

Figure 7 below shows the steps of the high level end to end process of how the PALP model and the methods will complement each other and will be applied in tandem during the SDLC:

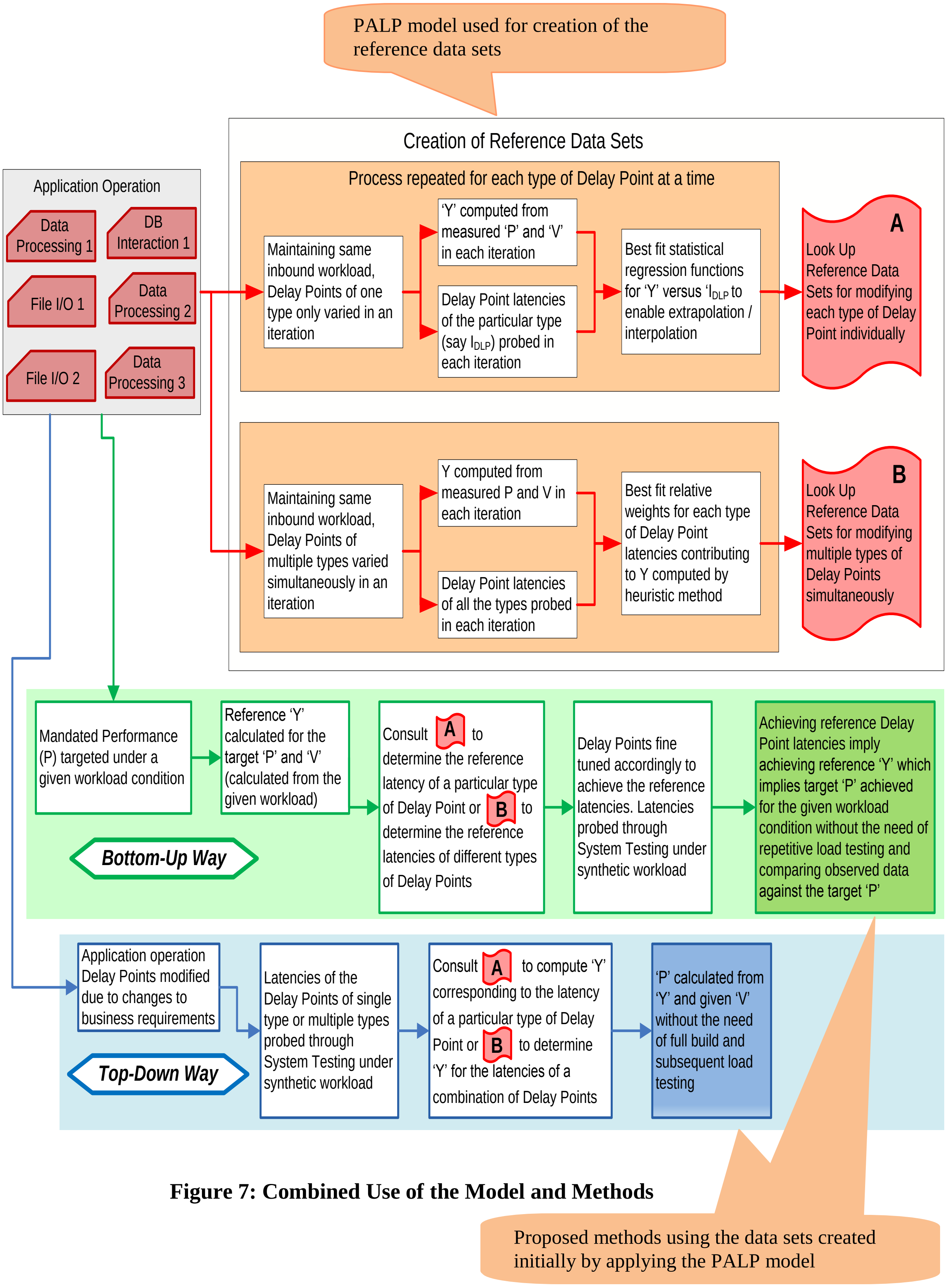

PALP model used for creation of the reference data sets
Creation of Reference Data Sets
Process repeated for each type of Delay Point at a time
Application Operation
Data Processing 1
DB Interaction 1
File I/O 1
Data Processing 2
File I/O 2
Data Processing 3
Maintaining same inbound workload, Delay Points of one type only varied in an iteration
'Y' computed from measured 'P' and 'V' in each iteration
Delay Point latencies of the particular type (say I_{DLP}) probed in each iteration
Best fit statistical regression functions for 'Y' versus 'I_{DLP} to enable extrapolation / interpolation
A
Look Up Reference Data Sets for modifying each type of Delay Point individually
Maintaining same inbound workload, Delay Points of multiple types varied simultaneously in an iteration
Y computed from measured P and V in each iteration
Delay Point latencies of all the types probed in each iteration
Best fit relative weights for each type of Delay Point latencies contributing to Y computed by heuristic method
B
Look Up Reference Data Sets for modifying multiple types of Delay Points simultaneously
Mandated Performance (P) targeted under a given workload condition
Reference 'Y' calculated for the target 'P' and 'V' (calculated from the given workload)
Consult A to determine the reference latency of a particular type of Delay Point or B to determine the reference latencies of different types of Delay Points
Delay Points fine tuned accordingly to achieve the reference latencies. Latencies probed through System Testing under synthetic workload
Achieving reference Delay Point latencies imply achieving reference 'Y' which implies target 'P' achieved for the given workload condition without the need of repetitive load testing and comparing observed data against the target 'P'
Bottom-Up Way
Application operation Delay Points modified due to changes to business requirements
Latencies of the Delay Points of single type or multiple types probed through System Testing under synthetic workload
Consult A to compute 'Y' corresponding to the latency of a particular type of Delay Point or B to determine 'Y' for the latencies of a combination of Delay Points
'P' calculated from 'Y' and given 'V' without the need of full build and subsequent load testing
Top-Down Way
Proposed methods using the data sets created initially by applying the PALP model

Figure 7: Combined Use of the Model and Methods

A brief description of the main steps shown in Figure 7 is provided below. There are broadly 3 steams of activities that may be undertaken:

1) Initial creation of the Reference Data Sets for the application operation.
2) Fine-tuning the application operation’s algorithm and configuration to retrofit a target performance level under a specified load condition in a bottom-up way
3) Determining the performance level for modification to the application operation’s algorithm and configuration under a specified load condition in a top-down way

Creation of Reference Data Sets for the application operation

This is the first phase in which the reference data sets for an existing application operation hosted on a given environment are created. The lookup data sets are created upfront by applying the *PALP* model and the proposed methods. We present below the step-by-step description of the process to create the reference data sets:

1) The Stress Point (SP) of the application operation is determined by incrementally varying the generated inbound workload to the operation and recording the workload beyond which the response time becomes unacceptable to the business stakeholders.
2) Under any given workload (WL), the operation’s Load Potential ‘V’ is calculated using Equation (1.1) i.e. $V = SP – WL$.
3) Maintaining positive ‘V’, by varying the inbound workload, it is observed how the operation’s performance ‘P’ varies against changes to ‘V’. In this way, the variation of ‘P’ as a non-linear function of ‘V’ is recorded.
4) Applying *PALP* model, the non-linear function of ‘P’ versus ‘V’ is broken into piecewise linear functions (Fourer et al., 2003). For every linear piece, the best fit linear function is derived using established mathematical techniques. For the purpose of our experiments, Microsoft Excel Toolset has been used. In this way, the ‘V-Bands’ are demarked with their respective linear functions of $P = YV + \mathrm{c}$. Details of the experiments and derivation of this model is presented in Chapter 5.
5) For each ‘V-Band’, their respective ‘c’ values (y-intercept) and their range of ‘V’ values are recorded. This is to help identifying the appropriate ‘c’ value for a value of ‘V’ calculated from any future workload.

6) Create the reference data for changes to Delay Points of individual types from the following steps:

   a. Taking each type of Delay Points at a time, the latencies of the Delay Points of a specific type (say I_{DLP} cumulatively) are varied under a given generated workload condition. All the other types of Delay Points are maintained unchanged.

   b. For every variation of $\mathrm{I}_{\mathrm{DLP}}$, the following data is recorded:

      i. Actual operation performance ‘P’. This is the inverse of the measured response time of the operation under the given workload.

      ii. ‘V’ is calculated as SP – WL.

      iii. The operation’s admittance ‘Y’ is computed by applying the PALP model’s linear function $P = YV + \mathrm{c}$ for the relevant ‘V-Band’. Details of the process of establishing this model are presented in Chapter 6. ‘Y’ is calculated by applying Equation (6.1): $Y = (P – \mathrm{c}) / V$. The value of ‘c’ is determined by the relevant ‘V-Band’.

      iv. The actual cumulative $\mathrm{I}_{\mathrm{DLP}}$ across all the Delay Points of the specific type under observation is recorded.

   c. From the computed values of ‘Y’ and the actual measured values of $\mathrm{I}_{\mathrm{DLP}}$, the function of ‘Y’ versus ‘$\mathrm{I}_{\mathrm{DLP}}$’ is derived from the best fit LSF function. For the purpose of our experiments, Microsoft Excel Toolset has been used. For example, in our experiments as detailed in Chapter 6, ‘Y’ of the application operation varied as a polynomial function of 4th order of the Database Interactions latencies and is represented by the Equation (6.2) : $Y = 0.1173I_{DB}^{4} - 0.4938I_{DB}^{\ 3} + 0.7728I_{DB}^{\ 2} - 0.5367I_{DB} + 0.1814$ where I_{DB} is the cumulative Database Interactions latency across all the Database Interactions Delay Points.

   d. The same process is repeated for every type of Delay Points and their respective ‘Y’ versus ‘$\mathrm{I}_{\mathrm{DLP}}$’ functions are derived in the same way as above.

   e. These derived functions serve as the reference data for future predictions. For the given range of ‘V’ (i.e. given range of workloads), these functions can be used to interpolate or extrapolate possible future values of ‘Y’ (say Y_p) corresponding to any future value of Delay Point latency. This Y_p can then be fed into the *PALP* model for that particular ‘V-Band’ to have an estimate of the possible future performance ‘P’ (say P_p) i.e. $P_p = Y_p\mathrm{V} + \mathrm{c}$, where ‘c’ is ‘V-Band’ specific as explained earlier. Details of the process and the associated steps are presented in Chapter 6.

7) Create the reference data for changes to Delay Points of multiple types from the following steps:

   a. For the purpose of illustration, let us assume that there are 3 different types of Delay Points – DLP1, DLP2 and DLP3. Unlike the previous exercise of varying the latencies of the Delay Points of a specific type at a time, the latencies of the Delay Points of all the 3 types are varied simultaneously under a given generated workload condition.

   b. For every combination of variations of I_{DLP1}, I_{DLP2} and I_{DLP3}, the following data is recorded:

      i. Actual operation performance ‘P’ measured in the same way as explained before.

      ii. ‘V’ is calculated as explained before.

      iii. The operation’s admittance ‘Y’ is computed in the same way as explained before.

      iv. The actual cumulative latencies I_{DLP1}, I_{DLP2} and I_{DLP3} for all the Delay Points of type DLP1, DLP2 and DLP3 respectively.

   c. The computed values of ‘Y’ and the corresponding sets of the actual measured values of I_{DLP1}, I_{DLP2} and I_{DLP3} are represented in a matrix form $AX = B$ as presented in Equation (3.7). In the context of this illustration, the components of the matrix equation are as follows:

      - *A* is the [r x 3] matrix containing ‘r’ rows of I_{DLP1}, I_{DLP2} and I_{DLP3} from ‘r’ number of test runs. Each row is treated as a set of predictor variables.
      - *B* is the single column [r x 1] matrix of the calculated ‘Y’ for each row in *A*. Each value of ‘Y’ is treated as the dependant variable for the corresponding row in *A*.
      - *X* is the single column [3 x 1] matrix of the *best-fit* relative weights associated with latencies I_{DLP1}, I_{DLP2} and I_{DLP3}.

   d. The *best fit* value of *X* is calculated using Equation (3.8) : $X = (A^TA)^{-1}A^TB$

   e. The Delay Points relative weight matrix *X* acts as the reference data to project ‘Y’ (say Y_{DLPrw}) for any future arbitrary combination of latencies I_{DLP1}, I_{DLP2} and I_{DLP3}. This ‘Y_{DLPrw}’ can then be fed into the *PALP* model for that particular ‘V band’ to estimate the possible performance ‘*P*’ (say P_{DLPrw}). The predicted performance ‘P_{DLPrw}’ can be calculated using Equation (5.2) : $P_{DLPrw} = (Y_{DLPrw} * V) + c$

Fine-tuning the application algorithm and configuration to retrofit a target performance level under a specified load condition in a bottom-up way

This refers to the “***Bottom-Up Way***” part of the Figure 7 above. If a target performance level is to be achieved under a specific workload condition on the given hosting environment, the proposed methodology will enable upfront fine-tuning of the application algorithm and configuration during development to achieve the purpose without the need of repetitive performance testing to attain the benchmark.

If the Delay Points of one particular type is needed to be fine-tuned, a target ‘Y’ (say Y_t) for the operation is calculated for the target ‘P’ (say P_t) and ‘V’ (computed from the specified workload and reference *Stress Point*), using the *PALP* model. ‘Y_t’ is calculated using Equation (6.1): $Y_t = (P_t – \mathrm{c}) / V$. The value of ‘c’ is determined by the relevant ‘V-Band’. Using the reference function of ‘Y’ versus ‘I_{DLP}’ derived for the Delay Points of that particular type in the previous phase “Creation of Reference Data Sets for the application operation”, the Delay Point latency I_{DLP} corresponding to the target ‘Y_t’ is estimated. This I_{DLP} serves as the target Delay Point latency. With this knowledge of a target Delay Point latency, the software developers will then be able to fine tune the Delay Points of that particular type so that the actual measured cumulative latency of the modified Delay Points of that type is less than or at most equal to the previously computed target Delay Point latency. In this way, the application algorithm may be fine-tuned upfront to attain the prescribed performance level.

In case of fine tuning multiple types of Delay Points, the *best-fit* relative weights data (matrix *X* as derived in the previous phase) is consulted to adjust the latencies of every type of Delay Point so that the resultant ‘Y’ is equal to the previously calculated target ‘Y_t’. These adjusted latencies of the different types of Delay Points then serve as the target for the actual measured latencies of the modified Delay Points. Equipped with the knowledge of these target Delay Point latencies to achieve the target ‘Y_t’ (hence the target ‘P’), the software engineers can then probe the latencies of the modified Delay Points of different types and fine tune those as needed. The actual measured latencies of the modified Delay Points should be less than or at most equal to the previously computed target Delay Point latencies. This may be achieved through systems testing by only covering all the relevant

Delay Points catering to the operation instead of execution of the full build of the application and subsequent load testing. A multi-threaded client may be used to generate synthetic workload for the operation. This method will facilitate fine-tuning the application operation's algorithm and configuration upfront to attain the prescribed performance level instead of recourse to a repetitive and *reactive* approach of setting thresholds for observed performance metrics and raising alarms when these thresholds are violated as adopted during CPM.

Determining the performance level for modification to application algorithm and configuration under a specified load condition in a top-down way

This refers to the "***Top-Down Way***" part of the Figure 7 above. During modification of an application operation, depending on whether Delay Points of a single type are being modified at a time or of multiple types being modified simultaneously, the software developers can probe the Delay Point latencies as appropriate (for a particular type or multiple types) through systems testing using a multi-threaded client to generate synthetic workload for the operation. The developers can generate the inbound workload themselves if required.

If Delay Points of a specific type are being modified, the reference function of 'Y' versus 'I_{DLP}' derived for the Delay Points of that particular type in the previous phase "Creation of Reference Data Sets for the application operation" is consulted and a predicted value of 'Y' (say Y_p) corresponding to the measured Delay Point latency is estimated from the function. The operation's predicted performance P_p can then be calculated using Equation (5.2): $P_p = Y_pV$ + c.

If Delay Points of multiple types are being modified simultaneously, the Delay Points relative weight matrix X as derived in the previous phase "Creation of Reference Data Sets for the application operation" acts as the reference data to project 'Y' (say Y_{DLPrw}). The measured latencies of the various Delay Points (say I_{DLP1}, I_{DLP2} and I_{DLP3}) are multiplied (matrix multiplication) by their corresponding relative weights in matrix X to predict Y_{DLPrw}. This 'Y_{DLPrw}' is then used in the *PALP* model for that particular 'V band' to estimate the possible performance '*P*' (say P_{DLPrw}). The predicted performance 'P_{DLPrw}' can be calculated using Equation (5.2): $P_{DLPrw} = (Y_{DLPrw} * V) + c$

3.7 Chapter Summary

This thesis adopts the approach as advocated by (Westermann et al., 2010) and (Cherkasova et al., 2007). Developers and testers depend on experience to use the results from instrumentation tools for diagnosing performance problems. The current performance modelling processes are costly, require heavy effort and lack measurement standards. Many developers do not trust or understand performance models, even if such models are available. Better, simple and easy to use methods are proposed by (Woodside et al., 2007) as a future requirement for interpreting the results and diagnosing performance problems. This thesis aligns with this future direction.

The chapter describes how this thesis augments the work done in (Cherkasova et al., 2007), which is a very linear, sequential approach used to compute the overall latency of a transaction, which relates to the transaction's performance. This thesis proposes a combination of model based and measurement based approach focussing primarily on the application layer of the system. Other than the application layer, the approach uses "black-box" performance models, which do not contain any information about the underlying system's internal structure and the performance of the system is captured by a function of its usage. The reasons for this approach are manifold. Typically, the application components are modified due to changes in business requirements while the underlying system comprising the operating system, hardware infrastructure and network remain unchanged. It will also allow the software developers to assess the impact of the changes to an application operation upfront without the need of additional quality assurance experts. The approach will circumvent the unnecessary notational hurdle acting as an impediment to the understanding and uptake of modern performance analysis technologies. No system level utilization diagnostics and measurement will be required for which the software developers may not have adequate expertise (Marz, 2005).

The concepts of operation Admittance, Load Potential and Delay Points are revisited. This chapter discusses the rationale behind the performance analysis methodology proposed in the thesis and states the three hypotheses which form the core of this thesis.

The *first hypothesis* is stated in the context of the *PALP* model between an operation's ***P***erformance, ***A***dmittance and ***L***oad ***P***otential. It states that, for a given class of workload, a given hosting environment and a given range of Load Potential *'V'*, an application operation's Performance *'P'* can be expressed as a linear function of its Load Potential *'V'* at runtime and the gradient constant is the runtime Admittance *'Y'* of that operation. All the other boundary assumptions for the applicability of the research are discussed in this context and the constancy of runtime operation Admittance *'Y'* is explained and justified. Ways in which various caching techniques, thread pools, Intelligent Query Optimization including JIT Compilers, Cloud and Elasticity etc. may influence performance have also been discussed.

The impact on operation Admittance *'Y'* for changes to Delay Points of a *specific* type is discussed. The *second hypothesis* is stated in this context. It states that under a given workload condition and hosting environment, *'Y'* can be expressed as a *function* of the total cumulative latency of the Delay Points of a specific type across all the supporting application components catering to the operation.

The chapter then discusses the impact on operation Admittance for simultaneous changes to Delay Points of multiple types. In this context, the *third hypothesis* of this thesis states that under a given workload condition and hosting environment, for simultaneous changes to multiple types of Delay Points, the total cumulative latency of Delay Points of each type has an implicit relative weight associated to it, which determines the degree of its impact on the operation Admittance. Determining the *best-fit* relative weights will facilitate predicting the operation Admittance and hence the operation Performance for any future set of latency values of the Delay Points. The chapter explains the rationale for this thesis to adopt the technique proposed by (Johnson, 2000) to evaluate the *best-fit* relative weights associated with the latencies of each type of Delay Points, which determine the proportionate contribution of those respective latencies to the overall operation Admittance. The set of relative weights is then used to predict the operation Admittance for any future set of latency values for the different Delay Points.

We conclude by explaining how the *PALP* model and the associated methods complement each other and work in conjunction during the SDLC. The lookup reference data sets are created once upfront for all the Delay Points supporting an application operation. During development, the application Delay Points can be fine-tuned by the developers through system testing to *retrofit* a target performance level under a specified load condition instead of recourse to a repetitive and *reactive* approach of setting thresholds for observed performance metrics and raising alarms when these thresholds are violated as adopted during CPM. Impact on the performance for modification to application algorithms under a specified load condition may be determined by only covering the modified Delay Points without the need of a full build of the application and subsequent load testing, which has overhead from time, resource and cost perspective.

4 The Experiment Environment

This chapter describes the implementation details of the prototype application framework designed and built for the purpose of conducting the experiments. It describes the high level overall framework before delving down into the specifics of the application operation consumer and the operation provider sub-frameworks. It then explains at a high level with the aid of sequence diagrams how a typical operation request from a consumer is processed.

4.1 Overall Application Framework

To assess the impact on the performance of an operation due to changes to the underlying application component processing activities (Delay Points) catering to the application operation and the variation in the operation's inbound workload, an application framework is required. The framework is required to resemble a real life scenario. For the purpose of this thesis, the framework will resemble our example TCG application. In the framework, a consumer will request a particular type of transaction by calling an operation on the application provider's web service interface. This is similar to our example's Auto-Trader application calling the getConfidence(…) operation of the TCG application. The framework is required to be configurable to enable spawning of multiple simultaneously transaction requests by the operation consumer. There has to be provision for varying the number of requests thus varying the workload and also provision to configure the application Delay Points catering to the operation. To achieve this, a prototypical application framework comprising a web service front end facade with other lower level backend application components is created. This served as the provider of the application operation, very similar to the TCG application providing the getConfidence(…) operation. A multi-threaded, configurable client is developed to serve as the operation consumer and generate workload. The consumer will invoke a particular transaction by calling the operation on the application's interface.

As shown in Figure 8 below, the application operation provider – consumer framework comprising a multi-threaded operation consumer, a consumer facing web service acting as the operation provider, other backend application components and some other utility components is built. For the purpose of the experiments, some illustrative application component level Delay Points with activities similar to those in the TA and FA components of our example TCG application like Database Interactions, In-

Memory Data Processing, File I/O, Request Authentication and Request Authorization involving XML parsing etc. are also created.

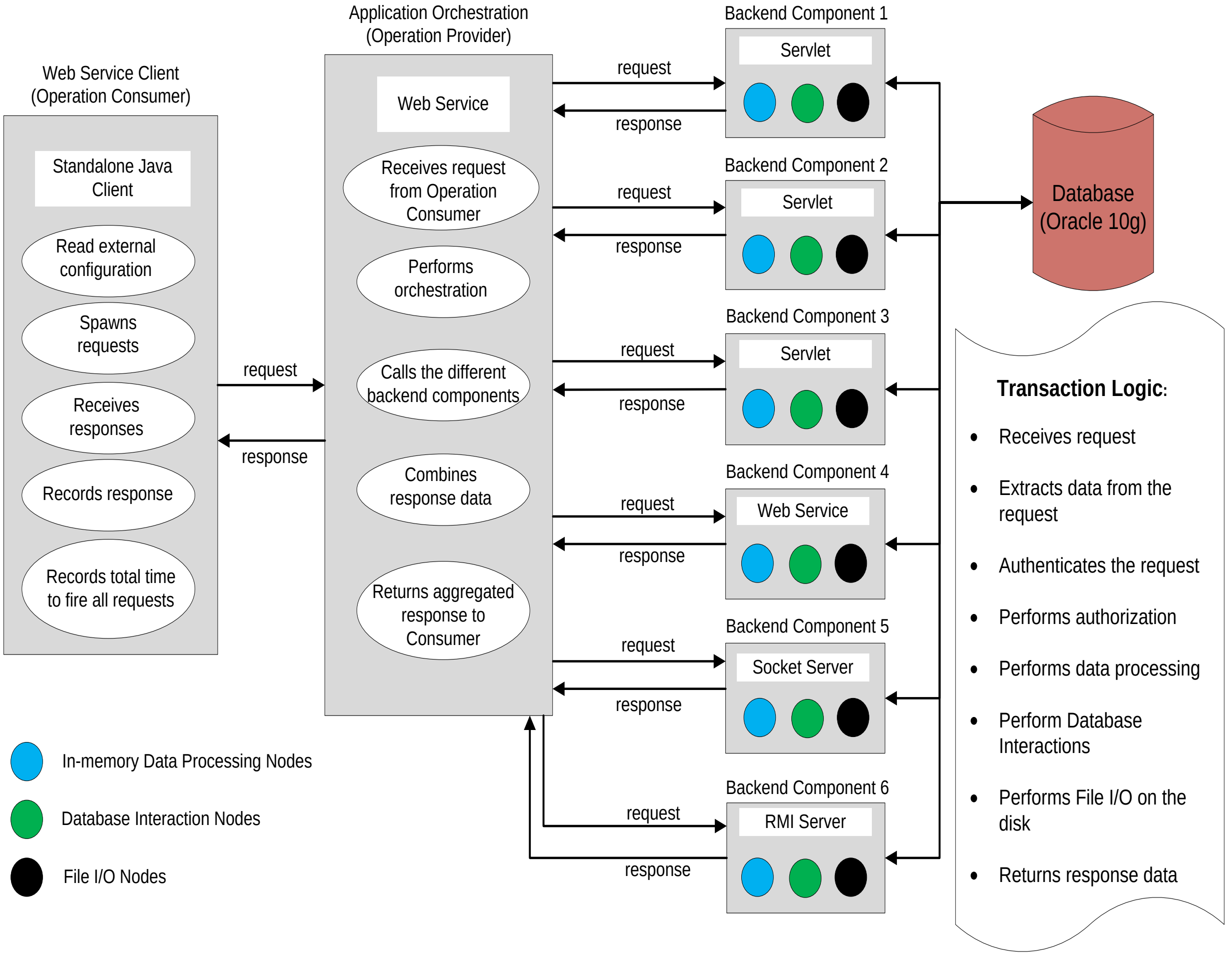


Figure 8: Logical view of the Application Framework

To increase the precision of the model and standardize request resource requirements, partitioning of the inbound workload is achieved by constraining the model and the methods to the application operation level. Requests from the consumer to the different application operations from the same application provider may have different resource requirements based on the nature of the activities required to process the transaction requests.

4.2 The Application Operation Consumer Framework

The application framework is made up of an operation consumer sub-framework and a provider sub-framework. The primary purpose of the operation consumer framework is to:

1. generate the transaction requests with appropriate request data
2. vary the rate of inbound workload based on the external configuration file
3. invoke the application operation on the provider's orchestration web service
4. receive the responses from the application operation and
5. measure the response times and record those

The key constituents of the operation consumer framework are:

a. ***WebServiceClient***

This is a public class which implements a Runnable interface imported from java.lang package. This is done to enable the WebServiceClient to generate multiple operation requests to the Service Provider. Implementing a Runnable interface is preferred over extending the Thread class (java.lang.Thread) to allow future extension of the WebServiceClient class from any business functional class other than the Thread class. The WebServiceClient object serves as the client to the main Orchestration Web Service at the provider's end. The multi-threaded Web Service client (operation consumer) is driven by an external configuration file. It spawns operation requests as per the configuration file and calls the main Orchestration Web Service of the application operation provider framework. This client is similar to the Auto-Trader application in our example, which consumes the getConfidence(…) operation of the TCG application.

Note: The source code of WebServiceClient.java class has been appended to the Annexes.

b. ***Latency.properties*** **configuration file**

This is the external configuration file which is used to configure the Web Service client. The file consists of the following configurable parameters:

1. *numOfRequests* – this parameter determines the inbound workload to the application operation. It specifies the number of operation requests the web service client will invoke on the provider's web service. Each request is a thread spawned.

2. *totalDurationOfRequests* – this parameter enables the web service client to invoke requests within a given period of time.

3. *requestIntervalInMS* – this parameter determines the time interval between two consecutive requests being invoked. Through this parameter, we can vary the number of operation requests invoked within a given period of time.

4. *dataProcessLoop* – this parameter determines the intensity of the in-memory data processing. Increasing the value of this parameter increases the intensity of data processing.

5. *fileIOLoop* – this parameter determines the intensity of the File Input / Output on disk. Increasing the value of this parameter increases the intensity of the File Input / Output.

6. *dbAccessLoop* - this parameter determines the intensity of the database interactions. Increasing the value of this parameter increases the intensity of the database interactions.

7. *Authorization* – this is a toggle switch through which operation request authorization is switched on and off. Authorization is performed by reading rights and privileges from an externally configured XML file.

8. *Authentication* - this is a toggle switch through which operation request authentication is switched on and off. Authentication is performed by reading rights and privileges from an externally configured XML file.

9. *toPrint* – through this parameter setting (Y/N) the printing of the response times gathered from probing the Delay Points is switched on and off.

10. *userId* – this parameter contains the user id of the user invoking the operation request. This id is validated during the authentication process. Based on this id, the request is granted relevant access rights and privileges during the authorization process. For the purpose of the experiments this is sent in clear text form. However, in real life, it will be encrypted.

11. *Password* - this parameter contains the password of the user invoking the operation request. This is checked during the authentication process. For the purpose of the experiments this is sent in clear text form. However, in real life, it will be encrypted.

4.3 The Application Operation Provider Framework

The application operation provider framework exposes the application operation for consumption. This resembles the TCG application in our example. It comprises a *facade* Orchestration Web Service, which orchestrates between the different lower level backend application components. The backend components perform different types of data operations. The Orchestration Web Service receives the transaction request, extracts the request data, calls the relevant backend application components, amalgamates the different responses from the backend components to create the response and sends the response back to the operation consumer, which is the Web Service client. The key constituents of the application operation provider framework are:

a. *MathOrchestrationWebService*

This is the front orchestration web service. It receives the request for operation from the Web Service client (operation consumer), extracts the request parameters and calls the other backend components in their respective protocols. Upon receipt of the responses from the backend components, it manipulates the results and returns a final result to the WebServiceClient. This is basically an application facade for the backend components.

b. *Backend Components 1, 2 and 3*

These application components are similar to the TA and FA components in our example TCG application. They extend the HttpServlet contained in the imported javax.servlet.http package.

These components parse the request data and perform processing activities as defined in the LatencyActivites object. These nodes of activities are the Delay Points which introduce some varied latencies to the overall transaction. LatencyActivities is the object responsible for performing all the internal application level processing activities to introduce latencies to the response formulation. All the components have dependencies on LatencyActivities, which implements the Latencies interface.

c. ***Backend Component 4***

This backend component is similar to the FA component in our example and is wrapped with a web service implementation. It parses the request data and performs some of the operations as defined in the LatencyActivites object.

d. ***Backend Component 5***

This component resembles one of the TA components in our example. This backend application component runs as a Socket Server running on port 6500. The class java.net.ServerSocket is being run as the server. As with the other components, this has dependency on LatencyActivities as well and performs the activities as defined in the Latencies interface.

e. ***Backend Component 6***

This component also resembles one of the TA components in our example and runs as a RMI server. The *RMIServer* class implements a business interface (*DifffSquareIntf*), which in turn extends java.rmi.Remote class. This has dependency on LatencyActivities and performs the activities as defined in the Latencies interface.

f. ***Latencies Interface***

This is the interface which is implemented by the LatencyActivities class. This interface contains all the different processing activities that introduce various types of latencies to the overall transaction in some way. The activities defined in this interface are similar to the processing activities undertaken by the TA and FA components in our example TCG application. The activity methods are related to database interactions, file I/O, in-memory data processing, request authentication and authorization.

Note: The source code of the Latencies interface has been appended to the Annexes.

*g. **LatencyActivities***

This object contains the methods to perform all the activities resembling the different types of processing activities of the TA and FA components of our example TCG application. It is a Singleton class implementing the Latencies interface. This means only one instance of this class will be instantiated per virtual machine. This was a design decision to make the framework more memory efficient. The various backend services have dependencies on the LatencyActivities object and invoke its operations to perform database interactions, file I/O, in-memory data processing, request authentication and authorization. As this has been made singleton, each of the backend components will run only one instance of the LatencyActivities object. Hence, the activity methods in this singleton class have been made thread-safe by making them "synchronized".

*h. **AuthenticationHandler***

This is the XML handler object responsible for reading the authentication.xml configuration file. It parses the client's security credentials (i.e. userId and password) from the XML file and authenticates the credentials. Upon successful verification of the credentials, the transaction is allowed to be processed. This component resembles the FA component in the sense that it parses and processes XML. The FA Component1 in our example application parses XML feeds to process the news events.

*i. **Authentication.xml configuration file***

This is the XML configuration file for authenticating the client requests. The AuthenticatinHandler object reads this file and validates the client's credentials sent with the operation requests against the credentials specified in this file. The format of the XML file is like this:

```
<credentials>
    <id>sid.kargupta</id>
    <password>Reno1234</password>
</credentials>
```

j. AuthorizationHandler

This is the XML handler responsible for reading the authorization.xml configuration file and authorizing the client requests. Through this authorization process, the requests are granted appropriate access rights and privileges based on the user credentials. Based on the request's user, some requests are granted read/write/execute privileges during the transaction, some others are granted only read/write privileges while some others are only granted read privileges. Like the AuthenticationHandler, this component also resembles the FA component in the sense that it parses and processes XML data.

k. Authorization.xml configuration file

This is the XML configuration file for authorizing the client requests. The AuthorizationHandler object reads this file and based the client's credentials grants relevant access rights and privileges to the requests. The format of the XML file is shown below:

```
<privileges>
    <user>
            <id>sid.kargupta</id>
            <privilege>X</privilege>
    </user>
    <user>
            <id>sami.kargupta</id>
            <privilege>R</privilege>
    </user>
</privileges>
```

A lower level static model (Class Hierarchy diagram) of the application framework is shown in Figures 9 and 10 below:

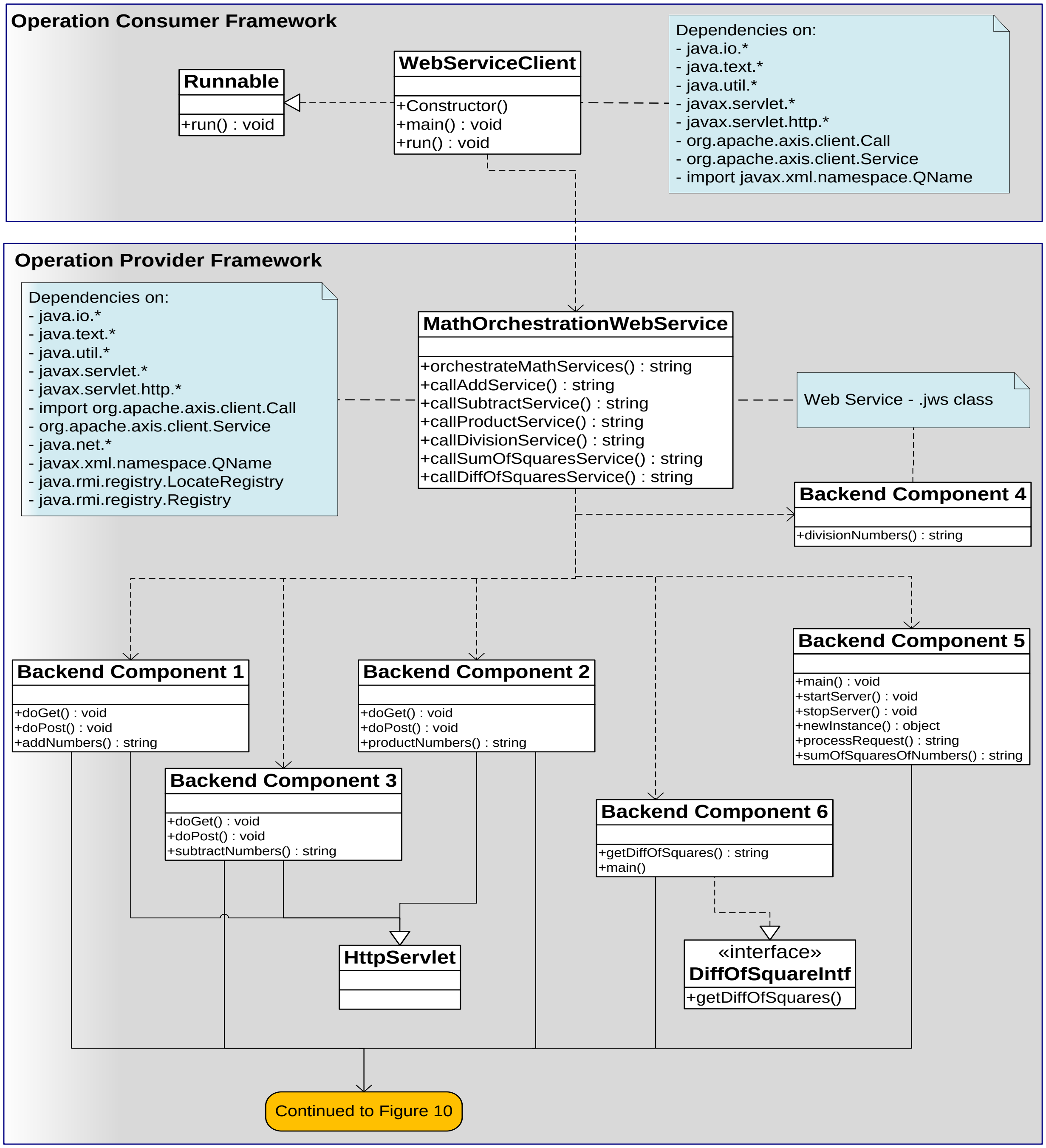

Operation Consumer Framework
Runnable
+run() : void
WebServiceClient
+Constructor()
+main() : void
+run() : void
Dependencies on:
- java.io.*
- java.text.*
- java.util.*
- javax.servlet.*
- javax.servlet.http.*
- org.apache.axis.client.Call
- org.apache.axis.client.Service
- import javax.xml.namespace.QName
Operation Provider Framework
Dependencies on:
- java.io.*
- java.text.*
- java.util.*
- javax.servlet.*
- javax.servlet.http.*
- import org.apache.axis.client.Call
- org.apache.axis.client.Service
- java.net.*
- javax.xml.namespace.QName
- java.rmi.registry.LocateRegistry
- java.rmi.registry.Registry
MathOrchestrationWebService
+orchestrateMathServices() : string
+callAddService() : string
+callSubtractService() : string
+callProductService() : string
+callDivisionService() : string
+callSumOfSquaresService() : string
+callDiffOfSquaresService() : string
Web Service - .jws class
Backend Component 4
+divisionNumbers() : string
Backend Component 1
+doGet() : void
+doPost() : void
+addNumbers() : string
Backend Component 2
+doGet() : void
+doPost() : void
+productNumbers() : string
Backend Component 5
+main() : void
+startServer() : void
+stopServer() : void
+newInstance() : object
+processRequest() : string
+sumOfSquaresOfNumbers() : string
Backend Component 3
+doGet() : void
+doPost() : void
+subtractNumbers() : string
Backend Component 6
+getDiffOfSquares() : string
+main()
HttpServlet
«interface»
DiffOfSquareIntf
+getDiffOfSquares()
Continued to Figure 10

Figure 9: **Static model of the Application Framework**

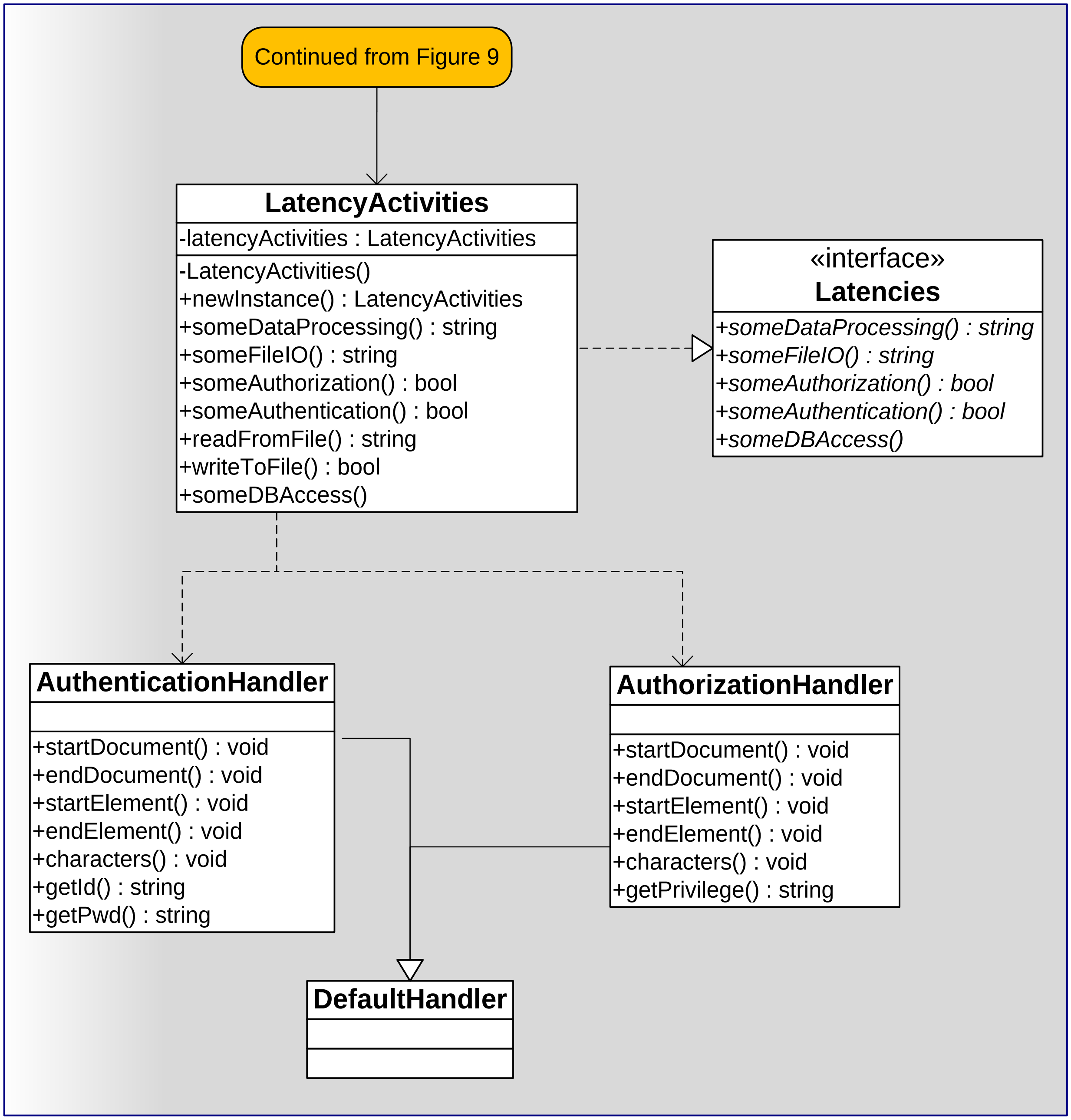


Figure 10: **Static model of the Application Framework (continued)**

4.4 Snapshot of the Application Framework Object Types

Class/Interface Name	Extends (E)/ Implements (I)	Type
WebServiceClient	I – java.lang.Runnable	Multi-threaded Java Client
MathOrchestrationWebService	N/A	Java Web Service
Backend Component 1	E - javax.servlet.http .HttpServlet	Servlet
Backend Component 2	E - javax.servlet.http .HttpServlet	Servlet
Backend Component 3	E - javax.servlet.http .HttpServlet	Servlet
Backend Component 4	N/A	Java Web Service
Backend Component 5	N/A	Java class wrapping a Server Socket
DiffOfSquareIntf	E – java.rmi.Remote	Java Interface
Backend Component 6	I - DiffOfSquareIntf	Java class wrapping a RMI server
ServletClient	N/A	Java class wrapping a client to any Servlet
Latencies	N/A	Java Interface
LatencyActivities	I - Latencies	Java utility class for all the activities causing latencies to Services
AuthenticationHandler	E - org.xml.sax.helpers. DefaultHandler	SAX Parser
AuthorizationHandler	E - org.xml.sax.helpers. DefaultHandler	SAX Parser

4.5 Processing of a Transaction Request to the Operation

In our example, the Auto-Trader application initiates a request to get the trade confidence level from the TCG application. The request is made to the getConfidence(…) operation exposed through a web service from the TCG application. Upon receipt of the request, the credentials are first authenticated and authorised. After successful completion of the authentication and authorization, the Process Orchestrator calls the TA Components (1 to 5) and the FA Component1 in a particular sequence. Each of the TA components and the FA component performs their respective in-memory data processing, file I/O activities and database interactions. The data is then aggregated by the orchestrator and response sent back to the client i.e. the Auto-Trader application. In a similar way, the *facade* orchestration web service of our experimental framework is invoked by the web service client (operation consumer) and it in turn calls the different back end application components one after the other. Each of the backend components then extracts the data to be processed from the input request and performs the following activities:

1. *Gets a Singleton instance of the LatencyActivities object*:

   For overall efficiency of memory usage, the LatencyActivities class has been designed as a Singleton i.e. only one object of the class will be instantiated per virtual machine. The component gets the singleton object and calls the authentication and authorization methods on the LatencyActivities object. For authentication and authorization of requests through XML data parsing, two objects namely AuthenticationHandler and AuthorizationHandler are used. These two objects extend the DefaultHandler class in org.xml.sax.helpers package.

2. *Performs authentication of the request for transaction*:

   The LatencyActivities object has a method which performs authentication on incoming requests for transactions. It reads an XML configuration file and authenticates the User Id and Password supplied with the request. Upon successful authentication, the transaction is allowed to proceed to the next step. Otherwise, the transaction processing is aborted.

3. *Performs authorization of the request:*

   The LatencyActivities object has a method which performs authorization on incoming requests for transactions. It reads an XML configuration file and authorizes the request to perform certain activities. The XML file contains the User to Permissions mappings. Based on the requestor's credentials, specific permissions are granted to the request for transaction.

4. *Performs in-memory data processing:*

Upon successful authentication and authorization, the component then calls the data processing method on the LatencyActivities object, which does some in-memory data processing. Volume of processing is determined by the value of the *dataProcessLoop* parameter, which is set by the client side when invoking the transaction request.

5. *Performs File I/O:*

After completion of the data processing, the component calls a method on the LatencyActivities object, which does some file read/write on the file system. Magnitude of file I/O is determined by the value of the *fileIOLoop* parameter, which is set by the client side when invoking the transaction request.

6. *Performs database interactions:*

After completion of the file input/output activities, the backend component interfaces with an Oracle 10g database. Based on the permission granted through authorization, it does read and write operations. Intensity of database interactions is determined by the value of the *dbAccessLoop* parameter, which is set by the client side when invoking the transaction request.

7. *Applies component specific algorithm:*

Every backend component performs some activities specific to the component. At the end of all the LatencyActivities defined processing activities, the component calls its specific method to perform the activity based on the type of component.

8. *Returns data back to the calling Orchestration Web Service:*

After all the transaction activities of the particular backend component are completed and processing done, the response is sent back to the orchestration web service. Upon receipt of this response, the orchestration web service calls the next backend component. This process carries on till the last backend component is called. Upon receipt of the response from the last backend component, the orchestration web service in turn sends the response back to the web service client.

The Sequence diagrams as shown in Figures 11 and 12 below illustrate the sequence of activities that take place behind the main orchestration web service of the application operation provider for Backend Component 1 and Backend Component 2.

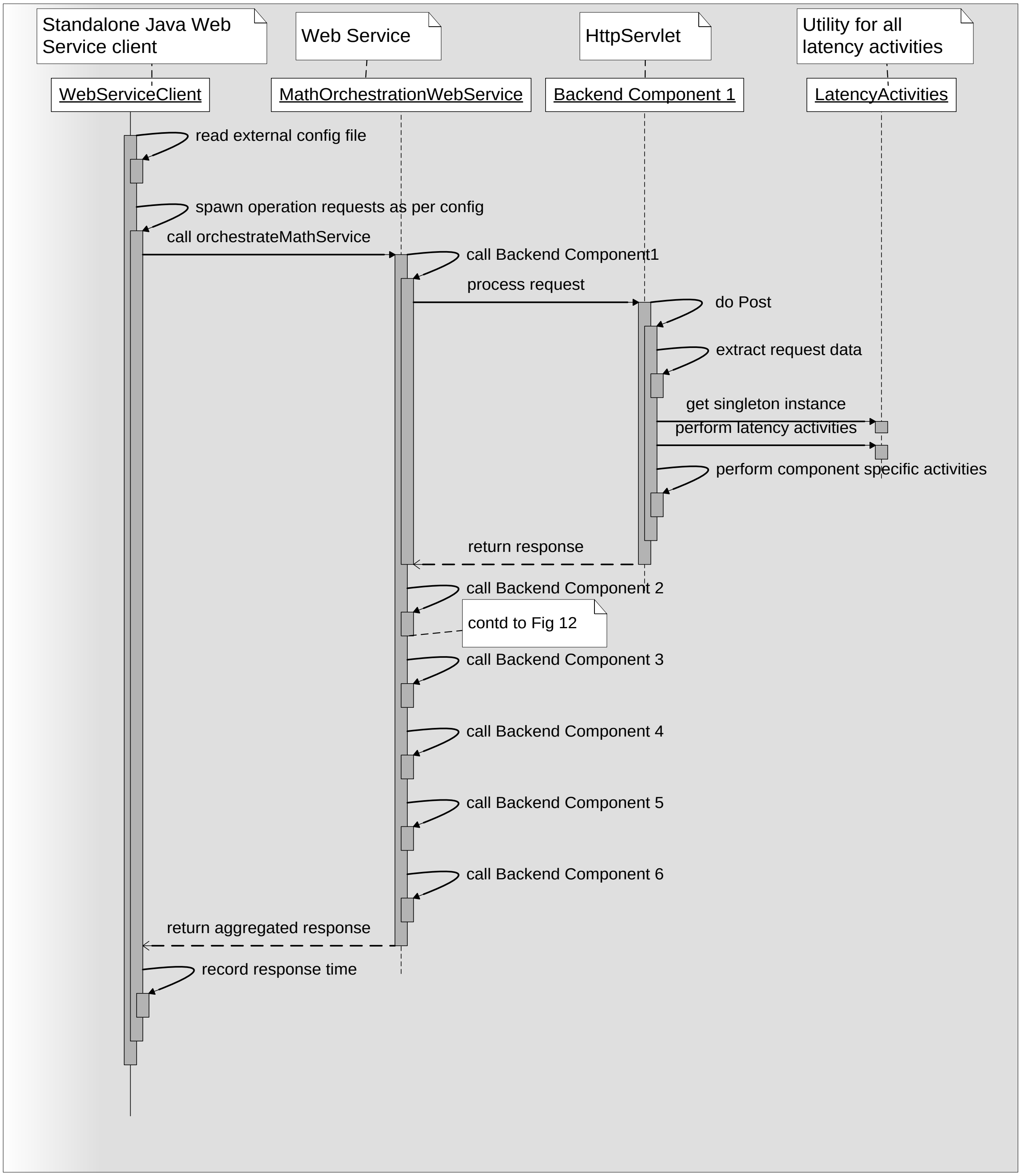


Figure 11: **Sequence model for Backend Component 1 activities**

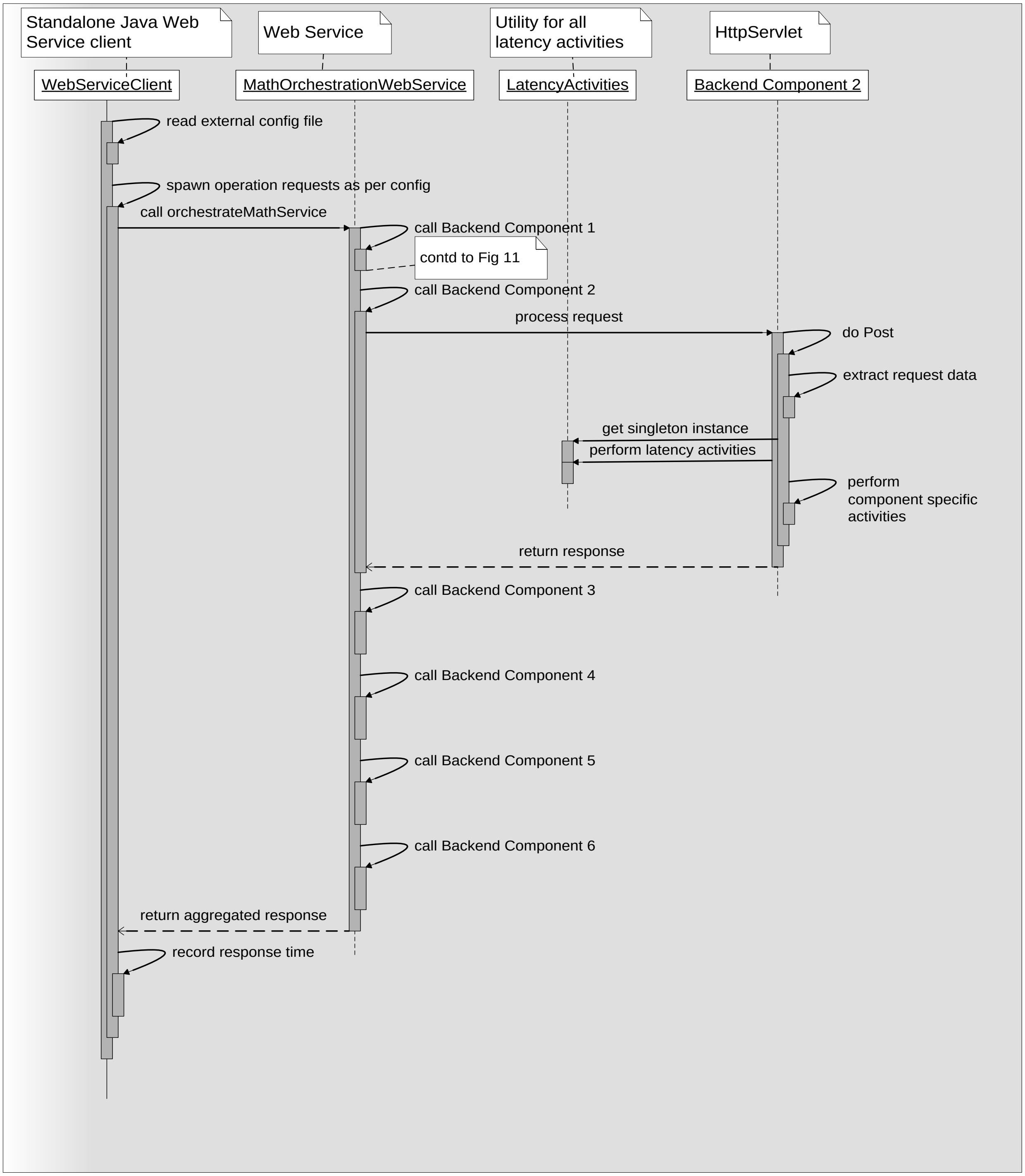


Figure 12: **Sequence model for Backend Component 2 activities**

Figure 13 below is a lower level Sequence diagram which shows the different methods invoked on the LatencyActivities object from Backend Component 1. The same happens for each of the other backend application components.

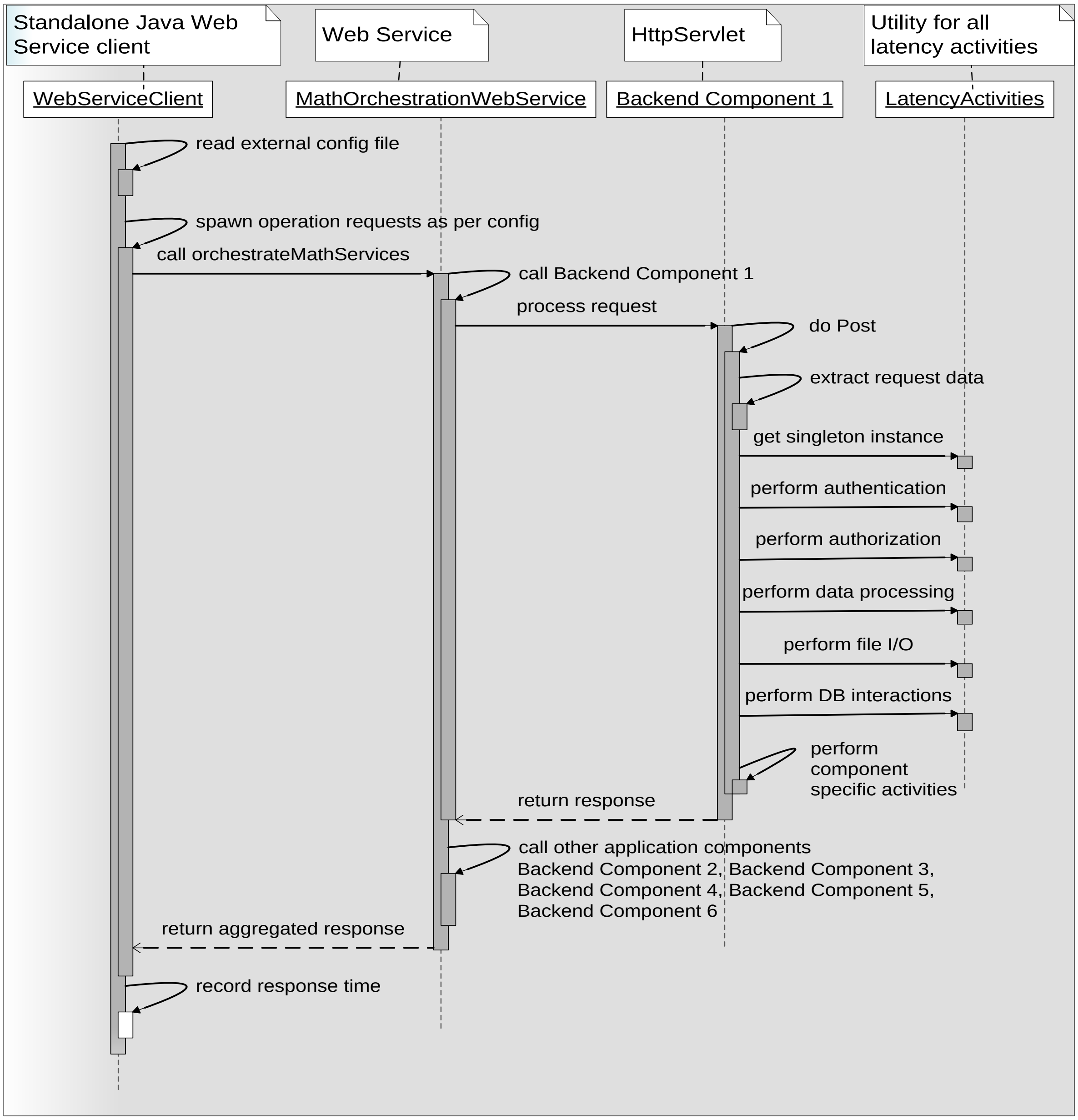

Standalone Java Web Service client
Web Service
HttpServlet
Utility for all latency activities
WebServiceClient
MathOrchestrationWebService
Backend Component 1
LatencyActivities
read external config file
spawn operation requests as per config
call orchestrateMathServices
call Backend Component 1
process request
do Post
extract request data
get singleton instance
perform authentication
perform authorization
perform data processing
perform file I/O
perform DB interactions
perform component specific activities
return response
call other application components Backend Component 2, Backend Component 3, Backend Component 4, Backend Component 5, Backend Component 6
return aggregated response
record response time

Figure 13: **Lower level Sequence diagram for Backend Component 1**

The Sequence Diagrams of the activities that take place behind the main orchestration web service of the application operation provider for Backend Component 3, Backend Component 5 and Backend Component 6 are included in Annexes A.

4.6 The Hosting Environment

To compare the results and analyze the generic nature of the proposed methodology, controlled experiments are performed separately on the same application framework implemented on different hosting environments. The main purpose was to observe the impact of augmented hardware on the various measured data under similar workload conditions.

4.6.1 Hosting Environment 1 (HE1)

The application framework is run on a Dell Server with Intel Pentium 2.00 GHz Processor, Single Core, 2.00 GB RAM, 32-bit operating system. The operating system is MS Windows XP Professional, version 2004, Service Pack 3. The orchestration web service is hosted on a Tomcat container with maxThreads set to 150 on the connector port to cater to those many concurrent request threads.

4.6.2 Hosting Environment 2 (HE2)

The application framework is run on a HP G56 Server with Pentium 2.30 GHz, T4500 Processor, Dual Core, 4.00 GB RAM, 64-bit operating system. The operating system is MS Windows 7 Premium, Service Pack 2. Hosting of the orchestration web service is the same as the first environment, i.e. it is hosted on a Tomcat container with maxThreads set to 150 on the connector port to cater to those many concurrent request threads.

4.7 Chapter Summary

In this chapter we delved into the implementation specifics of the prototype application framework developed to resemble our example TCG application and other real life applications. Controlled experiments are performed on this application framework in a controlled environment. It describes the application operation provider and consumer sub-frameworks and the application components forming the sub-frameworks respectively. The operation’s transaction process and the associated sequence of actions that are triggered from a call to the exposed web service operation are explained in details. The backend interactions between the facade orchestration web service and the various backend application components are explained with the aid of UML Sequence diagrams. The static class hierarchy of the application framework is presented with the aid of UML Class diagrams. Some of the environmental constraints that have been observed in this thesis are discussed. The same prototype application framework is run separately on two separate hosting environments to compare the results and observe the impact of augmented hardware on the various measured data under similar workload conditions. The different hosting environments have been described.

5 Evaluating the runtime *PALP* Model

We begin with a high-level overview of the objectives of our evaluation and the corresponding experiments. The goal of the evaluation presented in this chapter is to validate the first hypothesis as stated in Section 3.3.2 and establish the relational *PALP* model between an application operation's ***P***erformance ('P'), its ***L***oad ***P***otential ('V') and its ***A***dmittance ('Y'). Firstly, through controlled experiments, it is observed how 'P' varies as a function of 'V' with varying inbound workload. Section 3.3.1 explains the rationale behind assuming constancy of 'Y' at runtime. In relation to this deduction, the objective then is to demonstrate that 'P' may be expressed as a linear function of 'V' for a given range of 'V' (referred to as "V Bands" in Figure 17) and 'Y' is the gradient constant for that function of 'V' for a given class of workload and a given hosting environment including the network. We present a mathematical deduction to support this thesis's intrinsic definition of 'Y' which suggests that an increase in 'Y' will result in an increase in 'P' with the workload and other conditions remaining constant. As prescribed by (Wohlin et al., 2012), the statistical power of the experiments is determined to demonstrate the ability of the experiments to reveal the true pattern in the collated data.

This evaluation and establishing the *PALP* model will help to address the aspects f), g) and h) of the main research question as explained in Section 1.3. The model will enable a "bottom-up" approach towards achieving a *target* performance criterion for an operation by fine-tuning the timeliness of the various processing activities of the operation under a specified inbound workload condition without the need of repetitive load testing as done in CPM. It will also allow the conventional "top-down" approach of performance assessment and enable the software engineers to assess the impact on the performance of an application operation due to changes to the configuration and algorithm of the operation, under a given inbound workload. It will also enable the software engineers to ascertain the inbound workload that an application operation may sustain while maintaining a *target* performance level for a given configuration and algorithm of the application.

For the evaluation methodology, we follow the guidelines on experimentation in software engineering as they are presented in (Wohlin et al., 2012). While keeping other variables constant, different treatments are applied to or by different subjects and the effects on outcome variables are measured. Controlled experiments are performed in a laboratory environment, which provides a high level of control. When experimenting, the objective is to manipulate one or more variables and control all the other variables at fixed levels. The effect of the manipulation is measured and based on this measurement, statistical analysis is performed. Details of the controlled experiments are provided in Section 5.1 below.

5.1 Empirical Evaluation of the Model

In the context of the *PALP* model, the first hypothesis of this thesis stated that for a given class of workload, a given hosting environment and a given range of Load Potential *'V'*, an application operation's Performance *'P'* can be expressed as a linear function of its Load Potential *'V'* at runtime and the gradient constant is the runtime Admittance *'Y'* of that operation. The hosting environment includes the connecting network between the application and the operation consumer. In this section, we describe the experiments performed to evaluate the first hypothesis and establish the *PALP* model.

To demonstrate the ability of the experiments to reveal the true pattern in the collated data, the statistical power of the experiments is needed to be determined (Wohlin et al., 2012). Hence the first ***null hypothesis*** (**$H1_0$**) is formulated corresponding to the first hypothesis. The null hypothesis states that an application operation's Performance ('P') has no correlation to its Load Potential ('V') at runtime. P cannot be expressed as a function of Y and V for a given range of V, a given class of workload and a hosting environment including the connecting network between the application and the operation consumer.

A *black-box* approach was taken with a Web Server by (Menasce et al., 2002) to model a finite queue system for determining the variation in response time against increase of arrival rate of requests per second. To evaluate the *PALP* model as stated in the first working hypothesis, a *black-box* approach similar to Menasce et al is taken towards the application operation under observation.

Using the prototype application framework described in Chapter 4, experiments are performed iteratively to empirically establish a high-level *runtime* abstract model between an application operation's performance, its load potential and its admittance by applying established mathematical techniques. Tests are run by gradually increasing the inbound workload to the application operation. In our example, this scenario resembles assessing the variation in the response times of the getConfidence(…) operation of TCG application for gradual increase in request workload from the client Auto-Trader application. Under a particular load configuration, multiple transaction requests (the number depending on the load configuration) are spawned by the web service client. The Average Response Time (ART) for all the operation requests provided an indicative measure of the response time for that particular workload condition. We considered applying Set Theory (Saltzman, 2013) for obtaining a representative operation response time under a given workload condition. However, during some initial tests, while obtaining the response times for the same workload condition, we observed some "noise" (very minimal though) across a few runs. Although, the variations were in the order of sub-milliseconds, applying set theory would have been difficult as there were no intersections of datasets. Also, from the data coverage point of view we wanted to include all the data as the real life systems may behave in the same way. So we calculated the mean of the response times for the same workload to obtain a response time representative of all the individual response times for that particular workload condition. For different inbound workload configurations Set Theory wouldn't have been applicable in any case.

Initial experiments showed that the variation of the ARTs was minimal across test runs for the same inbound workload configuration. To assess whether the ARTs are spread across a wide range of values, the Standard Deviation of the ARTs for 5 test runs is computed. The result indicated a very low deviation. Hence, for one given inbound workload condition, '5' is considered a reasonably optimal number for test run iterations. The test runs are conducted under a given workload condition and the ART for each of the test runs is recorded. In (Cherkasova et al., 2007) the 50% percentile value of the service time Cumulative Distribution Function is taken as the measure of the service time for a particular type of transaction under a given workload. We could have taken something similar like the statistical mean of the ARTs across the 5 test runs for a given inbound workload. However, worst-case tolerancing is a safer approach than statistical tolerancing. It ensures that if the inputs are within their respective tolerances, the output is guaranteed to be within its worst-case tolerance (Taylor, 1995). As the purpose of the experiments was to determine the impact on performance due to varying workload conditions, the highest of the 5 ARTs (i.e. the worst-case performance) is recorded for each load condition instead of the statistical mean.

Tests are performed against different increasing inbound workload conditions. As Stress Point depends on the business requirements and is typically constant for a given application configuration and environment setup, for the purpose of our controlled experiments, we assume the Stress Point to be 40 requests/second. From the operation consumer (web service client), transaction requests are spawned for a fixed duration of time (5 seconds in our experiment) for every test run. Once all the 5 test runs are executed for a particular workload condition, the inbound workload is increased to the next step. To increase the rate of generated requests (i.e. increase the workload), the time interval between requests being generated is reduced in steps of 100 milliseconds initially and then 10 milliseconds. Every step represented a particular workload condition for which 5 test runs were repeated. The Load Potential 'V' is calculated from the difference of the assumed Stress Point and the generated workload. The application operation's performance data is recorded against the variability of the operation's Load Potential. The data obtained demonstrated a *finite queue system curve* (Menasce et al., 2002) and reached near saturation level after the 9th workload condition for the particular hosting environment.

Details of the experiment steps are shown in the form of Flow Charts in Figures 14 and 15 below. Figure 14 shows the high level execution flow of the experiment runs. The 'Load Configuration count' keeps track of the experiment runs for different load configurations. For each load configuration, 5 tests are run, which is tracked by the 'Test Run count'.

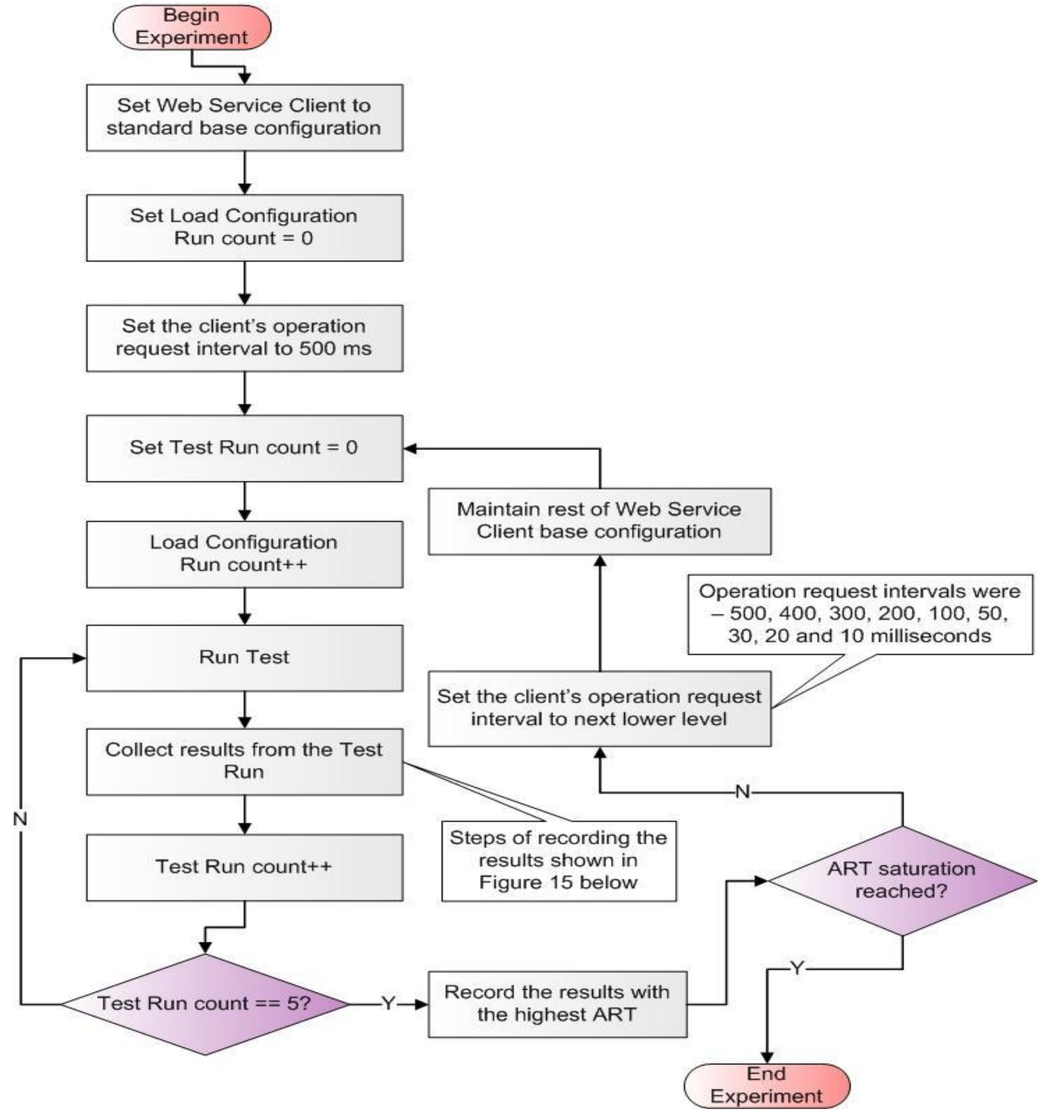

Begin Experiment
Set Web Service Client to standard base configuration
Set Load Configuration Run count = 0
Set the client's operation request interval to 500 ms
Set Test Run count = 0
Load Configuration Run count++
Run Test
Collect results from the Test Run
Test Run count++
Test Run count == 5?
N
Y
Record the results with the highest ART
Steps of recording the results shown in Figure 15 below
ART saturation reached?
N
Y
End Experiment
Set the client's operation request interval to next lower level
Operation request intervals were – 500, 400, 300, 200, 100, 50, 30, 20 and 10 milliseconds
Maintain rest of Web Service Client base configuration

Figure 14: Flow Chart to determine relationship between P and V

Figure 15 shows the steps of recording the data for each of the 5 test runs for a given workload configuration. This figure shows the details of the 'Collect results from the Test Run' step of the flow chart shown in Figure 14.

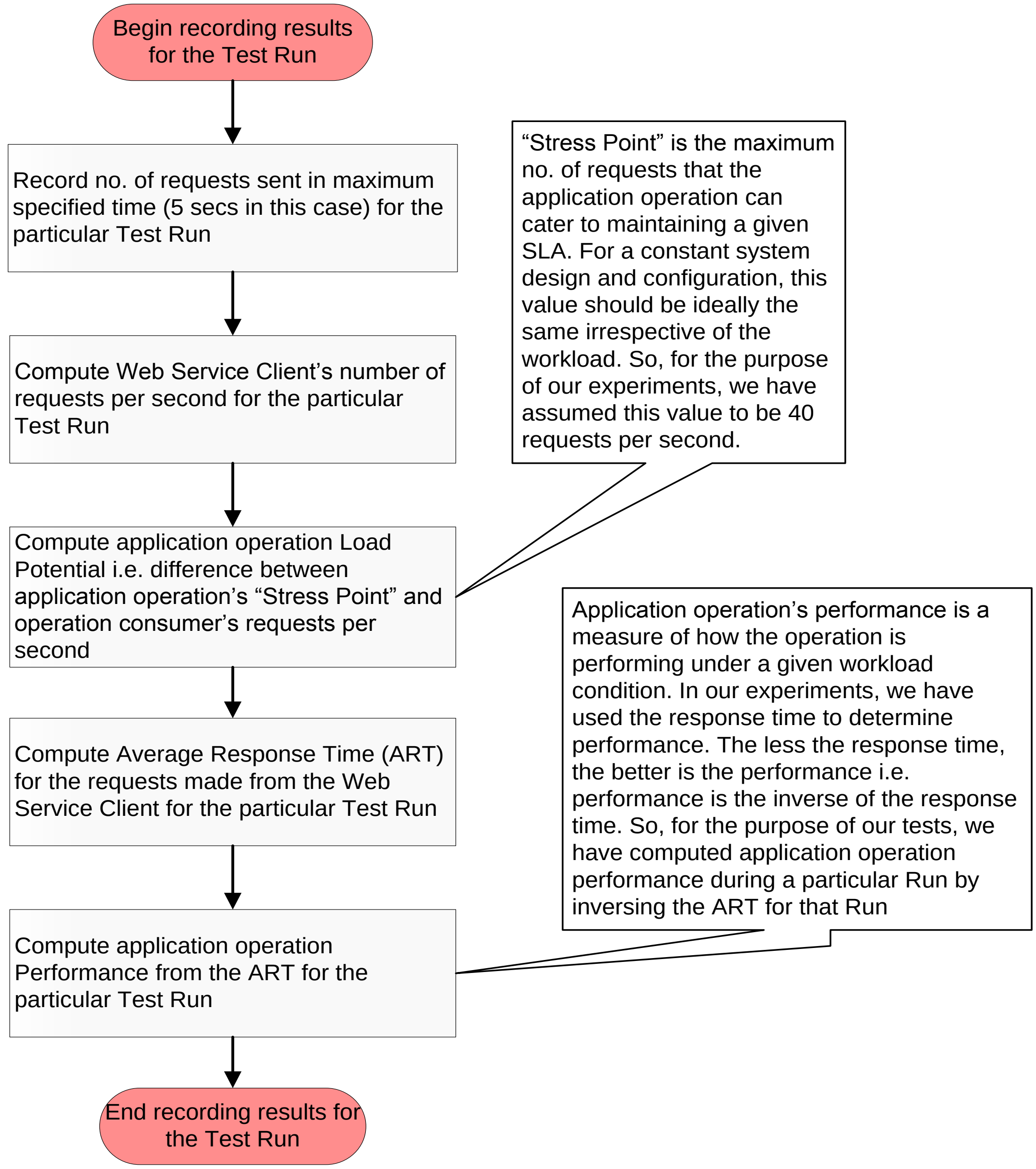

Begin recording results for the Test Run
Record no. of requests sent in maximum specified time (5 secs in this case) for the particular Test Run
Compute Web Service Client's number of requests per second for the particular Test Run
Compute application operation Load Potential i.e. difference between application operation's "Stress Point" and operation consumer's requests per second
Compute Average Response Time (ART) for the requests made from the Web Service Client for the particular Test Run
Compute application operation Performance from the ART for the particular Test Run
End recording results for the Test Run
"Stress Point" is the maximum no. of requests that the application operation can cater to maintaining a given SLA. For a constant system design and configuration, this value should be ideally the same irrespective of the workload. So, for the purpose of our experiments, we have assumed this value to be 40 requests per second.
Application operation's performance is a measure of how the operation is performing under a given workload condition. In our experiments, we have used the response time to determine performance. The less the response time, the better is the performance i.e. performance is the inverse of the response time. So, for the purpose of our tests, we have computed application operation performance during a particular Run by inversing the ART for that Run

Figure 15: Flow Chart to record the Average Response Time of each run

The data obtained from the above experiments to deduce the high level, abstract runtime model associating the application operation's performance 'P', admittance 'Y' and load potential 'V' demonstrated a typical finite queue system graph. The inbound workload is gradually increased resulting in a gradual decrease of ‘V’. As ‘V’ gradually decreased to nearly 35, an abrupt change is observed as the server utilization approached near 100% (Menasce et al., 2002). At this point the throughput of the server approached its maximum throughput sharply and its performance degraded drastically reaching saturation. We observed a typical *finite queue* system curve for the performance ‘P’ versus the operation load potential ‘V’.

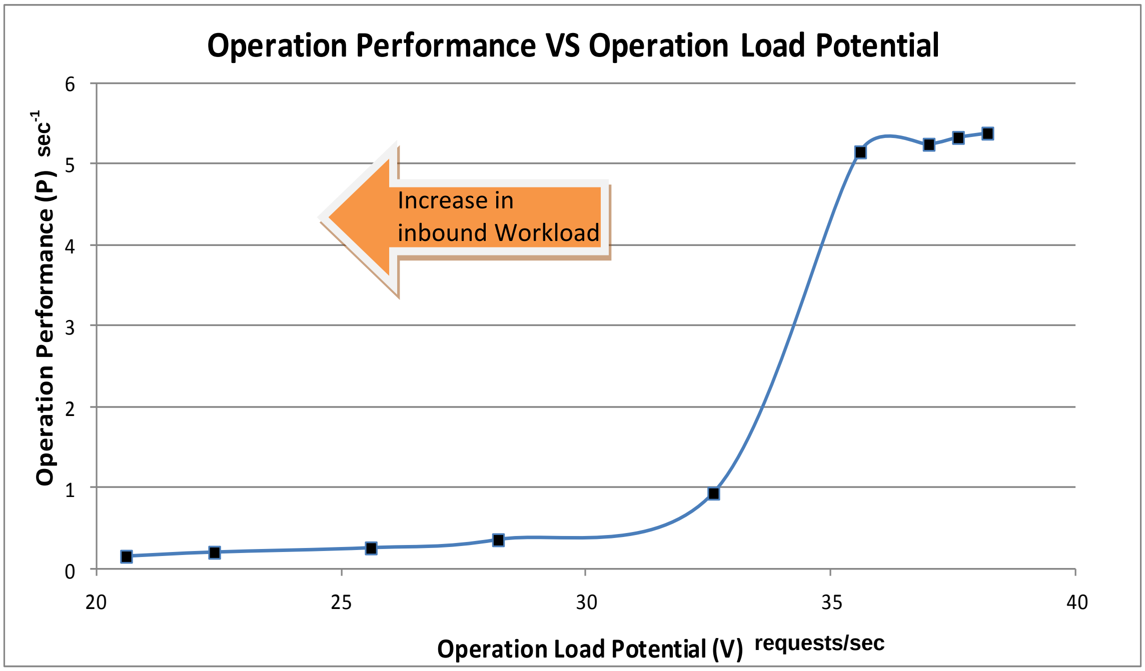


Figure 16: **Empirical Data Graph of P for varying V**

Figure 16 shows the finite queue system curve as observed from the above experiment. In the graph, the X-axis represents the operation load potential. Given the definition of application operation load potential in Section 1.4, which is the difference between the application operation’s “Stress Point” and the application operation’s rate of inbound workload, a decrease in the value of X-axis signifies an increase in the rate of inbound workload. As evident from the graph, a sudden fall in performance occurred when the operation load potential decreased to nearly 35 (from right to left in the diagram). This graph shows ‘P’ as a non-linear function of ‘V’.

To simplify the model, accepting approximation error, *Piecewise Linear Functions* (Fourer et al., 2003) are applied to divide the 'V' values into 3 ranges (or *bands*), each with a *linear regression* as the best fit for 'P'. As shown in Figure 17 below, 'P' is considered as a linear function of 'V' for each of the three ranges/bands of 'V' values:

$$P = mV + c \qquad (5.1)$$

where '*m*' is the gradient constant. The values of '*m*' and '*c*' are specific to the 'V' bands.

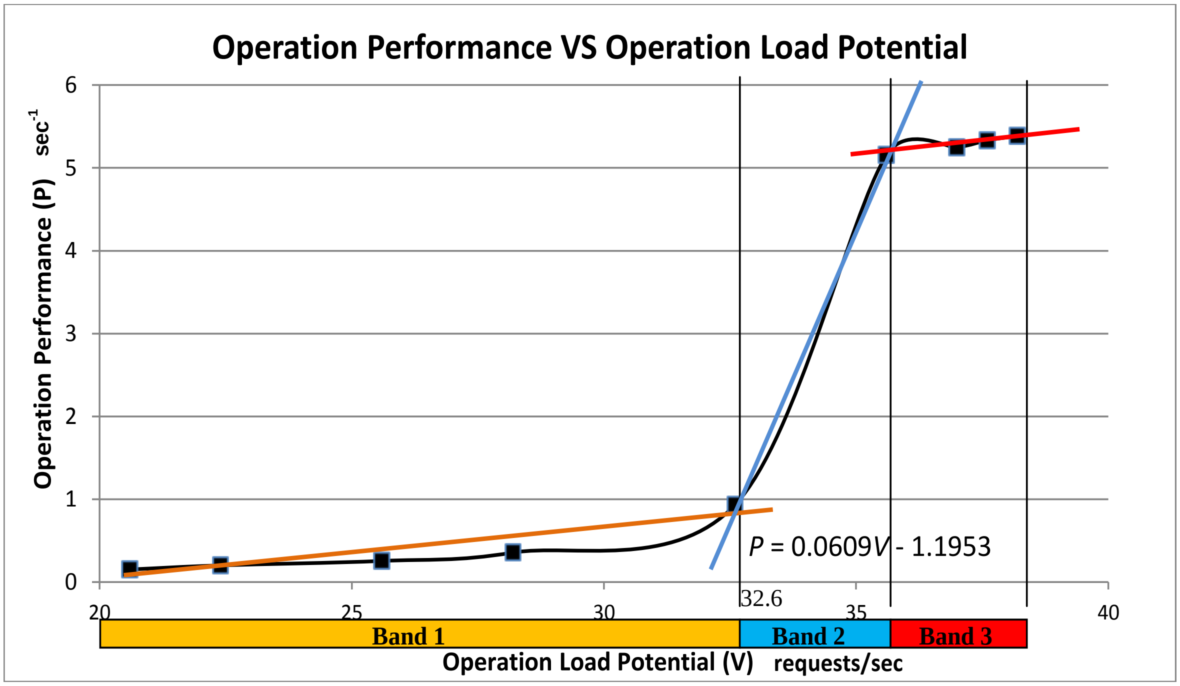


Figure 17: Linear Regression for individual V Bands

5.2 Operation Admittance 'Y' – Constancy and Mapping

It is deduced in Section 3.3.1 that an application operation's admittance 'Y', which is the measure of ease with which a request to the operation is processed and response sent back, is constant for all the transactions of the same type under the same workload condition and hosting environment across time. Given this inherent definition of 'Y', for a given application operation load potential V, increase in 'Y' should result in increase in the application operation performance 'P'.

Figure 18 below, which is an extension of Figure 17 derived from the empirical data, proves the above assumption:

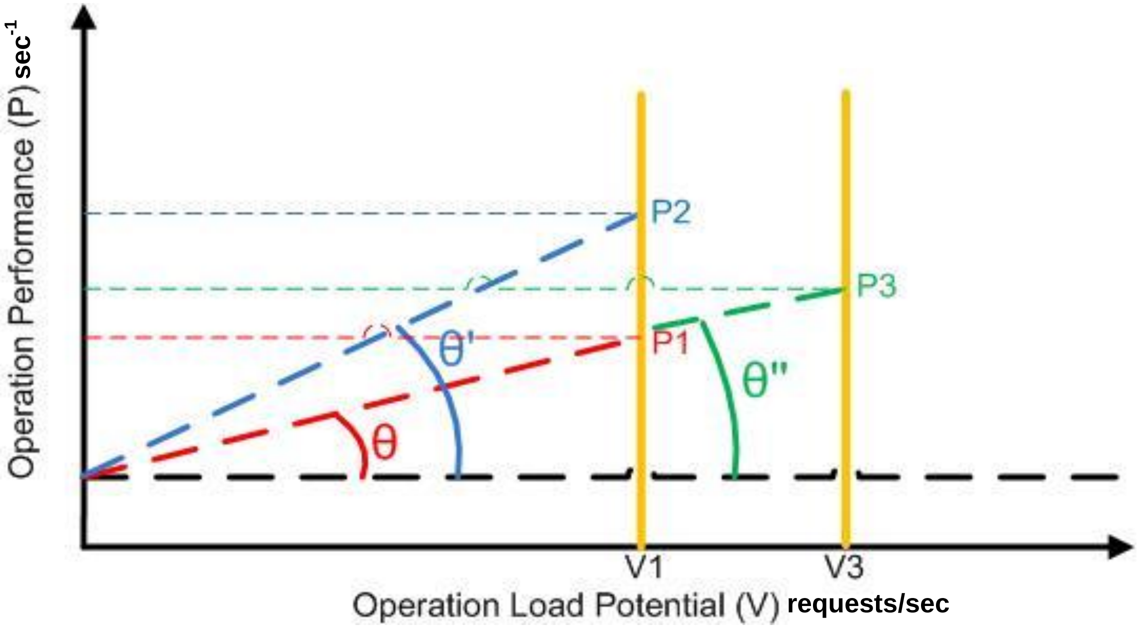

Operation Performance (P) sec^{-1}
P2
P3
P1
θ'
θ''
θ
V1
V3
Operation Load Potential (V) requests/sec

Figure 18: Impact of Admittance on Performance

In Figure 18, for the Operation Load Potential *V1*:

Let the initial operation Admittance *Y* = tanθ = *P1*/*V1*.
Increased operation Admittance *Y*' = tanθ' = *P2*/*V1*
If *Y*' > *Y*,
Then tanθ' > tanθ, i.e. *P2*/*V1* > *P1*/*V1*
This is possible only if *P2* > *P1*.

Hence increase in the operation's admittance 'Y' will boost the application operation performance 'P' for any given application operation load potential 'V'. This deduction supports this thesis's definition of application operation Admittance. In view of this, the runtime 'Y' is mapped to the gradient constant *m* in (5.1). Hence, the application operation performance 'P' can be expressed as:

$$P = YV + c \qquad (5.2)$$

5.3 Verifying the Statistical Power of the Experiment

The above deduction also proves the following:
For constant operation Admittance
tanθ = tanθ''
i.e. *P1/V1 = P3/V3*
i.e. *P1/P3 = V1/V3*

Therefore, change in the operation's Performance ('P') is directly proportional to the change in the operation's Load Potential ('V') for a given range of 'V'. Hence, within the bounds of the conducted experiments, the first *null hypothesis* (**$H1_0$**) is proved to be false and may be rejected.

A *Type-II-error* (Wohlin et al., 2012) would have occurred had the experiments not indicated the proportional relationship between Performance and Load Potential. But as the relationship is demonstrated, we can treat the probability *P*(Type-II-error) as zero within the bounds of the conducted experiments. The statistical power (Wohlin et al., 2012) of an experiment can be expressed as:

$$\text{Power} = 1 - P(\text{Type-II-error}) \qquad (5.3)$$

In our scenario, statistical power = 1 – 0 = 1. Hence, the collated data from the experiments are ideal to reveal the true pattern (Wohlin et al., 2012).

5.4 Threats to the Validity of Findings

This thesis focuses on the application layer of a system. We have evaluated our model on a test-bed application framework designed to resemble real life applications. The framework comprises various types of components like Web Services, Servlets, RMI Server and low level Socket Server. Due to the reasons as explained in Section 3.5, controlled experiments are performed on the test-bed to evaluate the model. Use of such controlled experiments on test-bed applications is a common approach for validation of software performance analysis methods (Menasce et al., 2002). However, strictly speaking, our findings are limited to this test-bed framework. Further large-scale experiments will be needed to validate them on real life applications. The following sections discuss the other threats to validity for the experiments we conducted on our test-bed application framework:

5.4.1 Conclusion Validity

This validity is concerned with the relationship between the treatment and the outcome (Wohlin et al., 2012). Threats to the *Conclusion Validity* of an experiment may impact the ability to reach valid conclusions in a negative manner.

Low Statistical Power – Low statistical power of experiments highlights the inability of the experiments to reveal the true patterns in the collated data. In Sections 5.2 and 5.3, through mathematical deductions we refuted the first null hypothesis $\mathbf{H1_0}$ and proved the first hypothesis to be true. The experiments showed direct proportionality between 'V' and 'P' and there was no Type-II-error. The statistical power of our experiments is proven to be high, highlighting the ability of the experiments to reveal the true patterns in the collated data. As shown in Figures 12 and 13, for each workload configuration, 5 test runs are executed with each test run comprising 30 transaction requests of the same class of workload. Data from the test run with the highest ART is recorded. Different workload configurations are tested till the ART reached near saturation after the 9th configuration. To ascertain the pattern and trend of impact on the operation's Performance due to variation in its Load Potential, *1350* sets of request-response and probed data are observed. This is not a small number and signifies good data coverage.

Violated Assumptions of Tests - In Section 3.3.1 it is explained that an application operation's admittance 'Y', which is the measure of ease with which a request to the operation is processed and response sent back, is constant for all the transactions of the same type under the same workload condition and hosting environment across time. Given this inherent definition of 'Y', for a given application operation load potential V, increase in 'Y' should result in increase in the application operation performance 'P'. To deduce this, several boundary conditions are assumed. However, if any of the boundary conditions is violated, the gradient of 'Y' may be impacted as the degree of change of 'P' for variation to 'V' may alter. For example, this thesis deals with changes to the application layers and technology refreshes are out of scope. If changes are applied to the underlying infrastructure, the rate of change of 'P' for changes to 'V' will be different on the new platform and hence the gradient of 'Y' may vary. The function of 'Y' and 'V' has to be recreated. Another example is that only *formal* application operation consumption contracts are in scope of this thesis with dedicated, controlled network traffic and *not* any random application access over public network. Hence, at *runtime*, no unpredictable fluctuation of network bandwidth or latency is assumed. If this assumption is violated, the runtime constancy of ‘Y’ may be affected. Any variation to the workload class will also impact ‘Y’. Our experiments are performed in controlled environments and all of the boundary conditions are observed. Hence, within the bounds of the experiments, no assumption is violated.

Fishing and the Error Rate – Searching for a specific result is a threat since the analyses are no longer independent and the researchers may influence the result by looking for a specific outcome (Wohlin et al., 2012). In our case, instead of any specific outcome, every outcome was observed and recorded with the aim of extracting patterns out of the collated data. The Error Rate is concerned with the significance level or statistical power of the experiments. In Section 5.3, the statistical Power of our experiments is demonstrated to be high, highlighting the ability of the experiments to reveal the true patterns in the collated data.

Reliability of Treatment Implementations – Our experiments are fully automated with only the instrumentation towards probing the response times of the operations done manually for the time being. The instrumentation is done once upfront without any further alteration during the course of the experiments, eliminating the risks of any difference in type of measurements across different test runs. Also, the experiments do not involve any human participants. Hence the risk of dissimilar implementations between different persons applying the treatment is eliminated too.

Random Irrelevancies in Experimental Setting – Elements outside of the experimental setting may disturb the experimental results. Our experiments are conducted in a controlled environment. As explained in Section 3.3.1, several boundary conditions related to the QN model, application hosting environment, network provisioning and others are observed during the experiments. This reduced the possibility of any external impact on the experiments to a great extent.

Random Heterogeneity of Subjects - Real workloads are collections of heterogeneous components. Concerning the type and level of resource usage, a request involving complex database query will differ significantly than a request involving simple in-memory data processing. Partitioning the workload and classifying the request types increase the predictive power of the model. To classify the request type, we restricted our model and methods to operations of an application. At a given time T_1, for a particular type of transaction request to the same application operation, the request type, the process logic to serve the transaction, the system configuration, the resource requirements and the contract request load condition will be ideally the same. Hence, the lookup datasets and patterns extracted from the experiments are applicable to a particular application operation associated to a single class of workload. The pattern of variation of '*P*' against changes to '*V*' for an operation may differ from the pattern of 'P' versus 'V' for another operation of the same application.

5.4.2 Internal Validity

The *Internal Validity* of an experiment concerns the degree to which the independent variables actually influence the observed outcome. In our experiments, we had specific independent variables and a specific dependent variable being measured. This eliminated the *Single Group Threat*, which imposes problems in determining if the treatment or another factor caused the observed effect. Constancy of the experimental environment is maintained by observing all the boundary conditions related to the QN model, application hosting environment, network provisioning and others as explained in Section 3.3.1. In this way the risk that the *history* will affect the experimental results due to changes in the circumstances is avoided in our experiments. Also, the experiments are performed in a fairly controlled environment without any unexpected or unforeseen factors influencing the resultant data. Lastly, the experiments do not involve any human participants. Hence, effects of motivation, maturation and other such factors do not apply. All of these ensured that the *Internal Validity* of our experiments is maintained.

5.4.3 Construct Validity

The *Construct Validity* of experiments describes the extent to which the independent and dependent variables represent the real-world cause and effect under investigation. Some threats to *Construct Validity* relate to the design of the experiment and other social factors.

Design Threats - The application operation provider – consumer prototypical framework comprising a multi-threaded operation consumer, a consumer facing web service acting as the operation provider, other backend application components and some other utility components is built to resemble a real-life scenario. For the purpose of the experiments, some illustrative application component level Delay Points with activities such as Database Interactions, In-Memory Data Processing, File I/O, Request Authentication and Request Authorization involving XML parsing etc.

are also created. While this framework covers a wide range of different types of components supporting an application operation and we have reasons to assume that it simulates many real-world frameworks, it is practically not possible to incorporate all the various types of component complexities in the experimental framework. In this thesis, instrumentation for probing the operation's response time has been done manually. As future extension, the instrumentation may be automated through the use of Aspect Oriented Language. Tool support will expedite the instrumentation process otherwise manual insertion of the probes may be time consuming. However, automation may be difficult if the same type of Delay Points is implemented in different ways in different real-world applications. Single core and dual core machines have been used during the experiments. Under certain specified workload conditions, the patterns of variation of the Performance of an application operation due to changes to the operation's Load Potential are extracted. However, nowadays, extreme high spec machines with many cores are available, which are typically deployed in the production environments. The workload injected during our experiments may not be sufficient to impact the Performance on those types of high spec hosting environments to the extent that patterns of variation can be extracted. More workload may be needed to be injected.

While we have covered a range of combinations of treatments in the experiments, it is possible that we may not have explored all the combinations of treatments in our experiments. There are many more types of treatments that may be applied to observe the effects. In the *Future Work* section (9.4), we have briefly discussed some of the possible extended experiments that may be undertaken in the future.

Social Threats – These threats are concerned with issues related to the behavior of the subjects and the experimenters. They may, based on the fact that they are part of an experiment, act differently than they do otherwise, which results in false outcome from the experiment. However, our experiments do not involve any human participants and hence these *social threats* are mitigated to a great extent.

5.4.4 External Validity

The *External Validity* of an experiment signifies how well its results generalize to industrial practice in a wider context.

Interaction of Selection and Treatment – In our experiments we have designed and developed various types of application processing activities, which are commonly found in real life industrial applications. These are being referred to as the Delay Points in this thesis. Some examples of these

Delay Points are in-memory Data Processing, File I/O activities, Database Interactions and other types. As shown in Figures 12 and 13, for each workload configuration, 5 test runs are executed with each test run comprising 30 transaction requests of the same class of workload and representative of real life scenarios. Data from the test run with the highest ART is recorded. Different workload configurations are tested till the ART reached near saturation after the 9th configuration. To ascertain the pattern and trend of impact on the operation's Performance due to variation in its Load Potential, *1350* sets of request-response and probed data are observed. This is not a small number and signifies good data coverage. Even though we have reasons to assume that the experiments covered reasonably good amount of real life representative data, there still may be other types of workload configurations, which we have not been able to cover within the scope of this thesis.

Interaction of History and Treatment – This concerns with any temporal effect on the results of the experiments. Our experiments are fully automated with only the instrumentation towards probing the operation response times performed manually for the time being. The instrumentation is done once upfront without any further alteration during the course of the experiments, eliminating the risks of any difference in type of measurements across different test runs over time. Also, the experiments do not involve any human participants. Hence the risk of dissimilar implementations towards the treatment over time is eliminated too.

5.4.5 Repeatability

Finally we consider whether our experiments are repeatable so that independent verification of our results is possible. We have described the experiment environment in details in Chapter 4. This includes the details of the high level prototypical application framework, the lower level operation consumer and provider sub-frameworks, the multi-threaded operation client, the client side configuration files along with its parameters, the server side application operation, its details and the application components supporting the operation. Details of the Java implementation are provided including the classes and object interactions, which are documented through UML models (Static and Sequence Diagrams). The details of the two hosting environments are specified in Chapter 4 as well. This should enable replication of our experiments with different sets of configurations.

5.5 Chapter Summary

This chapter described the steps of the experiments run on the prototype application framework to evaluate the first working hypothesis of this thesis. It is in the context of the *PALP* model and states that for a given class of workload, a given hosting environment including the connecting network between the application and the operation consumer and a given range of ‘V’, the application operation’s ‘P’ can be expressed as a linear function of ‘V’ with ‘Y’ as the gradient constant. The first null hypothesis (**H1**$_{\mathbf{0}}$) is proved false in this chapter.

Previous work by (Menasce et al., 2002) and (Cherkasova et al., 2007) formed some of the basis of the key assumption that a particular application operation’s admittance is constant at runtime under a given load band. Through controlled experiments on the application framework, it is proved that given certain constraints related to the inbound workload, the hosting environment and the range of ‘V’, the runtime relationship between the three key attributes ‘P’, ‘Y’ and ‘V’ may be represented in the form:

$$P = YV + c.$$

Accepting approximation error, the non-linear function of ‘P’ against ‘V’ is simplified through the use of *Piecewise Linear Functions* (Fourer et al., 2003) by dividing the ‘V’ values into 3 ranges (or *bands*), each with a *linear regression* as the best fit for ‘P’.It is also proved mathematically that an increase in the application operation admittance ‘Y’ will boost the application operation performance ‘P’ for any given application operation load potential ‘V’. The statistical power (Wohlin et al., 2012) of the experiments is derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data.

In the next few chapters, we will use this derived *PALP* model to calculate and extract the statistical functions in which an application operation’s admittance vary with the change of Delay Point latencies. In this chapter, we verified the statistical power of the experiments conducted and proved that the experiments have the ability to reveal the true underlying patterns of the collated data. An assessment of the threats to the validity of the experimental findings has also been presented in this chapter.

6 Single-Type Delay Point Changes: Method Evaluation

We divide the evaluation of our approach to analysing the impact of changes to the Delay Points of a particular type across the application components catering to an operation on its Admittance (‘Y’) into an intrinsic and an extrinsic one. The goal of the former is to demonstrate that our approach addresses the aspects a), b), c), d) and e) of the main research question as explained in Section 1.3. That is to demonstrate that the approach can be used by the software developers without the need of additional quality assurance experts. The software developers will neither be required to learn new, non-standard modelling formalisms and notations nor to specify the software system using process algebras or complex SPE techniques. The approach will not need repetitive performance testing or resource intensive application profilers to assess the impact of the changes to an evolving application. It will yield precise and accurate results. The statistical power (Wohlin et al., 2012) of the forecasts shall be as high as possible. The method will be applied during the implementation phase or later to ensure more precise performance measures.

The extrinsic part of the evaluation is concerned with quantifying the effects of the changes to the Delay Points of a particular type across the application components catering to an operation and establishing the fact that under a given workload condition and hosting environment, the operation’s Admittance (‘Y’) can be expressed as a statistical *function* of the total cumulative latency of the Delay Points of that type. This will establish the second working hypothesis of this thesis as explained in Section 3.4.1.

For the evaluation methodology, as with the previous evaluation of the PALP model in Chapter 5, we follow the guidelines on experimentation in software engineering as they are presented in (Wohlin et al., 2012). The same approach to controlled experiments is adopted. Details of the controlled experiments are provided in Section 6.1 below. As advocated by (Wohlin et al., 2012), the statistical power of the experiments is determined to demonstrate the ability of the experiments to reveal the true pattern in the collated data. The run-time monitoring approaches proposed by (Westermann et al., 2010) used systematic measurements to build mathematical models. The models served as interpolation of the measurements. The objective of the approach was to abstract from system internals by applying a combination of systematic goal-oriented measurements, statistical model inference and model integration. This approach is adopted to evaluate the second hypothesis. It does not consider the internal structure of the underlying system and takes a *black-box* view to it. However, it adopts a *white-box* approach for the application operation and focuses on the observable data.

To compute the statistical power of the experiments to demonstrate the ability of the experiments to reveal the true pattern in the collated data, a second ***null hypothesis*** (**$H2_0$**) is formulated corresponding to the second hypothesis. The null hypothesis states that under a given workload condition and hosting environment, the operation Admittance (‘Y’) has no correlation to the cumulative latency of the Delay Points of a particular type across the supporting application components catering to the operation and thus cannot be expressed as a statistical *function* of their total cumulative latency.

In this chapter we describe the experiments and the process by which we evaluate the second working hypothesis of this thesis. Patterns of variation of an application operation’s admittance due to changes to Delay Points of one particular type across the application components are extracted through controlled experiments. The integrity of the patterns is validated subsequently as well. The statistical power of the experiments is derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data. Keeping the other types of Delay Points unchanged, the processing intensities of the Delay Points of one particular type are varied and the impact on admittance and in turn the performance is recorded. The same process is repeated iteratively for all the different types of Delay Points.

To compare the results and analyze the generic nature of the proposed methodology, the same application framework is implemented on two different hosting environments and the same controlled experiments are performed on the respective environments. The main objective was to observe the impact of augmented hardware on the various measured data under similar workload conditions.

6.1 Database Interaction Delay Points

In the prototypical application framework, all the various backend components have dependency on the LatencyActivities object to perform their respective processing activities. In our example TCG application, each of the TA and the FA components performs some database interactions with an Oracle 10g database based on their respective business logic. To simulate this scenario in the experiment application framework, a public method named *someDBAccess* is created in the LatencyActivities class to perform some database interactions with an Oracle 10g database. It performs some Write operations followed by some Deletes and then some Read operations as detailed later in this section.

The method takes three input parameters namely *dbAccessLoop*, *component* and *toPrint*. A brief description of the parameters is provided below:

Parameter Name	Parameter Type	Purpose
dbAccessLoop	Java int	This counter is used to vary the intensity of the database interactions
Component	Java String	This indicator is used to specify which backend component is called
toPrint	Java String	This indicator is passed from the client side to switch the printing of the response times on and off.

The LatencyActivities.*someDBAccess*(...) method performs the following database interaction activities:

1. Records the start time at the beginning of the operation
2. Loads the jdbc OracleDriver class.
3. Gets a connection from the oracle DriverManager
4. Creates a Java.sql.Statement from the obtained Connection object.
5. Deletes all the rows in a particular table.
6. Inserts some rows.
7. Reads the data back into a Resultset.
8. Loops through the Resultset to read the fetched records
9. Performs these activities for a number of times as specified in the dbAccessLoop.
10. Records the finish time at the end of the operation.
11. From the start and finish times, calculate the total time taken for the database interaction activities.

Given the definition of Stress Point of an application operation in Section 1.4, it depends on the business stakeholders requirements and is typically constant for a given application configuration and environment setup. Also, with reference to the discussion presented in Section 1.4, we would be interested to observe the pattern of variation in "*acceptable*" performance. Hence, for all our controlled experiments we have maintained a positive Load Potential 'V' and assumed a Stress Point of 40 requests/sec. 'V' is calculated from the difference of this assumed Stress Point and the generated workload. The Database Interactions Delay Point activity intensities of the components were incrementally varied keeping the rest of the configuration constant ('V' kept positive). For every configuration, using the *PALP* model Equation (5.2), 'Y' is calculated from the measured values of 'P' and 'V' applying the following formula:

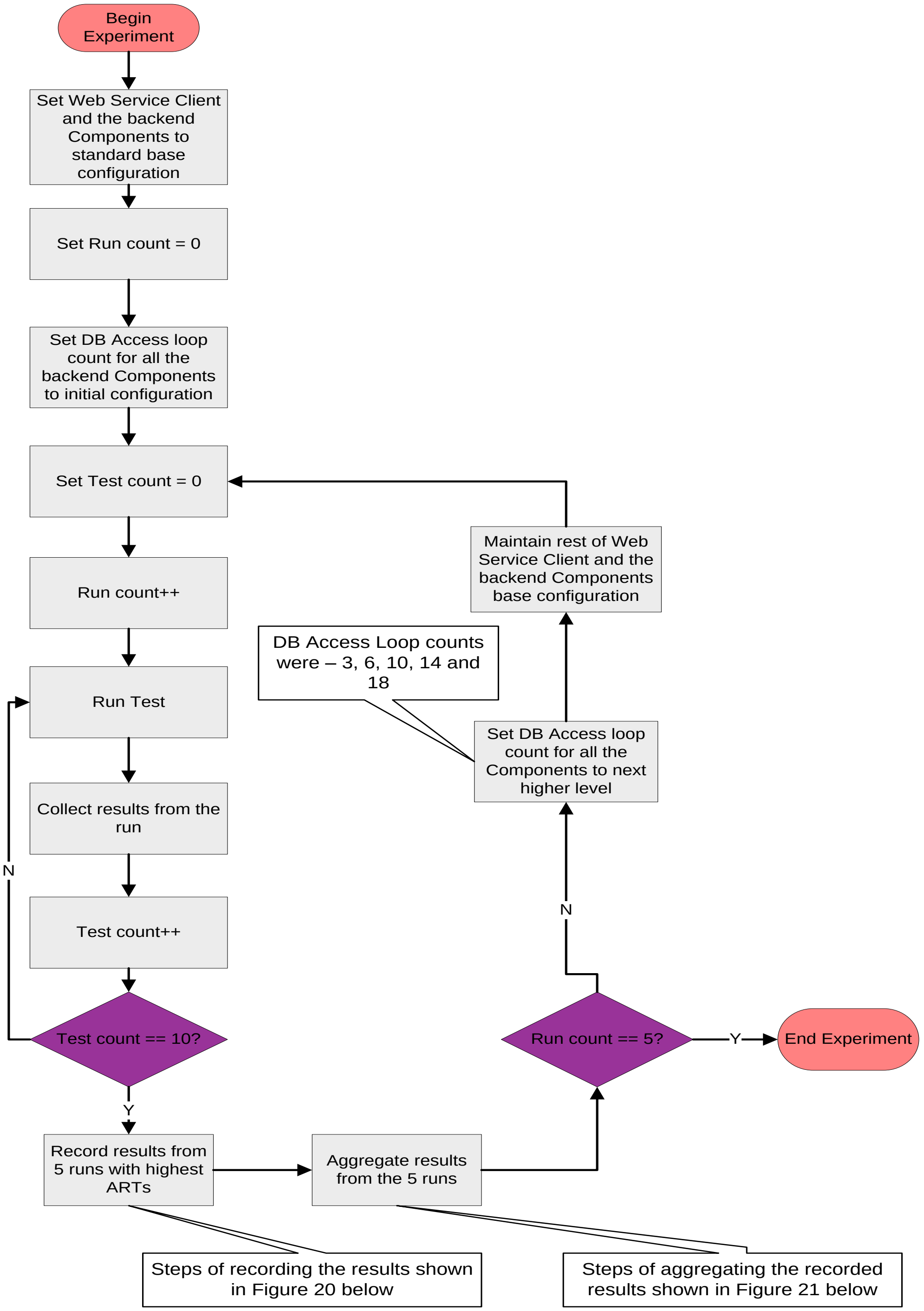
$$Y = (P - c) / V \qquad (6.1)$$

where the values of the y-intercept '*c*' and load potential 'V' are specific to the range of 'V'.

As the values of 'V' for all the configurations are less than the upper limit of Band 1 in Figure 17 (i.e. 32.6 requests/second), the value of the constant '*c*' is taken as ***-1.1953***, which is the value of '*c*' for Band 1 of 'V' as shown in Figure 17.

Data for measured *actual* 'P', *computed indicative* values of 'Y', the measured *actual* average Database Interactions Delay Point latency (I_{DB}) and the *computed* Database Interactions Admittance Factor (YF_{DB} = Y/I_{DB}) are recorded. Function graphs of 'Y' versus 'I_{DB}', 'YF_{DB}' versus 'I_{DB}' are plotted with the measured and model based computed data from the experiments. The graphs showed *distinct* trends in variation, which were consistent but non-linear. Accepting approximation error, the best-fit LSF for *Linear*, *Exponential, Polynomial* and *Power* regression types are verified.

Figure 19 below illustrates the high level flow of activities during the experiments and data collection for variation of intensities of the Database Interaction Delay Points:


Begin Experiment
Set Web Service Client and the backend Components to standard base configuration
Set Run count = 0
Set DB Access loop count for all the backend Components to initial configuration
Set Test count = 0
Run count++
Run Test
Collect results from the run
Test count++
Test count == 10?
N
Y
Record results from 5 runs with highest ARTs
Aggregate results from the 5 runs
Run count == 5?
Y
End Experiment
N
Set DB Access loop count for all the Components to next higher level
Maintain rest of Web Service Client and the backend Components base configuration

DB Access Loop counts were – 3, 6, 10, 14 and 18

Steps of recording the results shown in Figure 20 below

Steps of aggregating the recorded results shown in Figure 21 below

Figure 19: Data Collection for Database Interactions

Figure 20 below shows the steps for recording the data during a particular test run for a given database interaction configuration. Data from 5 such test runs (out of 10) yielding the highest ARTs are recorded.

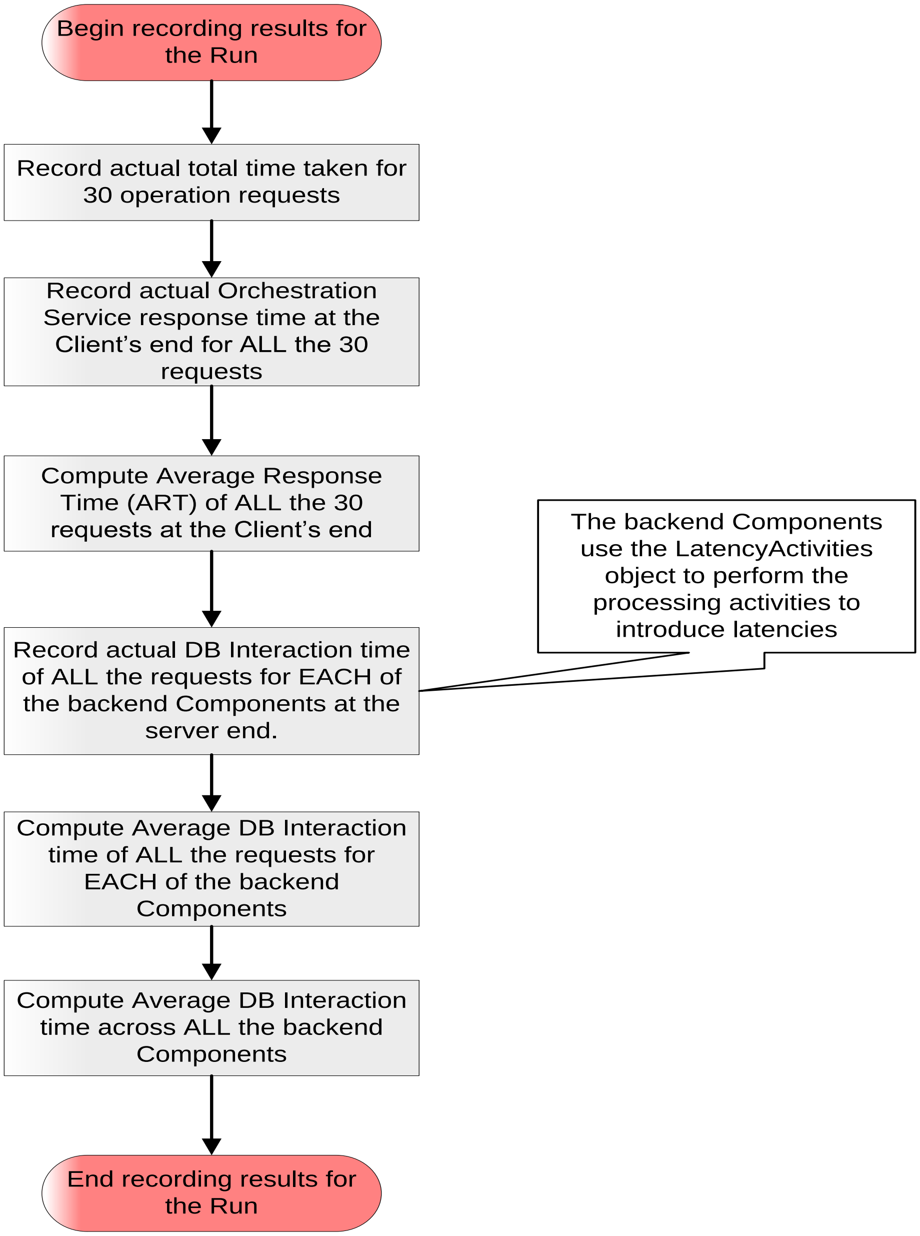

Begin recording results for the Run
Record actual total time taken for 30 operation requests
Record actual Orchestration Service response time at the Client's end for ALL the 30 requests
Compute Average Response Time (ART) of ALL the 30 requests at the Client's end
Record actual DB Interaction time of ALL the requests for EACH of the backend Components at the server end.
The backend Components use the LatencyActivities object to perform the processing activities to introduce latencies
Compute Average DB Interaction time of ALL the requests for EACH of the backend Components
Compute Average DB Interaction time across ALL the backend Components
End recording results for the Run

Figure 20: Steps for recording results for each run

Figure 21 below illustrates the steps of aggregating the database interaction results obtained from the 5 test runs with the highest ARTs.

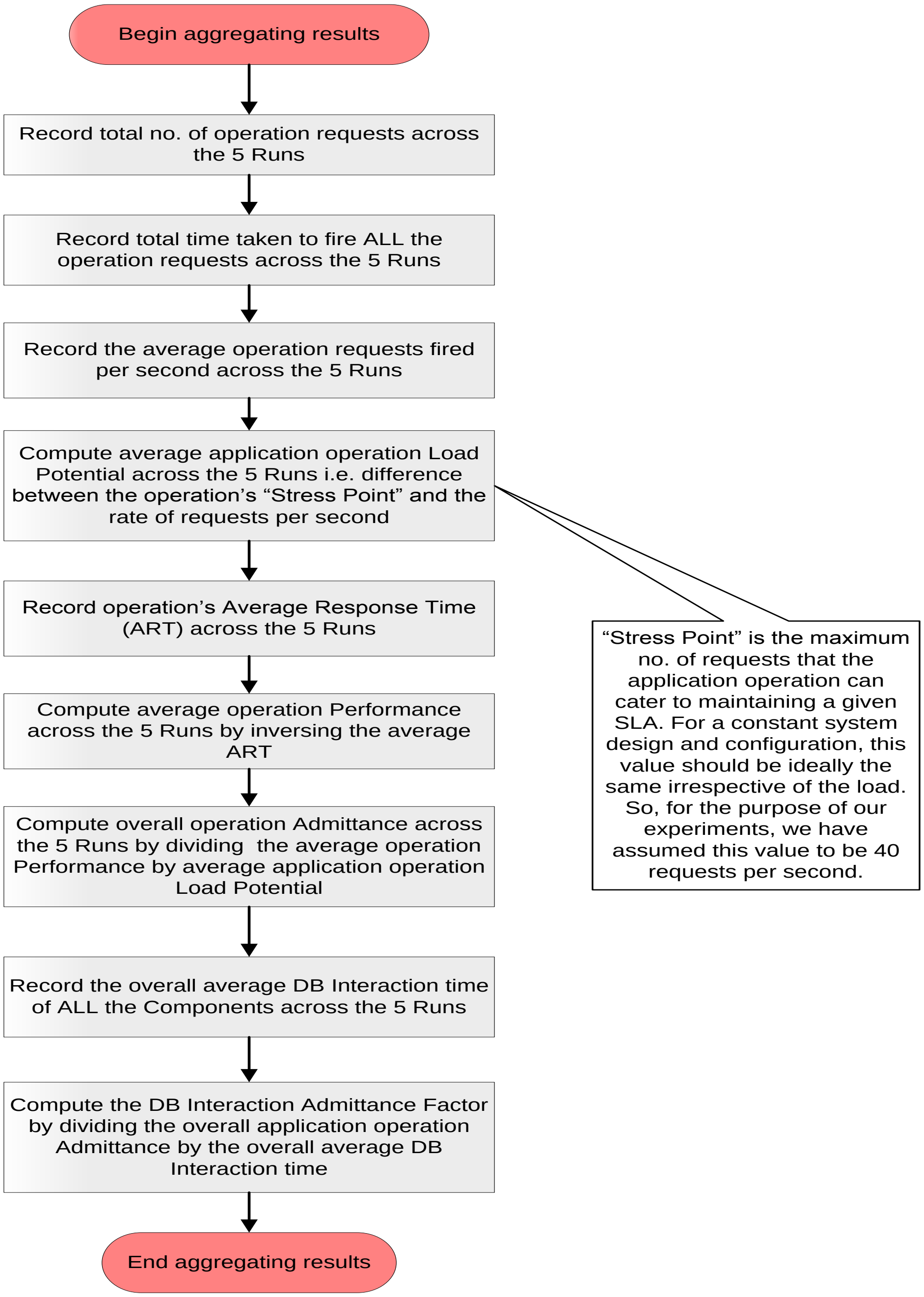

Begin aggregating results
Record total no. of operation requests across the 5 Runs
Record total time taken to fire ALL the operation requests across the 5 Runs
Record the average operation requests fired per second across the 5 Runs
Compute average application operation Load Potential across the 5 Runs i.e. difference between the operation's "Stress Point" and the rate of requests per second
Record operation's Average Response Time (ART) across the 5 Runs
Compute average operation Performance across the 5 Runs by inversing the average ART
Compute overall operation Admittance across the 5 Runs by dividing the average operation Performance by average application operation Load Potential
Record the overall average DB Interaction time of ALL the Components across the 5 Runs
Compute the DB Interaction Admittance Factor by dividing the overall application operation Admittance by the overall average DB Interaction time
End aggregating results
"Stress Point" is the maximum no. of requests that the application operation can cater to maintaining a given SLA. For a constant system design and configuration, this value should be ideally the same irrespective of the load. So, for the purpose of our experiments, we have assumed this value to be 40 requests per second.

Figure 21: Steps for aggregating results across all the runs

Table 2 and Figure 22 present the data obtained from the experiment runs on HE1 for the function graphs of ‘Y’ versus ‘I_{DB}’.

Pattern integrity validation, which is performed subsequently to confirm the correctness of the extracted functions, is highlighted in the table and indicated on the graphs.

OVERALL RESULTS COMPARISON										
DB Access Loop	Total Operation Requests	Total time for requests in milliseconds	Operation Requests per second	Operation Stress Point (Requests per second)	Operation Load Potential (V) in Requests per second	Average Operation Response Time (ART) in seconds	Operation Performance (P=1/ART) in seconds^{-1}	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual DB Access latency (I_{DB}) in seconds	Admittance Factor (YF_{DB} = Y/I_{DB})
3	150	18953	7.914314357	40	32.08568564	6.875533333	0.145443263	0.041786337	0.804137333	0.051964179
6	150	19109	7.849704328	40	32.15029567	7.514593333	0.1330744	0.041317642	0.891390667	0.04635189
8	**150**	**19312**	**7.767191384**	**40**	**32.23280862**	**8.367553333**	**0.119509247**	**0.040791023**	**1.02896**	**0.039642963**
10	150	19375	7.741935484	40	32.25806452	8.927793333	0.112009761	0.040526603	1.106497333	0.036626028
14	150	19312	7.767191384	40	32.23280862	10.55689333	0.094724837	0.040022105	1.194706667	0.033499524
18	150	19453	7.710892921	40	32.28910708	11.39134	0.087785985	0.039737425	1.337642667	0.029707056

Table 2: Experimental Data for varying Database Interactions on HE1

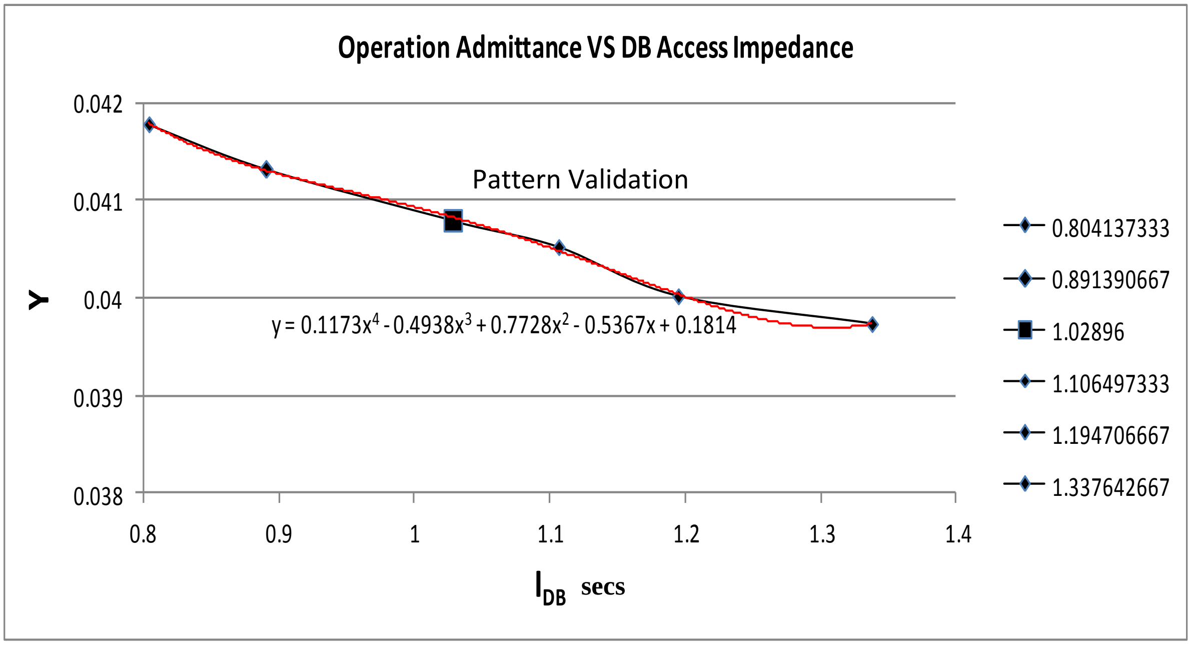


Figure 22: Function Graph of Y vs I_{DB} for varying Database Interactions on HE1

For 'Y' versus 'I_{DB}', pattern line with *Polynomial* regression of 4th order was the best fit:

$$Y = 0.1173I_{DB}^{4} - 0.4938I_{DB}^{3} + 0.7728I_{DB}^{2} - 0.5367I_{DB} + 0.1814 \qquad (6.2)$$

As expected, a gradual decrease in the application operation Admittance 'Y' is observed with increase of the Database Interactions Delay Points impedance. Here again, the observed data sets affirm the second hypothesis of this thesis that under a given workload condition and hosting environment, 'Y' can be expressed as a statistical *function* of the total cumulative latency of the Database Interaction Delay Points 'I_{DB}' across all the supporting application components catering to the operation. In our example TCG application, this is analogous to expressing 'Y' of the getConfidence(…) operation as a statistical function of the total cumulative latency of all the database interactions across TA and FA components.

For the given range of '*V*', the function of 'I_{DB}' as shown in Equation (6.1) above can be used to predict a possible value of operation Admittance say 'Y_p' corresponding to any future value of 'I_{DB}'.

This 'Y_p' can then be fed into the *PALP* model ($P = YV + c$) for that particular 'V band' to have an estimate of the possible performance 'P'. Hence the predicted performance P_p can be denoted by:

$$P_p = Y_pV + c \qquad (6.3)$$

For the Database Interaction Delay Points, experiments under identical workload conditions and configuration setup as Hosting Environment 1 (HE1) are performed on Hosting Environment 2 (HE2) as described in Sections 4.6.1 and 4.6.2. The objective of this was to verify whether hosting environment of higher specification has any positive impact on the variation of 'Y' for changes to 'I_{DB}'. Table 3 and Figure 23 present the data obtained from the experiment runs on HE2 for the function graph of 'Y' versus 'I_{DB}'. As with the data measured on HE1, the observed data sets from HE2 also affirm the second hypothesis of this thesis. Pattern integrity validation, which is performed subsequently to confirm the correctness of the extracted functions, is highlighted in the table and indicated on the graph.

OVERALL RESULTS COMPARISON										
DB Access Loop	Total Operation Requests	Total time for requests in milliseconds	Operation Requests per second	Operation Stress Point (Requests per second)	Operation Load Potential (V) in Requests per second	Average Operation Response Time (ART) in seconds	Operation Performance (P=1/ART) in seconds^{-1}	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual DB Access latency (I_{DB}) in seconds	Admittance Factor (YF_{DB} = Y/I_{DB})
3	150	18953	7.914314357	40	32.08568564	6.785533333	0.147372351	0.04184646	0.803137333	0.052103741
6	150	19109	7.849704328	40	32.15029567	7.314593333	0.136713	0.041430816	0.871390667	0.047545628
8	**150**	**19312**	**7.767191384**	**40**	**32.23280862**	**8.167553333**	**0.122435687**	**0.040881814**	**1.00496**	**0.040680041**
10	150	19375	7.741935484	40	32.25806452	8.427793333	0.118655022	0.040732606	1.039497333	0.039184906
14	150	19312	7.767191384	40	32.23280862	9.056893333	0.110413137	0.040508823	1.174706667	0.034484203
18	150	19453	7.710892921	40	32.28910708	9.24134	0.108209416	0.040369943	1.317642667	0.030638005

Table 3: Experimental Data for varying Database Interactions in HE2

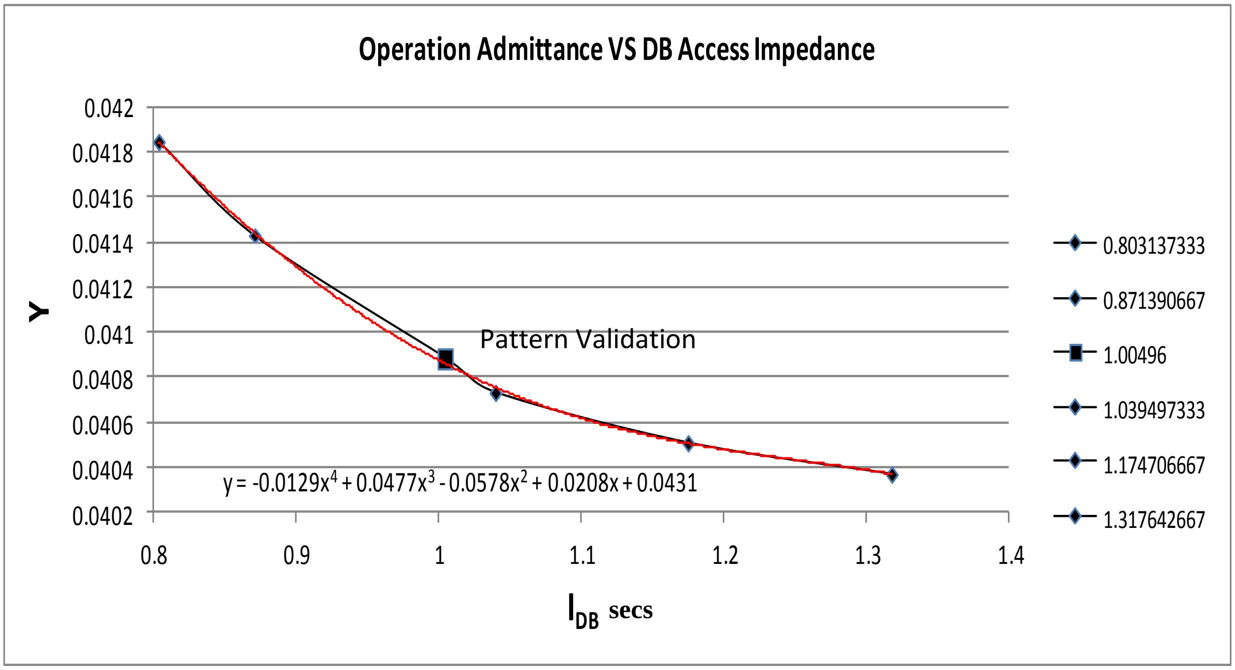


Figure 23: Function Graph of Y vs I_{DB} for varying DB Interactions on HE2

Comparing the graph functions of application operation Admittance ‘Y’ versus the Database Interaction Delay Point impedance/latency ‘I_{DB}’ in Figures 22 and 23, it is observed that the decrease in the ‘Y’ with the increase of the latency/impedance of the Database Interaction Delay Points ‘I_{DB}’ is sharper on HE1 than on HE2. Less sharp decrease in ‘Y’ implies improved performance on HE2 than HE1 for the same increase in Delay Point latencies. Hence, this observation also corroborates the fact that augmented hosting provision under same workload conditions has a positive impact on the performance of the application operation.

6.1.1 Verifying Precision of the Derived Patterns

Tests are performed to verify the precision of the derived patterns (statistical functions). The objective of the verification tests was to examine whether the results obtained from any intermediate database interaction configuration conformed to the previously derived functions. This is achieved by assessing the delta between the measured data during the tests and the data obtained from using the derived statistical functions.

During evaluation of the second hypothesis, the database interaction configurations were set to 3, 6, 10, 14 and 18. For the subsequent integrity validation of the derived functions, the database interaction configuration is set to 8, which is intermediate to 6 and 10. For this intermediate configuration, 10 test runs are executed with each test run comprising 30 transaction requests of the same class of workload and representative of real-life scenarios. Data from 5 of the test runs with the highest ARTs are recorded. Mean of all the relevant data are calculated across the 5 test runs.

Comparing the actual data measured during the tests to that of the forecast obtained from using the derived functions it transpired that for a DB Interactions cumulative latency of 1.02896 seconds, the operation Admittance (‘Y’) calculated from actual data observed is 0.002829773 while the value predicted by the previously derived function was 0.002933533. This yielded an error of 2.6% approximately, which is very low considering this is representative of 150 sets of request-responses. As with the Data Processing Delay Points, further calibration of the derived function can be achieved by repeating this process and recursively adjusting the Y vs I_{DB} function for every new set of ‘Y’ calculated from actual data observed and that predicted by the Y vs I_{DB} function. Through calibration, we shall be able to minimise the error and achieve much higher precision for any future configuration of Database Interaction.

6.1.2 Statistical Power of the Experiment

The data in Tables 2 and 3, the corresponding statistical function in Figures 22 and 23 and the subsequent data obtained from the experiments to validate the integrity of the extracted patterns prove that under a given workload condition and hosting environment, the operation Admittance (‘Y’) can be expressed as a statistical *function* of the total cumulative latency of all the DB Interactions Delay Points (‘I_{DB}’) across the supporting application components catering to the operation. This refutes the *second null hypothesis* (**$H2_0$**), which can be rejected.

A *Type-II-error* would have occurred had the experiments not indicated the true pattern of relationship between the operation Admittance and the total latency of the DB Interactions Delay Points. The experimental data demonstrated a definite relationship pattern. Further experiments to validate the integrity of the derived patterns highlighted the following:

For a DB Interactions cumulative latency of 1.02896 seconds, the actual operation Admittance observed is 0.002829773 while the value predicted by the extracted pattern was 0.002933533, yielding an error of 2.6% approximately. This is very low considering the fact that the test data is representative of 150 round trip requests-response transactions. In view of the accuracy of the extracted patterns, we consider the probability *P*(Type-II-error) approaches zero within the bounds of the conducted experiments. Given the expression of the statistical power of an experiment in (5.3), if *P*(Type-II-error) approaches 0, the statistical power approaches 1, which is very high. Hence, the collated data from the experiments are proven ideal to reveal the true pattern (Wohlin et al., 2012).

Figure 24 shows the steps of the experiment to validate the integrity of the functions:

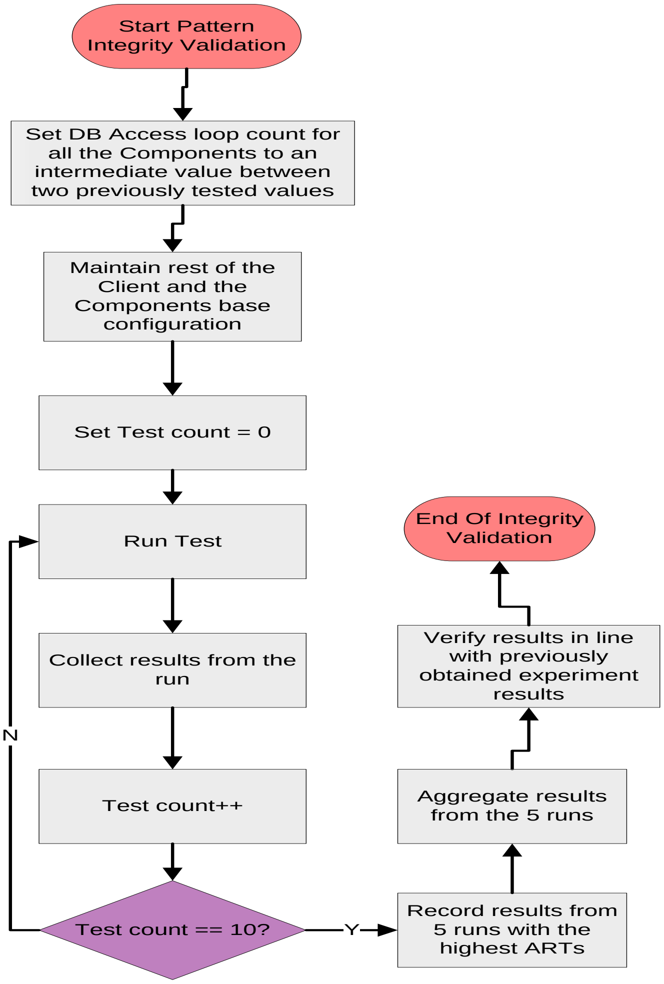

Start Pattern Integrity Validation
Set DB Access loop count for all the Components to an intermediate value between two previously tested values
Maintain rest of the Client and the Components base configuration
Set Test count = 0
Run Test
Collect results from the run
Test count++
Test count == 10?
N
Y
Record results from 5 runs with the highest ARTs
Aggregate results from the 5 runs
Verify results in line with previously obtained experiment results
End Of Integrity Validation

Figure 24: Steps of verifying the derived functions

6.2 File I/O Delay Points

In the same way as the Database Interactions Delay Points, the File I/O Delay Point processing intensities of the components are incrementally varied keeping the rest of the configuration constant ('V' kept positive).

In our example TCG application, each of the TA and the FA components performs various types of file I/O operations on the disk based on their respective business logic. To simulate this scenario, a public method named *someFileIO* is created in the LatencyActivities class to perform some file input/output operations with the File System in the application framework. The method takes four input parameters namely *fileName*, *fileIOLoop, component* and *toPrint*. A brief description of the parameters is provided below:

Parameter Name	Parameter Type	Purpose
filename	Java String	Through this parameter, the file name with which the File I/O activities will happen is sent to the operation.
fileIOLoop	Java int	With this counter the intensity of the File I/O interactions is varied.
component	Java String	This indicator specifies the backend component that is being called.
toPrint	Java String	This indicator is passed from the client side to switch the printing of the response times on and off.

The LatencyActivities.*someFileIO*(...) method performs the following file input / output activities to a file as specified by the *fileName* parameter:

1. Records the start time at the beginning of the operation

2. Creates a new file with name as specified by the fileName parameter or uses the existing one if the file exists.
3. Based on an *appendMode*, appends new textual contents to the existing contents of the file or overwrites the existing contents with the new contents.
4. If the write operation is successful, prepare to read from the file.
5. Creates a read channel by using FileInputStream, InputStreamReader and BufferedReader
6. Reads the contents of the File.
7. Performs these activities for a number of times as specified in the fileIOLoop.
8. Records the finish time at the end of the operation.
9. From the start and finish times, calculate the total time taken for the File Input / Output activities.

For the same reasons as explained in Section 6.1, we have maintained a positive Load Potential 'V' and assumed a Stress Point of 40 requests/sec in all our controlled experiments. 'V' is calculated from the difference of this assumed Stress Point and the generated workload. The File I/O Delay Point activity intensities of the components were incrementally varied keeping the rest of the configuration constant ('V' kept positive). For every configuration, using the *PALP* model Equation (5.2), 'Y' is calculated from the measured values of 'P' and 'V' applying Equation (6.1) : $Y = (P - c) / V$, where the values of the y-intercept '*c*' and load potential 'V' are specific to the range of 'V'.

As with the Database Interaction Delay Points, the values of 'V' for all the configurations are less than the upper limit of Band 1 in Figure 17 (i.e. 32.6 requests/second), the value of the constant '*c*' is taken as ***-1.1953***, which is the value of '*c*' for Band 1 of 'V' as shown in Figure 17.

With the measured and model based computed data from the experiments, the function graphs of 'Y' versus 'I_{FIO}' and 'YF_{FIO}' versus 'I_{FIO}' are plotted. Data for measured *actual* 'P', *computed indicative* values of 'Y', the measured *actual* average File I/O Delay Point impedance (I_{FIO}) and the *computed* File I/O Admittance Factor ($YF_{FIO} = Y/I_{FIO}$) are recorded. The graphs showed *distinct* trends in variation, which were consistent but non-linear. Accepting approximation error, for simplicity, the best-fit LSF for *Linear*, *Exponential*, *Polynomial* and *Power* regression types are verified.

Table 4 and Figure 25 present the data obtained from the experiment runs on HE1 for the function graphs of 'Y' versus 'I_{FIO}'. Pattern integrity validation, which is performed subsequently to confirm the correctness of the extracted functions, is highlighted in the table and indicated on the graphs.

OVERALL RESULTS COMPARISON										
File I/O Loop	Total Operation Requests	Total time for requests in milliseconds	Operation Requests per second	Operation Stress Point (Requests per second)	Operation Load Potential (V) in Requests per second	Average Operation Response Time (ART) in seconds	Operation Performance (P=1/ART) in seconds^{-1}	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual File I/O latency (I_{FIO}) in seconds	Admittance Factor (Y/I_{FIO})
1	150	19187	7.817793298	40	32.1822067	0.141966667	7.043907004	0.256017466	0.005017333	51.02660433
3	**150**	**21401**	**7.00901827**	**40**	**32.99098173**	**0.65048**	**1.537326282**	**0.082829493**	**0.070424**	**1.176154333**
5	150	21314	7.03762785	40	32.96237215	1.5107	0.661944794	0.056344391	0.179541333	0.313824067
10	150	21171	7.085163667	40	32.91483633	3.867106667	0.258591264	0.044171305	0.627218667	0.070424091
15	150	21104	7.107657316	40	32.89234268	5.60749333	0.178332802	0.041761477	0.932293333	0.044794353
20	150	20640	7.26744186	40	32.73255814	7.2279	0.138352772	0.040743921	1.22545333	0.03324804
25	150	20577	7.289692375	40	32.71030763	9.692086667	0.103176956	0.039696262	1.657397333	0.023950963

Table 4: Experimental Data for varying File I/O on HE1

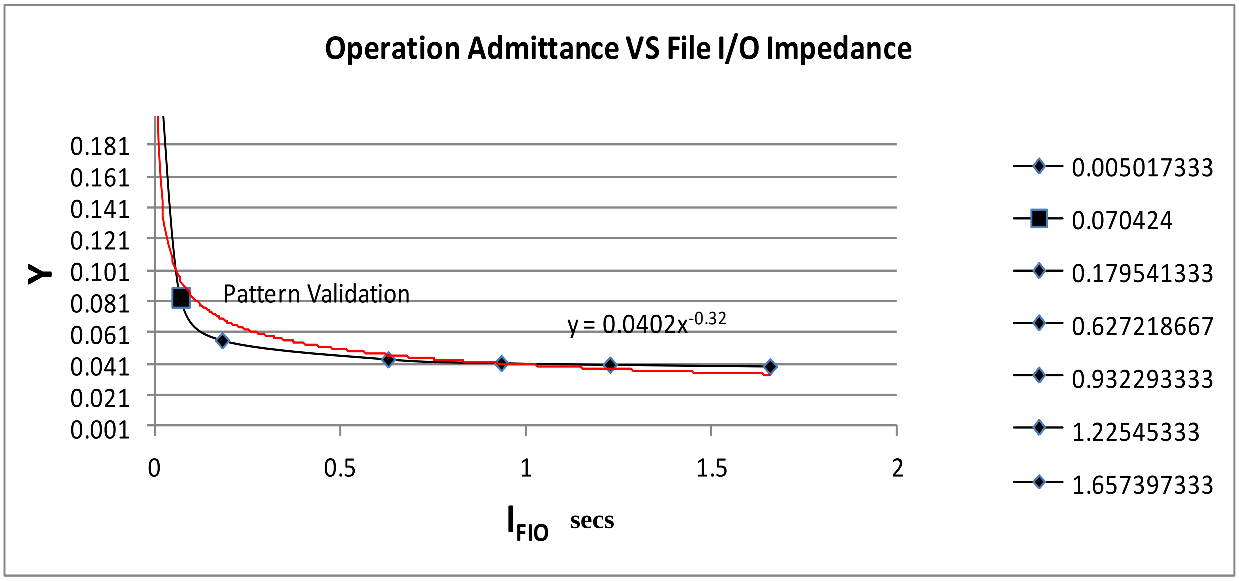


Figure 25: Function Graph of Y vs I_{FIO} for varying File I/O on HE1

In Figure 25, for ‘Y’ versus ‘I_{FIO}’, trend line with *Power* regression is the best fit:

$$Y = 0.0402\, I_{FIO}^{-0.32} \qquad (6.4)$$

Initially, there is a steep decrease of the application operation Admittance for an increase of the File I/O Delay Points impedance. However, it later reached saturation and flattened out with further increase of the Delay Points impedance. The observed data sets affirm the second hypothesis of this thesis as ‘Y’ is presented as a function of ‘I_{FIO}’. In our example TCG application, this is analogous to expressing ‘Y’ of the getConfidence(…) operation as a statistical function of the total cumulative latency of all the file I/O activities across TA and FA components.

For the given range of ‘V’, the function of ‘I_{FIO}’ as shown in Equation (6.4) above can be used to predict a possible value of operation Admittance say ‘Y_p’ corresponding to any future value of ‘I_{FIO}’. This ‘Y_p’ can then be fed into the *PALP* model ($P = YV + c$) for that particular ‘V band’ to have an estimate of the possible performance ‘P’. The predicted performance P_p can be calculated using the Equation (6.3) : $P_p = Y_pV + c$

As with the other types of Delay Points, for the File I/O Delay Points, experiments under identical workload conditions and configuration setup as HE1 are performed on HE2. Again, as before, the objective of this was to verify whether hosting environment of higher specification has any positive impact on the variation of ‘Y’ for changes to ‘I_{FIO}’. Table 5 and Figure 26 present the data obtained from the experiment runs on HE2 for the function graph of ‘Y’ versus ‘I_{FIO}’. As with HE1, pattern integrity validation, which is performed subsequently to confirm the correctness of the extracted functions, is highlighted in the table and indicated on the graph.

OVERALL RESULTS COMPARISON										
File I/O Loop	Total Operation Requests	Total time for requests in milliseconds	Operation Requests per second	Operation Stress Point (Requests per second)	Operation Load Potential (V) in Requests per second	Average Operation Response Time (ART) in seconds	Operation Performance (P=1/ART) in $seconds^{-1}$	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual File I/O latency (I_{FIO}) in seconds	Admittance Factor (Y/I_{FIO})
1	150	19187	7.817793298	40	32.1822067	0.121966667	8.198961442	0.291908555	0.003057333	95.47816826
3	**150**	**21416**	**7.004109077**	**40**	**32.99589092**	**0.30048**	**3.32800852**	**0.137087025**	**0.050724**	**2.702606752**
5	150	21314	7.03762785	40	32.96237215	0.6015	1.662510391	0.086699173	0.158541333	0.546855327
10	150	21171	7.085163667	40	32.91483633	1.267105662	0.789200167	0.060291965	0.423218667	0.142460553
15	150	21104	7.107657316	40	32.89234268	3.00747333	0.332505027	0.046448653	0.932293333	0.04982193
20	150	20640	7.26744186	40	32.73255814	5.0079	0.199684498	0.042617644	1.02545333	0.041559808
25	150	20577	7.289692375	40	32.71030763	7.473086667	0.133813516	0.040632865	1.057397333	0.038427244

Table 5: Experimental Data for varying File I/O on HE2

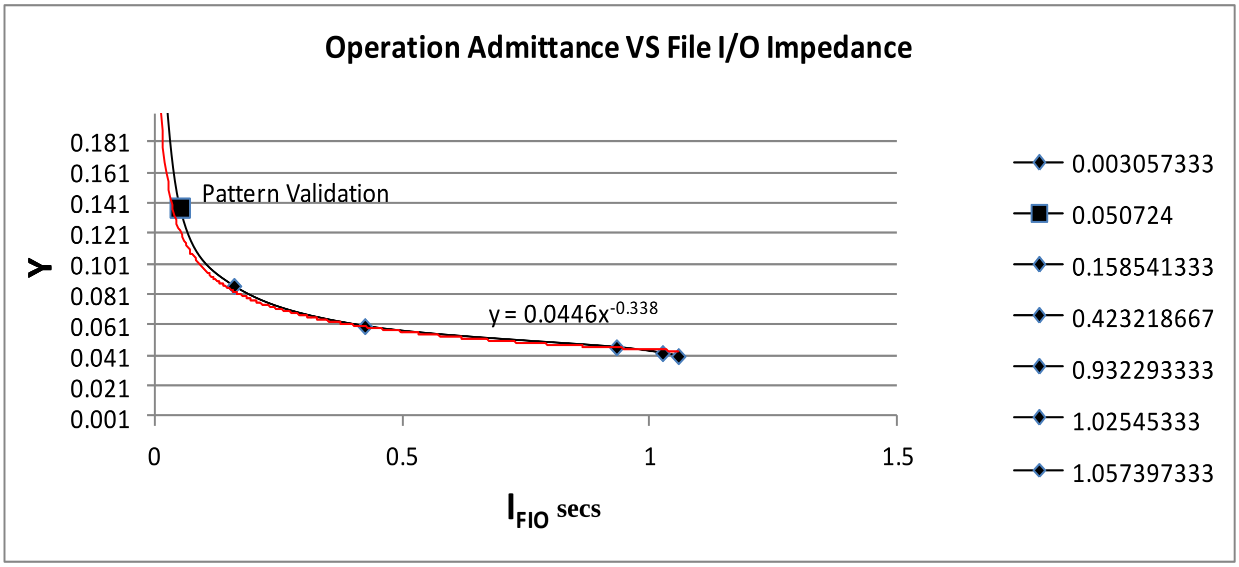


Figure 26: **Function Graph of Y vs I_{FIO} for varying File I/O on HE2**

Comparing the graph functions of application operation Admittance ‘Y’ versus the File I/O Delay Point latency ‘I_{FIO}’ in Figures 25 and 26, in line with the previous two types of Delay Points, it is observed that the decrease in the 'Y' with the increase of the latency/impedance of the File I/O Delay Points 'I_{FIO}' is steeper on HE1 than on HE2. Less steep decrease in 'Y' implies improved performance on HE2 than HE1 for the same increase in Delay Point latencies. Hence, aligning with the previous two types of Delay Points, this observation also supports that augmented hosting provision under same workload conditions has a positive impact on the performance of the application operation.

6.3 Data Processing Delay Points

In our example TCG application, each of the TA and the FA components performs in-memory data processing based on their respective algorithms. To resemble this scenario, a public method called *someDataProcessing* is created in the LatencyActivities class to perform some in-memory data processing. The LatencyActivities.*someDataProcessing*(...) method performs the following in-memory data processing:

1. Records the start time at the beginning of the operation
2. Converts the request data into a Character Array.
3. Applies a pre-defined algorithm to process the data several times.
4. Performs these activities for as many times as specified by the dataProcessLoop parameter.
5. Records the finish time at the end of the operation.
6. From the start and finish times, calculate the total time taken for the data processing activities.

The method takes four input parameters namely *dataProcessLoop*, *component*, *data* and *toPrint*. A brief description of the parameters is provided below:

Parameter Name	**Parameter Type**	**Purpose**
dataProcessLoop	Java int	With this counter we vary the intensity of the data processing
component	Java String	This indicator specifies which component is called
Data	Java String	This is the actual data which gets manipulated during the in-memory data processing.
toPrint	Java String	This indicator is passed from the client side to switch the printing of the response times on and off.

Keeping the rest of the configuration constant and ‘V’ positive, the Data Processing Delay Point activity intensities of all the supporting components are incrementally varied. Experimental data for the following are recorded:

- the measured ‘P’
- the computed indicative values of ‘Y’ based on the *PALP* model for the relevant ‘V’ band
- the average of the measured Data Processing Delay Point latency (I_{DP}) and
- the computed Data Processing Admittance Factor ($YF_{DP} = Y/I_{DP}$)

In the same manner as the previous experiments, a positive Load Potential ‘V’ is maintained and a Stress Point of 40 requests/sec is assumed in all our controlled experiments. ‘V’ is calculated from the difference of this assumed Stress Point and the generated workload. The Data Processing Delay Point activity intensities of the components were incrementally varied keeping the rest of the configuration constant (‘V’ kept positive). For every configuration, using the *PALP* model Equation (5.2), ‘Y’ is calculated from the measured values of ‘P’ and ‘V’ applying Equation (6.1) : $Y = (P – c) / V$, where the values of the y-intercept ‘*c*’ and load potential ‘V’ are specific to the range of ‘V’.

In line with the previous experiments, the value of the constant ‘*c*’ is taken as ***-1.1953***, which is the value of ‘*c*’ for Band 1 of ‘V’ as shown in Figure 17.

With the measured and model based computed data from the experiments, the function graphs of ‘Y’ versus ‘I_{DP}’ and ‘YF_{DP}’ versus ‘I_{DP}’ are plotted. Data for measured *actual* ‘P’, *computed indicative* values of ‘Y’, the measured *actual* average I_{DP} and the *computed* Data Processing Admittance Factor ($YF_{DP} = Y/I_{DP}$) are recorded. The graphs showed *distinct* trends in variation, which were consistent but non-linear. Accepting approximation error, for simplicity, the best-fit LSF for *Linear, Exponential, Polynomial* and *Power* regression types are verified.

Figure 27 below illustrates the high level flow of activities during the experiments and data collection for variation of intensities of the in-memory Data Processing Delay Points:

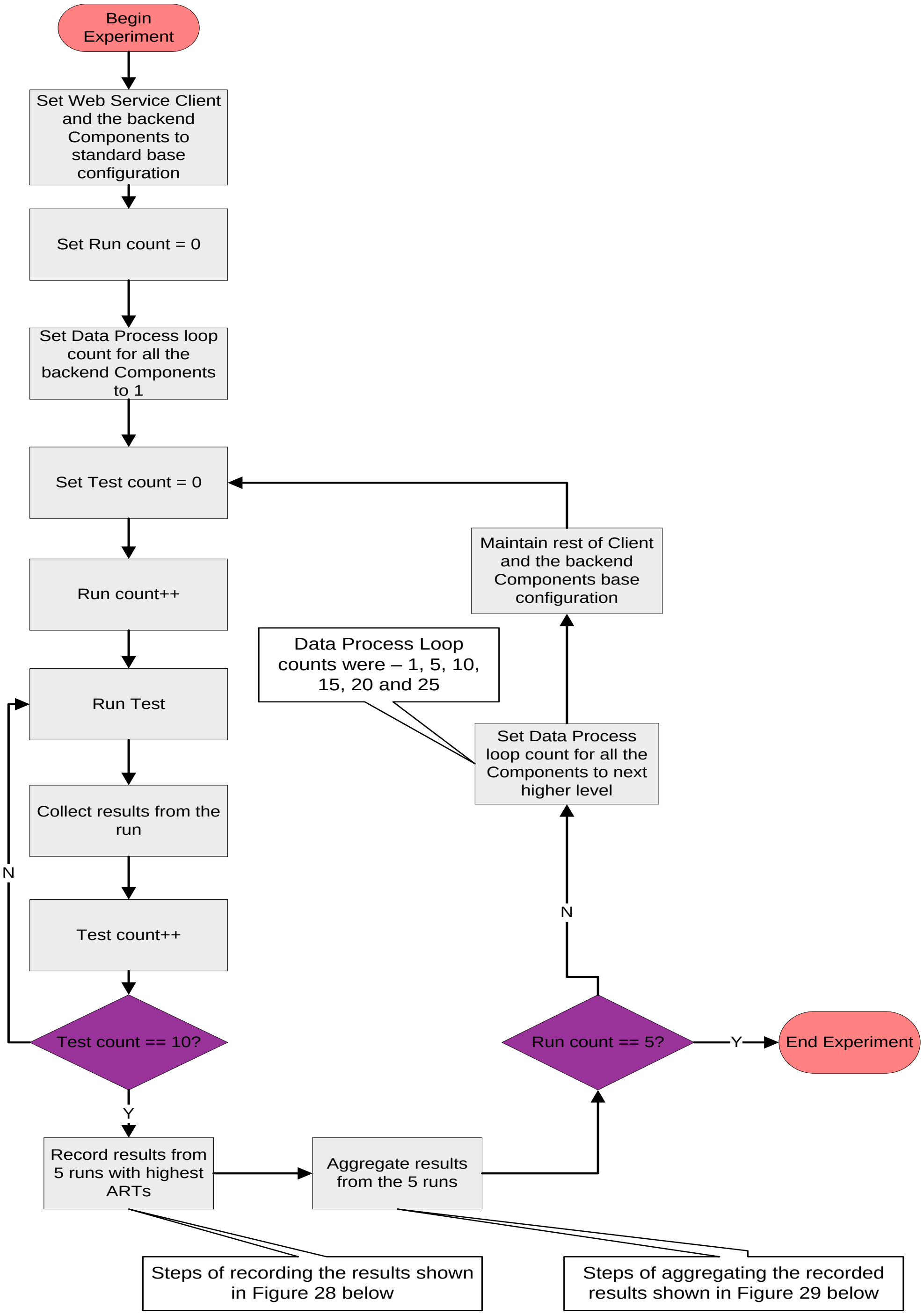

Begin Experiment
Set Web Service Client and the backend Components to standard base configuration
Set Run count = 0
Set Data Process loop count for all the backend Components to 1
Set Test count = 0
Run count++
Run Test
Collect results from the run
Test count++
Test count == 10?
N
Y
Record results from 5 runs with highest ARTs
Aggregate results from the 5 runs
Run count == 5?
Y
End Experiment
N
Set Data Process loop count for all the Components to next higher level
Maintain rest of Client and the backend Components base configuration
Data Process Loop counts were – 1, 5, 10, 15, 20 and 25
Steps of recording the results shown in Figure 28 below
Steps of aggregating the recorded results shown in Figure 29 below

Figure 27: Data Collection for In-Memory Data Processing

Figure 28 below shows the steps for recording the data during a particular test run for a given data processing configuration. Data from 5 such test runs (out of 10) yielding the highest ARTs are recorded.

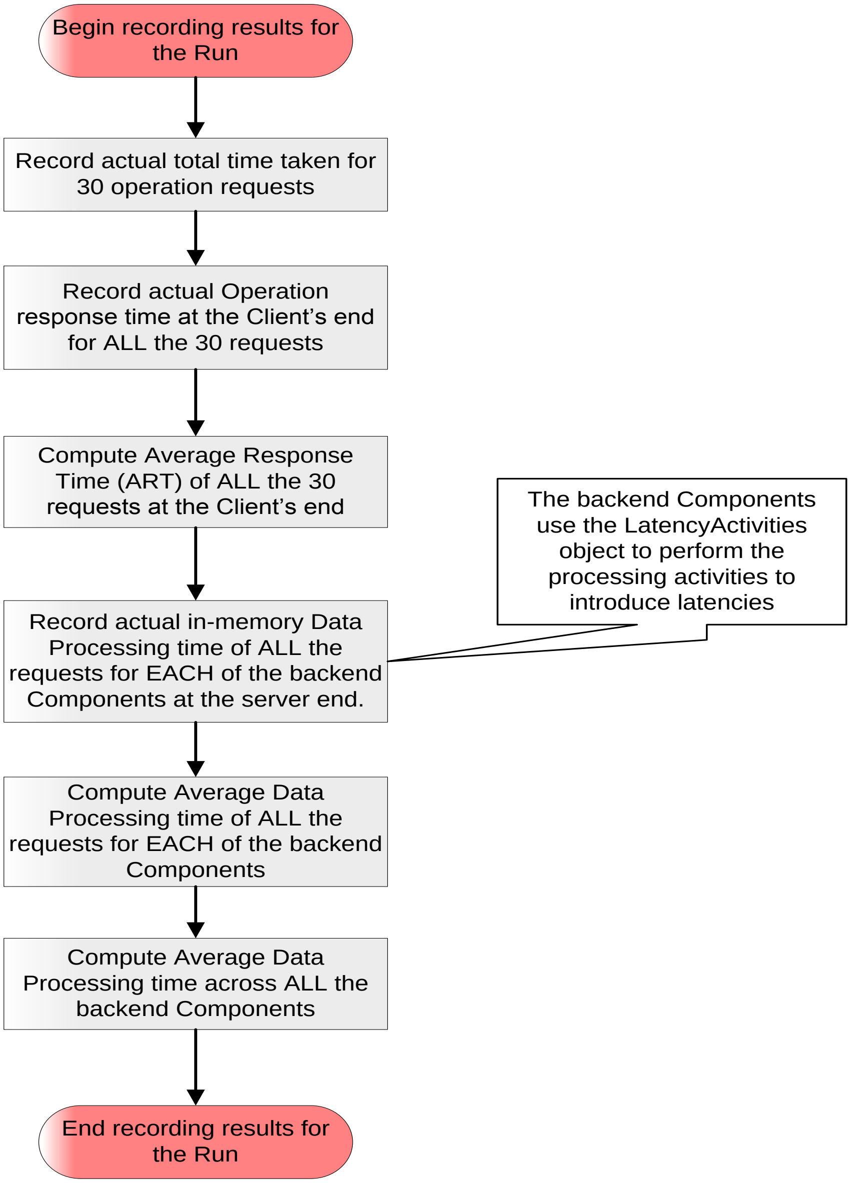

Begin recording results for the Run
Record actual total time taken for 30 operation requests
Record actual Operation response time at the Client's end for ALL the 30 requests
Compute Average Response Time (ART) of ALL the 30 requests at the Client's end
Record actual in-memory Data Processing time of ALL the requests for EACH of the backend Components at the server end.
The backend Components use the LatencyActivities object to perform the processing activities to introduce latencies
Compute Average Data Processing time of ALL the requests for EACH of the backend Components
Compute Average Data Processing time across ALL the backend Components
End recording results for the Run

Figure 28: Steps for recording results for each run

Figure 29 below illustrates the steps of aggregating the in-memory Data Processing results obtained from the 5 test runs with the highest ARTs.

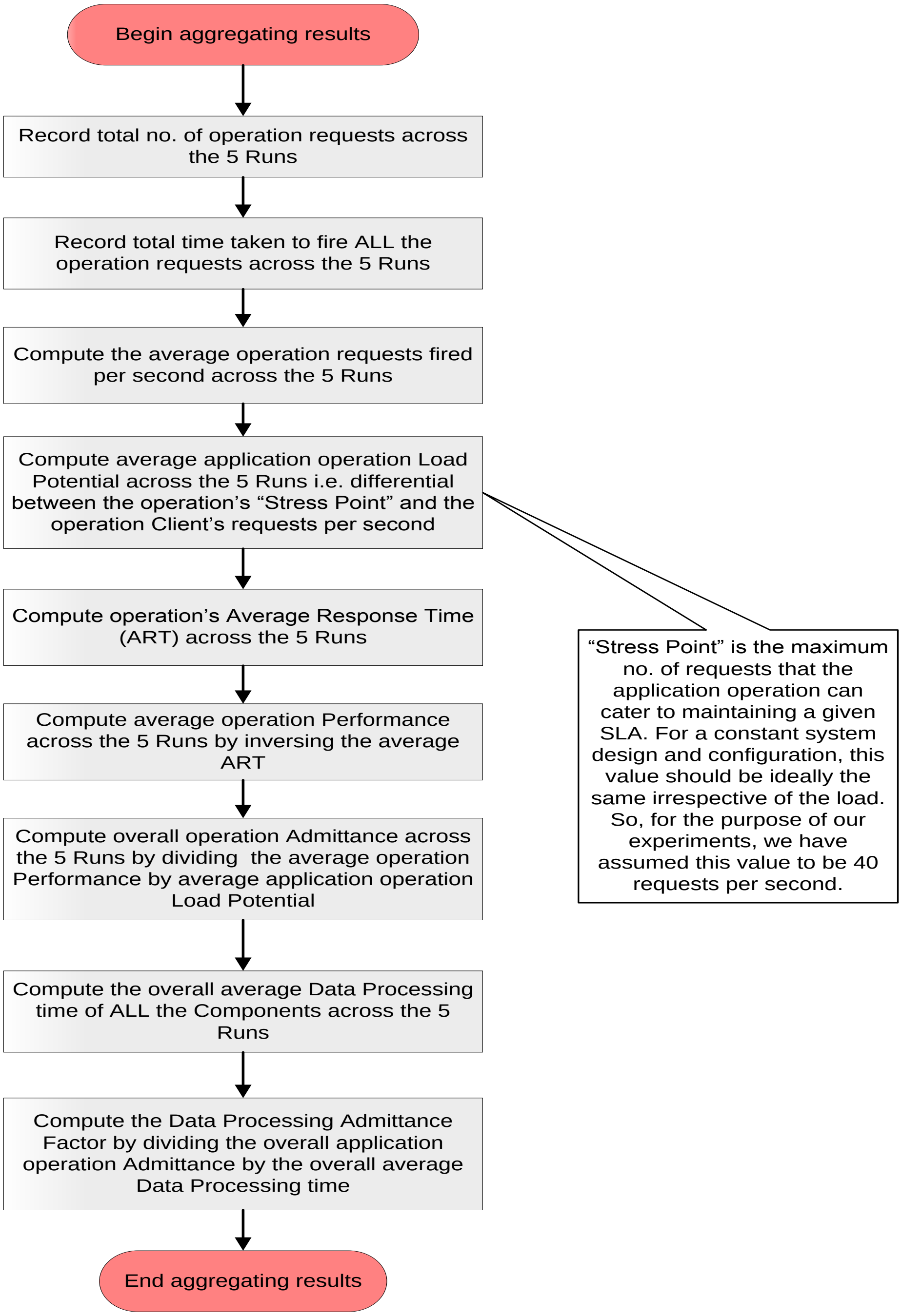

Begin aggregating results
Record total no. of operation requests across the 5 Runs
Record total time taken to fire ALL the operation requests across the 5 Runs
Compute the average operation requests fired per second across the 5 Runs
Compute average application operation Load Potential across the 5 Runs i.e. differential between the operation's "Stress Point" and the operation Client's requests per second
Compute operation's Average Response Time (ART) across the 5 Runs
Compute average operation Performance across the 5 Runs by inversing the average ART
Compute overall operation Admittance across the 5 Runs by dividing the average operation Performance by average application operation Load Potential
Compute the overall average Data Processing time of ALL the Components across the 5 Runs
Compute the Data Processing Admittance Factor by dividing the overall application operation Admittance by the overall average Data Processing time
End aggregating results
"Stress Point" is the maximum no. of requests that the application operation can cater to maintaining a given SLA. For a constant system design and configuration, this value should be ideally the same irrespective of the load. So, for the purpose of our experiments, we have assumed this value to be 40 requests per second.

Figure 29: Steps for aggregating results across all the runs

Table 6, Figure 30 and 31 present the data obtained from the experiment runs on HE1 for the function graphs of ‘Y’ versus ‘I_{DP}’ and ‘YF_{DP}’ versus ‘I_{DP}’. Pattern integrity validation, which is performed subsequently to confirm the correctness of the extracted functions, is highlighted in the table and indicated on the graphs.

OVERALL RESULTS COMPARISON										
Data Process Loop	Total Operation Requests	Total time for requests in milliseconds	Operation Requests per second	Operation Stress Point (Requests per second)	Operation Load Potential (V) in Requests per second	Average Operation Response Time (ART) in seconds	Operation Performance (P=1/ART) in seconds^{-1}	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual Data Processing latency (I_{DP}) in seconds	Admittance Factor (YF_{DP} = Y/I_{DP})
1	150	19672	7.625050834	40	32.37494917	0.255626667	3.911954929	0.157753296	0.000618667	254.9891632
3	**150**	**19790**	**7.579585649**	**40**	**32.42041435**	**0.266953333**	**3.745973084**	0.152412398	**0.000934667**	**163.0660459**
5	150	19863	7.551729346	40	32.44827065	0.28688	3.485778026	0.144262789	0.001504	95.91940764
10	150	19927	7.527475285	40	32.47252472	0.39452	2.534725743	0.114867131	0.003318667	34.61243393
15	150	20624	7.273079907	40	32.72692009	0.492813333	2.029165879	0.098526408	0.005601333	17.58981336
20	150	20731	7.235540977	40	32.76445902	0.591713333	1.69000755	0.088062115	0.007706667	11.42674499
25	150	20893	7.179438089	40	32.82056191	0.74885333	1.335374979	0.077106388	0.009857333	7.822236029

Table 6: Experimental Data for varying Data Processing activities in HE1

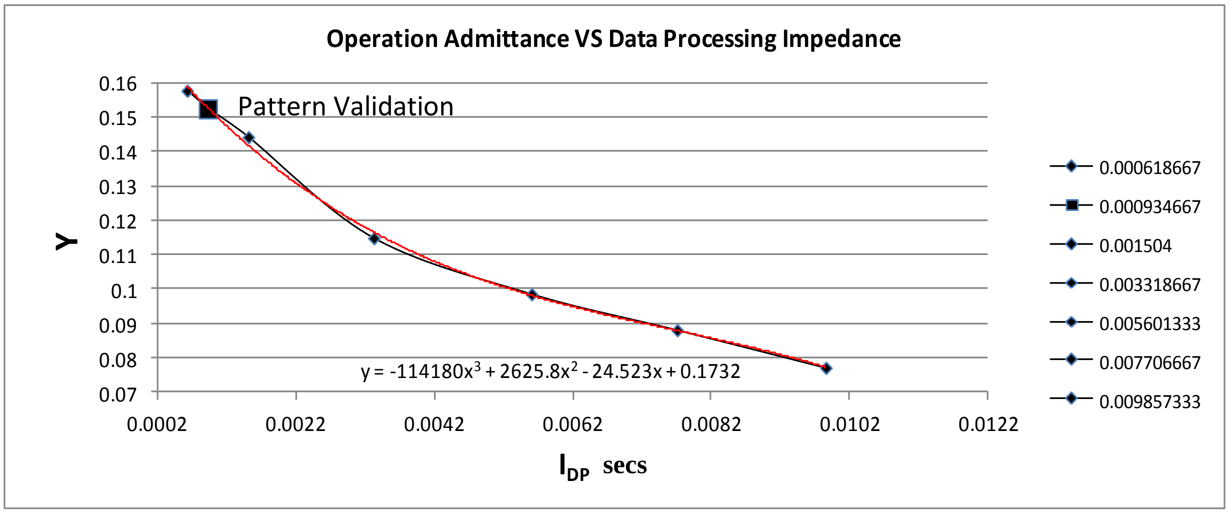


Figure 30: Function Graph of Y vs I_{DP} for varying Data Processing on HE1

For ‘Y’ versus ‘I_{DP}’, the trend line with *Polynomial* regression of 3rd order was the best fit:

$$Y = -114180I_{DP}^3 + 2625.8\ I_{DP}^2 - 24.523I_{DP} + 0.1732 \qquad (6.5)$$

A gradual decrease in the application operation Admittance ‘Y’ is observed with increase of the Data Processing Delay Points latency. The observed data sets affirm the second hypothesis of this thesis which states that under a given workload condition and hosting environment, the operation Admittance ‘Y’ can be expressed as a statistical *function* of the total cumulative latency of all the Delay Points of a particular type across all the supporting application components catering to the operation. In our example TCG application, this is analogous to expressing ‘Y’ of the getConfidence(…) operation as a statistical function of the total cumulative latency of all the in-memory data processing activities across TA Component1 to TA Component5 and the FA Component1.

For the given range of ‘V’, the function of ‘I_{DP}’ as shown in Equation (6.5) above can be used to predict a possible value of operation Admittance say ‘Y_p’ corresponding to any future value of ‘I_{DP}’. This ‘Y_p’ can then be fed into the *PALP* model ($P = YV + c$) for that particular ‘V band’ to have an estimate of the possible performance ‘P’. The predicted performance P_p can be calculated using the Equation (6.3) : $P_p = Y_pV + c$

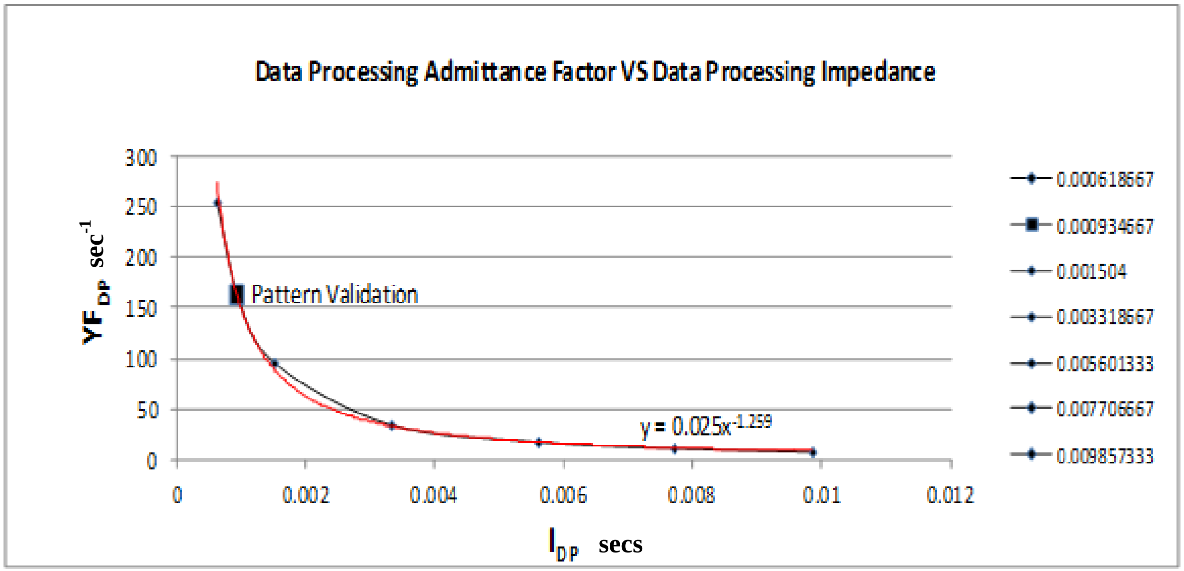


Figure 31: **Function Graph of YF_{DP} vs I_{DP} for varying Data Processing on HE1**

As shown in Figure 31, for ‘$\mathrm{YF}_{\mathrm{DP}}$’ versus ‘$\mathrm{I}_{\mathrm{DP}}$’, pattern line with *Power* regression was the best fit:

$$YF_{DP} = 0.025I_{DP}^{-1.259} \qquad (6.6)$$

Initially, there is a steep decrease of the Data Processing Admittance Factor for an increase of the Data Processing Delay Points latency (or Impedance). However, it later reaches saturation and flattens out with further increase of the Delay Points latency. Results affirmed (with some approximation errors) the *distinct* underlying patterns of variations in ‘Y’ due to changes in application component Delay Points under a given load. From a *projected* value of ‘$\mathrm{YF}_{\mathrm{DP}}$’ corresponding to a given *actual* ‘I_{DP}’, we can also *project* ‘Y’:

$$Y = YF_{DP} \text{ x } I_{DP} + e \qquad (6.7)$$

where ‘*e*’ is the error factor.

To observe the impact of augmented hardware on the various measured data under similar workload conditions, controlled experiments are performed separately with the same application framework implemented on Hosting Environment 1 (HE1) and Hosting Environment 2 (HE2) as described in Sections 4.6.1 and 4.6.2. Table 7 and Figure 32 present the data obtained from the experiment runs on HE2 for the function graph of ‘Y’ versus ‘I_{DP}’. As with HE1, pattern integrity validation, which is performed subsequently to confirm the correctness of the extracted functions, is highlighted in the table and indicated on the graph.

OVERALL RESULTS COMPARISON										
Data Process Loop	Total Operation Requests	Total time for requests in milliseconds	Operation Requests per second	Operation Stress Point (Requests per second)	Operation Load Potential (V) in Requests per second	Average Operation Response Time (ART) in seconds	Operation Performance (P=1/ART) in seconds^{-1}	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual Data Processing latency (I_{DP}) in seconds	Admittance Factor (YF_{DP} = Y/I_{DP})
1	150	19672	7.625050834	40	32.37494917	0.243526667	4.106326475	0.163757059	0.000618467	264.7791193
3	**150**	**19890**	**7.54147813**	**40**	**32.45852187**	**0.253953333**	**3.937731347**	**0.158141254**	**0.000924667**	**171.0251483**
5	150	20358	7.368110816	40	32.63188918	0.25688	3.892868265	0.155926255	0.001404	111.058586
10	150	21108	7.106310404	40	32.8936896	0.31452	3.179448048	0.132996575	0.003218667	41.32039398
15	150	22265	6.737031215	40	33.26296879	0.430813333	2.321190927	0.105717892	0.005401333	19.5725546
20	150	23343	6.425909266	40	33.57409073	0.501413333	1.994362603	0.095003693	0.007506667	12.65590935
25	150	24469	6.130205566	40	33.86979443	0.51931333	1.925619741	0.092144632	0.009657333	9.541415703

Table 7: Experimental Data for varying Data Processing activities in HE2

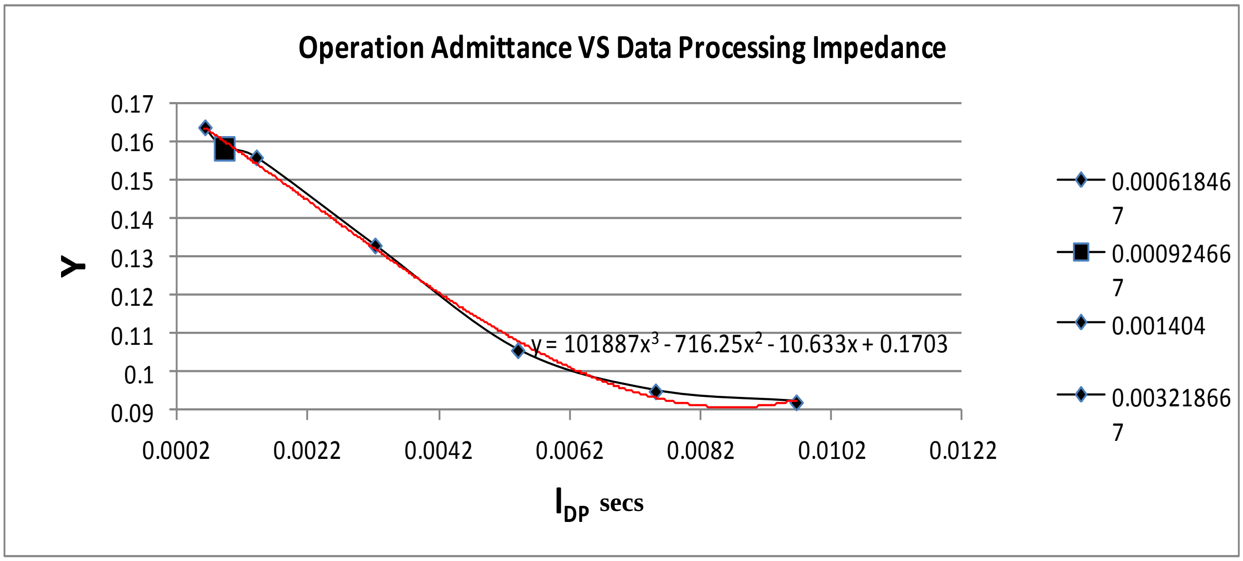


Figure 32: Function Graph of Y vs I_{DP} for varying Data Processing on HE2

Comparing the graph functions of application operation Admittance ‘Y’ versus the Data Processing Delay Point impedance/latency ‘I_{DP}’ in Figures 30 and 32, it is observed that the decrease in the ‘Y’ with the increase of the latency/impedance of the Data Processing Delay Points ‘I_{DP}’ is sharper on HE1 than on HE2, which comprises a superior hardware configuration with Dual Core processor. Less sharp decrease in ‘Y’ implies improved performance on HE2 than HE1 for the same increase in Delay Point latencies. This observation corroborates the fact that augmented hosting provision under same workload conditions has a positive impact on the performance of the application operation.

6.3.1 Verifying Precision of the Derived Patterns

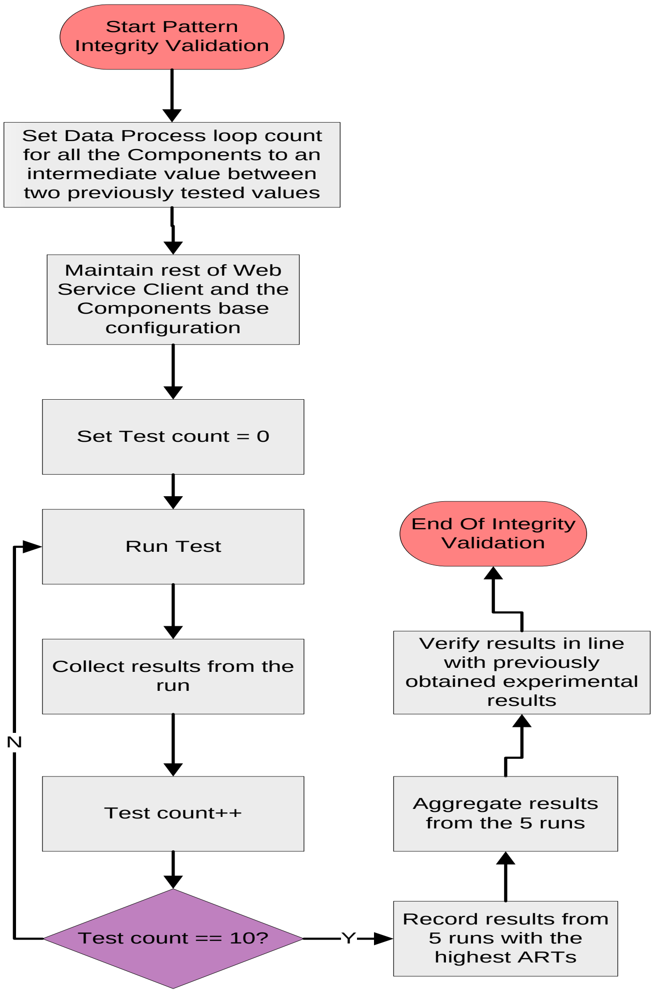

Start Pattern Integrity Validation
Set Data Process loop count for all the Components to an intermediate value between two previously tested values
Maintain rest of Web Service Client and the Components base configuration
Set Test count = 0
Run Test
Collect results from the run
Test count++
Test count == 10?
N
Y
Record results from 5 runs with the highest ARTs
Aggregate results from the 5 runs
Verify results in line with previously obtained experimental results
End Of Integrity Validation

Figure 33: Steps of verifying the derived functions

As shown in Figure 33 above, tests are performed to verify the precision of the derived patterns (statistical functions). The objective of the verification tests was to examine whether the results obtained from any intermediate data processing configuration conformed to the previously derived statistical functions. This is achieved by assessing the delta between the measured data during the tests and the data obtained from using the derived functions.

During evaluation of the second hypothesis, the data processing configurations were set to 1, 5, 10, 15, 20 and 25. For the subsequent verification of precision of the derived functions, the data processing configuration is set to 3, which is intermediate to 1 and 5. For this intermediate configuration, 10 test runs are executed with each test run comprising 30 transaction requests of the same class of workload and representative of real-life scenarios. Data from 5 of the test runs with the highest ARTs are recorded. Mean of all the relevant data are calculated across the 5 test runs.

Comparing the actual data measured during the tests to that of the forecast obtained from using the statistical functions it transpired that for a Data Processing cumulative latency of 0.000934667 seconds, the operation Admittance ('Y') calculated from actual data observed is 0.088226648 while the value predicted by the previously derived function was 0.088159076. This yielded an error of 0.076% approximately, which is very low considering this is representative of 150 sets of request-responses. However, further calibration of the derived function can be achieved by repeating this process and recursively adjusting the Y vs I_{DP} function for every new set of 'Y' calculated from actual data observed and that predicted by the Y vs I_{DP} function. Through calibration, we shall be able to minimise the error and achieve much higher precision for any future Data Processing algorithm.

6.3.2 Statistical Power of the Experiment

The data in Tables 6 and 7, the corresponding graph functions in Figures 30, 31, 32 and the subsequent data obtained from the experiments to validate the integrity of the extracted patterns prove that under a given workload condition and hosting environment, the operation Admittance ('Y') can be expressed as a statistical *function* of the total cumulative latency of all the Data Processing Delay Points across the supporting application components catering to the operation. This refutes the *second null hypothesis* (**$H2_0$**), which can be rejected.

A *Type-II-error* (Wohlin et al., 2012) would have occurred had the experiments not indicated the true pattern of relationship between the operation's Admittance and the total latency of the Data Processing Delay Points. The experimental data demonstrated a relationship pattern. Further experiments to validate the integrity of the extracted patterns highlighted the following:

For a Data Processing cumulative latency of 0.000934667 seconds, the actual operation Admittance observed is 0.088226648 while the value predicted by the derived statistical function was 0.088159076, yielding an error of 0.076% approximately. This is very low considering the fact that the test data is representative of 150 round trip requests-response transactions.

In view of the accuracy of the extracted patterns, we consider the probability *P*(Type-II-error) approaches zero within the bounds of the conducted experiments. Given the expression of the statistical power of an experiment in (5.3), if *P*(Type-II-error) approaches 0, the statistical power, which is 1 - *P*(Type-II-error), approaches 1. This is very high. Hence, the collated data from the experiments is proven ideal to reveal the true pattern (Wohlin et al., 2012).

Other types of Delay Points like XML parsing and processing for Request Authentication and Authorization are also tested in similar ways and experimental results are collected. They all showed distinct trends of variation of the 'Y' and in turn overall performance 'P' for variations to the underlying Delay Point impedances/latencies. In our experiments, we have used SAX Parsers. The underlying system resource level implementation of this parser will be the same irrespective of whether the XML has 100 nodes or 500 nodes. But what will matter is the degree of systems resource usage (for 100 it will be less and for 500 more) as that will impact the overall systems resource utilization of the operation. The former is related to the individual Delay Point latency while the latter is related to the overall operation's 'Y'. An XML may be modified in innumerable ways. What is of significance is the variation to the degree of systems resource utilization by the parser due to those modifications, which manifests itself through variation in the Delay Point latency/delay, which is being measured. As this impacts the overall systems resource utilization in a particular way, we observe 'Y' and consequently 'P' being impacted in particular patterns by individual Delay Points.

6.4 Threats to the Validity of the Findings

We evaluated the methods through controlled experiments on a test-bed application framework designed to resemble real life applications and our example TCG application in particular. The framework was hosted independently on two separate environments of different specifications. The specifications of the two hosting environments namely ***HE1*** and ***HE2*** are provided in Sections 4.6.1 and 4.6.2 respectively. However, each environment was implemented on a single hardware and our findings are limited to this setup. In typical heavy duty production systems, applications are hosted on fault tolerant, load balanced environments with multiple physical or virtual servers. Hence, future large-scale experiments will be needed to validate the findings on real life applications.

The threats to validity are discussed following the same structure as in Section 5.4. The following sections discuss the other threats to validity for the experiments we conducted on our test-bed application framework:

6.4.1 Conclusion Validity

Low Statistical Power – The data patterns initially extracted during the evaluation of the second hypothesis of this thesis are subsequently validated to verify their accuracies. As demonstrated in Sections 6.1.2 and 6.3.2, the statistical powers of our experiments are proven to be high, highlighting the ability of the experiments to reveal the true patterns in the collated data. As shown in Figures 19 and 27, for each application configuration, 10 test runs are executed with each test run comprising 30 transaction requests of the same class of workload. Data from 5 of the test runs with the highest ARTs are recorded. 6 different application configurations are tested. Hence, to ascertain the patterns and trends for each of the different types of Delay Points, *1800* sets of responses and probed data are observed. This signifies good coverage of data.

Violated Assumptions of Tests - Several boundary conditions related to the QN model, application hosting environment, network provisioning and others need to be observed for the proposed approach. In our experiments, we ensured that the conditions are observed. However, if any of the boundary conditions is violated, the methodology will yield incorrect results. For example, technology refreshes are out of scope of this thesis. If due to business requirements, changes are applied not only to the application layer but also to the underlying infrastructure, the approach may not work upfront. A fresh set of lookup reference data have to be created for the application on the new infrastructure and only then the reference data will be valid. Another example is that only *formal* application operation consumption contracts are in scope of this thesis with dedicated, controlled network traffic and *not* any random application access over public network. Hence, at *runtime*, no unpredictable fluctuation of network bandwidth or latency is assumed. If this assumption is violated, network latency needs to be included in the list of predictor variables.

The proposed approach allows the software engineers to fine tune the application algorithm and configuration to attain a specified performance level without the need to involve any external performance experts. The target users of the proposed methodology are the software engineers who have knowledge of the application codebase, which is required to identify the Delay Points. However, the requirement of having knowledge of the application codebase may become a constraint on the methodology. Understanding and knowledge of the Delay Points is crucial for the accuracy of the results as coverage of Delay Points needs to be complete. Probing the Delay Points accurately is critical to the precision of the performance analysis. Incomplete coverage of the Delay Points may lead to incorrect results, analysis and forecast. This implies that in our example TCG application, if we are assessing the impact of the in-memory data processing Delay Points on the getConfidence(…) operation's Admittance ('Y'), we need to probe and get the cumulative latency of all the data processing Delay Points spread across the different TA and FA components. In this thesis, the prototypical application framework was designed and developed by us, which ensured full coverage of the Delay Points.

Fishing and the Error Rate – During the evaluation of the methods every outcome was observed and recorded with the aim of extracting patterns out of the collated data. Hence there was no attempt of *fishing* as such. In Sections 6.1.2 and 6.3.2, the statistical powers of our experiments are demonstrated to be high, highlighting the ability of the experiments to reveal the true patterns in the collated data.

Reliability of Treatment Implementations – The process of automation of the experiments conducted is the same as described in Section 5.4.1. As the instrumentation is done only once upfront, any risk of difference in type of measurements across different test runs is mitigated. The risk of dissimilar implementations between different persons applying the treatment is eliminated as there is no involvement of human participants.

Random Irrelevancies in Experimental Setting – As our experiments are conducted in a controlled environment, the risk of any external impact on the experiments is mitigated.

Random Heterogeneity of Subjects - A request involving a complex database query will differ than a request involving a simple in-memory data processing regarding the type and resource usage. Our model and methods are confined to application operations to classify the request type. Applying the same logic as explained in Section 5.4.1, the lookup datasets and patterns extracted from the experiments are applicable to a particular application operation associated to a single class of workload. However, the same patterns may not be valid for other operations of the same application, for which the patterns have to be re-created.

6.4.2 Internal Validity

The application of specific independent variables (inbound workload and the latencies of the Delay Points of a specific type) and a group of dependent variables (the operation's Admittance and Performance) eliminated the *Single Group Threat*. The experimental environment is maintained constant by conforming to all the boundary conditions related to the QN model, application hosting environment, network provisioning and others as explained in Section 3.3.1. This eliminated the risk that the *history* will affect the experimental results due to changes in the circumstances. As the experiments are performed in a controlled environment there is no unexpected factor influencing the resultant data. Effects of motivation, maturation and other such factors do not apply as the experiments do not involve any human participants.

6.4.3 Construct Validity

Design Threats - For the same reasons explained in Section 5.4.3, it is practically not possible to incorporate all the various types of component complexities in the experimental framework used to evaluate the methods. While determining the impact of alterations to the Delay Points of a specific type on the operation's Admittance and Performance, we treated Delay Points of one particular type at a time. While this was required to fulfill the objective of that particular experiment, it may have possibly introduced *Mono-operation Bias* in the experiment. We have used different types of methods in our experiments to assess the impact of variations to the underlying Delay Points on the operation's Admittance. While we used statistical function graphs for changes to Delay Points of a specific type, we adopted a heuristic method to estimate the best-fit relative weights of predictor variables in multiple regressions for changes to Delay Points of multiple types. This is to eliminate the threat of *Mono-method Bias*. In this thesis, instrumentation for probing the Delay Points and the operation's response time has been done manually. This can be a lengthy manual process. However, automated instrumentation may become quite challenging if the Delay Points of the same type are implemented in different real-world applications in different ways. Nowadays, extreme high spec machines with multiple cores are available, which are typically deployed in the production environments. The workload injected during our experiments may not be sufficient to impact the Performance on those types of high spec hosting environments to the extent that patterns of variation can be extracted. More workload may be needed to be injected. While various types of application configurations are covered in the experiments, it is possible we may not have explored all the possible combinations of treatments in our experiments. There are many more types of treatments that can be applied to observe the effects.

Social Threats – As explained in Section 5.4.3, our experiments do not involve any human participants and hence these *social threats* are mitigated to a great extent.

6.4.4 External Validity

Interaction of Selection and Treatment – For each application configuration, the processing intensities of the Delay Point(s) of a specific type are set to particular levels and the other Delay Points are left unaltered. 10 test runs are executed with each test run comprising 30 transaction requests of the same class of workload and representative of real-life scenarios. Data from 5 of the test runs with the highest ARTs are recorded. 6 different application configurations are tested. Hence, to ascertain the patterns and trends for each of the different types of Delay Points, *1800* sets of responses and probed data are observed. This signifies good data coverage representative of the real world scenarios. However, despite the fact that the experiments covered reasonably good amount of data and types of Delay Points, there are other types of Delay Points, which we may not have been able to cover within the scope of this thesis.

Interaction of Setting and Treatment - The proposed approach will enable the software engineers to modify and probe the latencies of Delay Points of a specific type as appropriate through systems testing with a multi-threaded client for the operation. Typically from application hosting point of view, Development (Dev) or Systems Integration Testing (SIT) environments comprise lower specification/capacity infrastructure/hardware relative to the Production environments (Gunther, 2005). Due to this difference in hosting infrastructure specification/capacity, there may be differences in the performance outputs across the environments. If the differences between the hosting environments are significant, experiments in the non-production environments may not yield results reflective of the production environment. Production environments are typically built with the highest specification/capacity (Gunther, 2005). In this thesis, to observe the impact of augmented hardware under similar workload conditions, controlled experiments are performed separately on the application framework implemented on two different hosting environments (HE1 and HE2), one of which comprised hardware of higher specification than the other. As described in Sections 6.1, 6.2 and 6.3, the outcome of the experiments demonstrated improved Admittance and Performance under the same workload on the hosting environment with augmented hardware. Hence, it is likely that the performance results on a non-production environment will highlight a "worse-case scenario" under a given load condition for an application operation. However, worst-case tolerancing is a safer approach than statistical tolerancing. It ensures that if the inputs are within their respective tolerances, the output is guaranteed to be within its worst-case tolerance (Taylor, 1995). Hence, if not better, the production environment's performance is expected to be at the least the same as the performance observed on the non-production environments for any given workload condition.

Interaction of History and Treatment – The automated nature of our experiments (with only the instrumentation for probing the Delay Point latencies and the operation's response times performed manually) eliminates the risks of any difference in type of measurements across different test runs over time. Any risk of dissimilar implementations towards the treatment over time is also eliminated as the experiments do not involve any human participants.

6.4.5 Repeatability

Details of the technical specification of the end to end application framework along with the details of all the hosting environments have been described in the thesis. The process flows are documented through UML models (Static and Sequence Diagrams). The information provided should allow replication of our experiments with different sets of configurations.

6.5 Chapter Summary

Through controlled experiments, we proved the second null hypothesis (**$H2_0$**) to be false and evaluated the second hypothesis of the thesis in this chapter. The goals of the evaluation as described at the outset are achieved. The outcome of the evaluation demonstrated that the proposed approach can be used by the software developers without the need of additional quality assurance experts. The software developers will neither be required to learn new, non-standard modelling formalisms and notations nor to specify the software system using process algebras or complex SPE techniques. The approach will not need repetitive performance testing or resource intensive application profilers to assess the impact of the changes to an evolving application. It will yield precise and accurate results. It is also established that under a given workload condition and hosting environment, an application operation's Admittance ('Y') can be expressed as a statistical *function* of the total cumulative latency of the Delay Points of a particular type catering to that operation.

In a particular iteration, out of the different types of Delay Points (namely Data Processing, Database Interactions, File I/O and other types), the processing intensities of the Delay Points of one particular type are varied at a time across all the components supporting the application operation. The impact of this variation of the processing intensities of the particular type of Delay Point on the operation's Admittance and Performance is measured. From the experimentally measured and model based computed data, various types of function graphs like the operation's Admittance versus the variation of latencies introduced by a particular type of Delay Point, the operation's Performance versus its Admittance etc. are generated. The graphs showed *distinct* trends in variation, which are consistent but non-linear. Accepting approximation error, for simplicity, the function graphs are verified against various statistical regression types (*Linear, Exponential, Polynomial* and *Power*) and the best-fit LSF is identified for the function. The statistical regression functions for every individual type of Delay Point were distinct. For every type of Delay Point, pattern integrity validation is performed subsequently to confirm the correctness of the extracted functions.

The observed data sets affirmed the second hypothesis of this thesis which states that under a given workload condition and hosting environment, the operation Admittance ('Y') can be expressed as a statistical *function* of the total delay / latency introduced by the Delay Points of a particular type across all the supporting application components catering to the operation. The statistical power (Wohlin et al., 2012) of the experiments is also derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data.

To observe the impact of augmented hardware on the various measured data under similar workload conditions, controlled experiments are performed separately on the application framework implemented on two different hosting environments, one of which comprised hardware of higher specification than the other. Comparing the various graph functions derived from the two separate hosting environments, it is observed that the decrease in the operation's Admittance with the increase of the latency/impedance of the Delay Points is more rapid on the hosting environment of lower hardware specification than the one with higher hardware specification. Given the inherent definition of Admittance, less rapid decrease in Admittance implies improved performance on the hosting environment with augmented hardware. This observation corroborates that augmented hosting provision under same workload conditions has a positive impact on the performance of the application operation. In this chapter, we also verified the statistical power of the experiments conducted and proved that the experiments have the ability to reveal the true underlying patterns of the collated data. An assessment of the threats to the validity of the experimental findings has also been presented in this chapter.

To address real life scenarios, in the next chapter a matrix based technique is explored for the assessment of impact of simultaneous changes to multiple types of Delay Points on the Admittance and Performance of application operations. This technique applies a heuristic method for estimating the relative weights of predictor variables in multiple regressions.

7 Multi-Type Delay Points Changes: Method Evaluation

In mission critical Line Of Business (LOB) systems / applications, typically different types of application activities need to be modified simultaneously to address changes to business requirements. Very rarely just one type of activity is modified. Delay Points are the nodes in the application where these various types of processing activities take place. Let us consider our example scenario cited in section 3.1. A shift in the global financial market sentiment due to military tensions in the Middle East resulted in significant changes to the algorithms of the TA components and that of the FA component of the TCG application. As the market sentiment moved suddenly, changes to the business logic had to be implemented very rapidly. Temporary new logic were mostly published in text files rather than having those in the database. The reason being the latter would have warranted database schema changes, most probably requiring the tables and views to be recreated etc., which would have needed more time and resources. As the financial market became very sensitive to the global news and events, the frequency of processing the XML feeds from the external News and Events Feed Provider had to be increased as well. All of these changes warranted modifications to the algorithms of the TA Component1 to TA Component5 and the FA Component1. This in essence meant that most of the processing activities of the TA and FA components involving in-memory data processing, file I/O, database interactions and XML parsing/processing were required to be modified simultaneously.

To address scenarios similar to the one cited above with our example TCG application, it is important to establish a method to analyze and predict the impact on an application operation's performance due to simultaneous changes to Delay Points of multiple types across the supporting application components. The third hypothesis of this thesis addresses this scenario. The hypothesis states that under a given workload condition and hosting environment, in the event of simultaneous changes to multiple types of Delay Points, the total cumulative latency of each type has an implicit relative weight associated to it, which determines the degree of its impact on the operation Admittance ('Y'). Determining the *best-fit* relative weights will facilitate predicting the operation's 'Y' (say Y_p) for any future set of latency values of the Delay Points. This predicted 'Y_p' can then be fed into the *PALP* model for that particular 'V band' to have an estimate of the possible performance '*P*'. The predicted performance P_p can be calculated using the Equation (6.3) : $P_p = Y_pV + c$

Similar to the evaluation in Chapter 6, we divide the evaluation presented in this chapter into an intrinsic and an extrinsic one as well. The goal of the intrinsic one remains the same as that described in Chapter 6 i.e. to demonstrate that our approach addresses the aspects a), b), c), d) and e) of the main research question as explained in Section 1.3. However, the extrinsic part of this evaluation is concerned with quantifying the effects of the changes to the Delay Points of multiple types across the application components catering to an operation and establishing the fact that under a given workload condition and hosting environment, the degree of impact of the total cumulative latency of the Delay Points of each type on the operation's Admittance ('Y') is determined by an implicit relative weight associated to the total cumulative latency of the Delay Points of that type. The *best-fit* relative weights will help in predicting the operation's 'Y' for any future set of latencies of the Delay Points. This will establish the third working hypothesis of this thesis as explained in Section 3.4.2.

As with the previous evaluation in Chapter 6, we follow the guidelines on experimentation in software engineering as they are presented in (Wohlin et al., 2012). The same approach to controlled experiments in a laboratory environment is adopted. However, unlike the previous evaluation where some of the Delay Point variables were kept constant while varying some other Delay Point variables, in this evaluation different treatments are applied to different Delay Point subjects simultaneously and the effects on outcome variables are measured. The effect of the manipulation is measured and based on this measurement, statistical analysis is performed. Details of the controlled experiments are provided in Section 7.1 below. As advocated by (Wohlin et al., 2012), the statistical power of the experiments is determined to demonstrate the ability of the experiments to reveal the true pattern in the collated data. The run-time monitoring approaches proposed by (Westermann et al., 2010) to build mathematical models serving as interpolation of the measurements is adopted to evaluate the third hypothesis. While a *black-box* view of the internal structure of the underlying system is taken, the approach adopts a *white-box* view for the application operation and focuses on the observable data.

To compute the statistical power of the experiments to demonstrate the ability of the experiments to reveal the true pattern in the collated data, a third ***null hypothesis*** (**$H3_0$**) is formulated corresponding to the third hypothesis. The null hypothesis states that under a given workload condition and hosting environment, the degree of impact of the simultaneous changes to the Delay Points of multiple types, on an operation's Admittance ('Y'), is not determined by the implicit relative weights associated with the total cumulative latency of the Delay Points of each type. Determining the *best-fit* relative weights will not facilitate predicting the operation’s ‘Y’ for any future set of latency values of the Delay Points as the cumulative latencies of the Delay Points of each type influence ‘Y’ at random.

In (Johnson, 2000) a very effective technique to predict the best-fit weights is proposed by applying a heuristic method for estimating the relative weights of predictor variables in multiple regression. This method is proved to be computationally efficient with any number of predictors and is shown to produce results that are very similar to those produced by more complex methods. This thesis adopts the technique proposed by Johnson to determine the *best-fit* relative weights associated with the latencies of each type of Delay Points. The relative weights determine the proportionate contribution of those respective latencies to the overall operation Admittance. The set of relative weights is then used to predict the operation Admittance for any future set of latency values for the different Delay Points. This approach leads to a matrix based predictive method to forecast the application operation’s performance for simultaneous changes to multiple types of underlying application component activities.

7.1 Simultaneous changes to multiple types of Delay Points

In all of the above experiments, one particular type of Delay Point was varied keeping the rest of the configuration constant. But real world industrial scale component modifications will be more complex with Delay Point of various types being modified simultaneously. To address this scenario, Delay Points of multiple types (Database Interactions, Data Processing, File I/O and XML Processing) are varied simultaneously and experiments are run. The Standard Deviation of the ARTs for 5 test runs indicated a very low deviation. Hence, for a given set of Delay Point algorithm and configuration, ‘5’ is considered a reasonably optimal number for test run iterations. In line with the 50% percentile value approach by (Cherkasova et al., 2007), for each algorithm configuration of the set of Delay Points, the aggregate of 5 runs is calculated to smooth out the data and remove any occasional noise. Experiments are run for 5 different algorithm configurations for the set of Delay Points.

For every configuration run, the values of the application operation performance ‘P’ and admittance ‘Y’ are recorded in the same way as the experiments presented in Chapter 6. In the same manner as the previous experiments, a positive Load Potential ‘V’ is maintained and a Stress Point of 40 requests/sec is assumed in all our controlled experiments. ‘V’ is calculated from the difference of this assumed Stress Point and the generated workload. The processing intensities of the various types of Delay

Points of the components were incrementally varied keeping ‘V’ positive. For every configuration, using the *PALP* model Equation (5.2), ‘Y’ is calculated from the measured values of ‘P’ and ‘V’ applying Equation (6.1) i.e. $Y = (P - c) / V$, where the values of the y-intercept ‘*c*’ and load potential ‘V’ are specific to the range of ‘V’.

The aggregated results from 5 different process load configurations are as follows:

Orchestration Service Performance (P=1/ART) in seconds^{-1}	Calculated overall Operation Admittance (Y = (P-c)/V)	Average actual DB Access latency (I_{DB}) in seconds	Average actual Data Processing latency (I_{DP}) in seconds	Average actual File I/O latency (I_{FIO}) in seconds	Average actual Authentication latency (I_{AN}) in seconds	Average actual Authorization latency (I_{AR}) in seconds
0.096278051	0.002212067	1.14508	0.00104	0.063876667	0.0072	0.00339
0.07665596	0.00176775	1.549756667	0.001193333	0.208513333	0.003893333	0.002436667
0.07082897	0.001620992	0.879663333	0.002133333	1.082626667	0.006343333	0.005226667
0.059141444	0.001352816	1.187556667	0.002346667	1.032653333	0.00823	0.005473333
0.05007248	0.001151423	1.219683333	0.00275	1.375003333	0.003016667	0.003183333

Table 8: Calculated Y and corresponding measured Delay Point latencies

The above dataset presents the measured latencies of the different types of Delay Points and their corresponding ‘Y’ as computed from the measured ‘P’ and ‘V’ values for the different configurations of the Delay Point algorithms. The latencies of the Delay Points (DB Access, Data Processing, File I/O etc) are treated as the predictor variables in a multiple regression. The computed ‘Y’ is treated as the resultant dependant variable based on the predictor variables. In line with the approach proposed by (Johnson, 2000), the above data may also be presented in the form:

$$\boldsymbol{AX = B} \qquad (7.1)$$

where:

- *A* is the [5x5] matrix containing rows of Delay Point Impedances for Database Interactions (I_{DB}), Data Processing (I_{DP}), File I/O (I_{FIO}) and XML Processing (I_{AR} and I_{AN}) from different test runs i.e. the sets of predictor variables

- *B* is the single column [5x1] matrix of the calculated 'Y' for each row in *A* i.e. the dependant variable
- *X* is the single column [5x1] matrix of the *best-fit* relative weights associated with the latencies of each type of Delay Points

The *best fit* value of *X* is calculated through matrix transpose and inverse in the following way (Johnson, 2000):

$$\boldsymbol{X = (A^T A)^{-1} A^T B} \qquad (7.2)$$

The Delay Points relative weight matrix *X* is calculated to facilitate projection of 'Y' (say Y_{DLPrw}) for any future arbitrary combination of Delay Point latencies. This 'Y_{DLPrw}' can then be fed into the *PALP* model for that particular 'V band' to have an estimate of the possible performance '*P*' (say P_{DLPrw}). The predicted performance 'P_{DLPrw}' can be calculated using the Equation (6.3) i.e.:

$$P_{DLPrw} = (Y_{DLPrw} * V) + c$$

We present below the step by step process by which the DLPrw matrix X is derived. As with the single type Delay Point modification experiments, further experiments are performed subsequently to verify the precision of the derived DLPrw matrix X:

7.2 Delay Point Relative Weight (DLPrw) Matrix

The data obtained from experiments as presented in Table 7 may be represented in the form of Equation (7.1) in the following way:

		A			**X**		**B**
I_{DB}	**I_{DP}**	**I_{FIO}**	**I_{AN}**	**I_{AR}**	**DLPrw Matrix**		**Y**
1.14508	0.00104	0.063876667	0.00722	0.00339	w1		0.002212067
1.549756667	0.001193333	0.208513333	0.003893333	0.002436667	w2	=	0.00176775
0.879663333	0.002133333	1.082626667	0.006343333	0.005226667	w3		0.001620992
1.187556667	0.002346667	1.032653333	0.00823	0.005473333	w4		0.001352816
1.219683333	0.00275	1.375003333	0.003016667	0.003183333	w5		0.001151423

In the above matrices, w1, w2, w3, w4 and w5 are the best-fit relative weights associated to I_{DB}, I_{DP}, I_{FIO}, I_{AN} and I_{AR} respectively.

A^T =

1.145080000	1.549756667	0.879663333	1.187556667	1.219683333
0.001040000	0.001193333	0.002133333	0.002346667	0.002750000
0.063876667	0.208513333	1.082626667	1.032653333	1.375003333
0.007220000	0.003893333	0.006343333	0.008230000	0.003016667
0.003390000	0.002436667	0.005226667	0.005473333	0.003183333

A^TA =

7.384679783	0.011057803	4.252038802	0.033334164	0.022638321
0.011057803	0.000020126	0.008829413	0.000053296	0.000039182
4.252038802	0.008829413	4.176645611	0.020787127	0.016412298
0.033334164	0.000053296	0.020787127	0.000184357	0.000121766
0.022638321	0.000039182	0.016412298	0.000121766	0.000084838

$(A^TA)^{-1}$ =

73.126576858	-137811.610463035	231.170000124	18067.646113427	-26518.592219414
-137811.610463035	262699136.751477000	-441567.622706696	-34841834.398532800	50878569.949531800
231.170000124	-441567.622706696	747.448690658	60115.311849377	-88629.503157987
18067.646113427	-34841834.398532700	60115.311849377	5191460.766442940	-10441.560956830
-26518.592219414	50878569.949531700	-88629.503157986	-7810441.560956830	11945971.379581500

A^TB =

0.009709420
0.000014209
0.005245028
0.000047743
0.000031348

DLPrw Matrix $X = (A^TA)^{-1}A^TB$ =

-0.004356602
10.074393338
-0.017590749
-1.320577244
2.177624419

The Delay Point algorithm configuration of the components was varied in uniform steps. This resulted in a more linear regression for the application operation Admittance ('Y'). To cross check the integrity of X (DLPrw Matrix), we back-calculated matrix B' (single column Admittance Matrix) by multiplying A with X and compared it with the original matrix B. The values matched up till 6 decimal places despite DLPrw matrix comprising the *best-fit* values of the relative weights. This match was primarily due to the linear regression of the operation Admittance. There were some differences from the 7th decimal place onwards due to rounding off of all the values during the intermediate steps of the process.

AX = B' =

0.002212652
0.001767226
0.001620335
0.001352902
0.001151930

7.3 Verifying Precision of DLPrw Matrix

The DLPrw matrix had to be validated to assess the precision of the forecast for overall application operation Admittance (Y) for any given set of Delay Point latencies.

To achieve this, tests are run on the application framework with the Delay Point algorithm configurations for all the Delay Points ((Database Interactions, Data Processing, File I/O and XML Processing) set to new values, different from all the values previously used to derive the DLPrw matrix. Experimental data for the following are recorded:

- the measured *actual* ‘P’
- the *computed* value of ‘Y’ (say $\mathrm{Y}_{\mathbf{PALP}}$) based on the measured actual ‘P’ and ‘V’ by applying the Equation (6.1) i.e. $Y = (P – c) / V$ derived from the *PALP* model.
- the measured *actual* average Delay Point Impedances/latencies for Database Interactions (I_{DB}), Data Processing (I_{DP}), File I/O (I_{FIO}) and XML Processing (I_{AR} and I_{AN})

The single row Delay Point Impedance matrix comprising I_{DB}, I_{DP}, $\mathrm{I}_{FIO,}$ I_{AR} and I_{AN} is then multiplied by the previously derived single column DLPrw matrix *X* to calculate the overall ‘Y’ again (say $\mathrm{Y}_{\mathbf{DLPrw}}$). This is the projected value of ‘Y’ purely based on the matrix model.

The delta between $\mathrm{Y}_{\mathbf{DLPrw}}$ and $\mathrm{Y}_{\mathbf{PALP}}$ is observed to be 4.459133225%. As a first iteration, this error percentage is considered acceptable. Further calibration of the DLPrw matrix method can be achieved by repeating this process and recursively adjusting the relative weights in the DLPrw matrix *X* for every newly measured set of Delay Point latencies and the corresponding Y_{PALP}. Through calibration, we shall be able to minimise the delta between Y_{DLPrw} and Y_{PALP} and achieve much higher precision for random future Delay Point algorithm configurations.

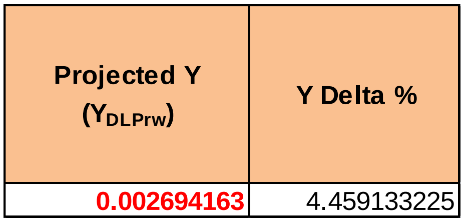
Data from the validation test runs are shown below:

PALP model Y (Y_{PALP})	**I_{DB}**	**I_{DP}**	**I_{FIO}**	**I_{AN}**	**I_{AR}**
0.002574027	1.034386667	0.00078	0.04260667	0.003173333	0.001966667

Previously derived DLPrw Matrix X:

DLPrw Matrix
-0.004356602
10.07439334
-0.017590749
-1.320577244
2.177624419

Projected Y_{DLPrw} and error percentage $((|Y_{PALP} – Y_{DLPrw}| / Y_{PALP}) * 100)$:



Projected Y (Y_{DLPrw})	Y Delta %
0.002694163	4.459133225

7.3.1 Statistical Power of the Experiment

Results from the experiments described in Section 7.2 and 7.3 reject the third *null hypothesis* (**$H3_0$**), which states that under a given workload condition and hosting environment, the degree of impact of the simultaneous changes to the various types of Delay Points, on an operation's Admittance ('Y'), is not determined by the implicit relative weights associated with the total cumulative latency of the Delay Points of each type. Determining the *best-fit* relative weights will not facilitate predicting the operation Admittance for any future set of latency values of the Delay Points as the different Delay Points influence the operation Admittance at random.

A *Type-II-error* would have occurred had the controlled experiments not indicated the relative weights associated to the total cumulative latency of the Delay Points of each type. Further experiments to validate the integrity of the derived relative weights (DLPrw matrix) highlighted that the operation Admittance can be predicted with an accuracy of 95.54%, which is very high.

In view of the accuracy of the derived relative weights, we consider the probability *P*(Type-II-error) approaches zero within the bounds of the conducted experiments. Again, given the expression of the statistical power of an experiment in (5.3), if *P*(Type-II-error) approaches 0, the statistical power approaches 1, which is very high. Hence, the collated data from the experiments are proven ideal to reveal the true pattern (Wohlin et al., 2012).

7.4 Calibration of the DLPrw Matrix Model

Some work is carried out towards calibrating the DLPrw matrix model. During evaluating the third hypothesis and deriving the single column DLPrw matrix, the Delay Point algorithm configurations of the components were varied in uniform steps, which resulted in a more linear regression for the application operation Admittance (‘Y’). This yielded forecasts with about 4.4% error approximately. In real life, the algorithm variations across the components may not be uniform. Hence the model needed to be calibrated for wider coverage of the types of Delay Point algorithm variations of the components. The algorithms of the Delay Points in the application framework are subjected to various configurations at random. This time the number of iterations is set to a higher value than the number of iterations conducted while evaluating the third hypothesis. Given the heuristic nature of this approach, 9 iterations instead of 5 are performed with varying Delay Point configurations across all the different types of Delay Points. Randomizing the process intensities of the Delay Points’ algorithms provided greater coverage of the Delay Point impedances/latencies. This facilitated increase in the forecast precision for various *random* Delay Point configurations thus reducing percentage error and making the DLPrw matrix method more robust. The aggregated results from 9 different process load configurations are as follows:

Calculated overall Operation Admittance (Y = (P-c)/V)	**Average actual DB Access latency (I_{DB}) in seconds**	**Average actual Data Processing latency (I_{DP}) in seconds**	**Average actual File I/O latency (I_{FIO}) in seconds**	**Average actual Authentication latency (I_{AN}) in seconds**	**Average actual Authorization latency (I_{AR}) in seconds**
0.001720278	1.500333333	0.000513333	0.1064	0.00253	0.00297
0.002041865	1.378543333	0.00094	0.05179	0.003656667	0.002283333
0.002204146	1.22150667	0.00098667	0.10039667	0.00344	0.002293333
0.002449165	0.954673333	0.001276667	0.08976667	0.003523333	0.002393333
0.001840742	1.281086667	0.00052	0.466443333	0.00344	0.00381
0.001937238	1.42033	0.001453333	0.080206667	0.006246667	0.00336667
0.001898819	1.460536667	0.001393333	0.077826667	0.004726667	0.00225
0.00196564	1.308123333	0.000723333	0.2152	0.00344	0.003043333
0.001899654	1.41837	0.00084	0.17285	0.003116667	0.003426667

Table 9: Calculated Y and corresponding actual Delay Point latencies

The above dataset presents different combinations of the Delay Point latencies and their corresponding calculated ‘Y’ by applying the formula $Y = (P - c) / V$ from the *PALP* model as explained in section

7.1. The Delay Points (DB Access, Data Processing, File I/O etc) are treated as the sets of predictor variables cumulatively resulting in the dependent variable 'Y'. As with the initial experiment the data is presented in the form as Equation (7.1) i.e.

$$AX = B$$

where:

- A is the [9x5] matrix containing rows of Delay Point Impedances/latencies for Database Interactions (I_{DB}), Data Processing (I_{DP}), File I/O (I_{FIO}) and XML Processing (I_{AR} and I_{AN}) from different test runs. Each row in A is a set of predictor variables
- B is the single column [9x1] matrix of the calculated 'Y' for each row in A. Every 'Y' in B represents a dependant variable
- X is the single column [5x1] DLPrw Matrix. Each row value in X represents the *best-fit* relative weight associated with the latencies of each type of Delay Points

As before the *best fit* value of X is calculated through matrix transpose and inverse in the same way as Equation (7.2) i.e.

$$X = (A^T A)^{-1} A^T B$$

The DLPrw matrix X is calculated to facilitate projection of 'Y' for any arbitrary combination of Delay Point Impedances. We present below the step by step process by which the DLPrw matrix X of the calibrated model is derived and subsequently how its precision is validated:

7.4.1 Calibrated DLPrw Matrix Model

A ***X*** ***B***

I_{DB}	I_{DP}	I_{FIO}	I_{AN}	I_{AR}	DLPrw Matrix		Y
1.500333333	0.000513333	0.1064	0.00253	0.00297	W_1		0.001720278
1.378543333	0.00094	0.05179	0.003656667	0.002283333	W_2	=	0.002041865
1.22150667	0.00098667	0.10039667	0.00344	0.002293333	W_3		0.002204146
0.954673333	0.001276667	0.08976667	0.003523333	0.002393333	W_4		0.002449165
1.281086667	0.00052	0.466443333	0.00344	0.00381	W_5		0.001840742
1.42033	0.001453333	0.080206667	0.006246667	0.00336667			0.001937238
1.460536667	0.001393333	0.077826667	0.004726667	0.00225			0.001898819
1.308123333	0.000723333	0.2152	0.00344	0.003043333			0.00196564
1.41837	0.00084	0.17285	0.003116667	0.003426667			0.001899654

$\boldsymbol{A^T} =$

1.500333333 1.378543333 1.221506670 0.954673333 1.281086667 1.420330000
1.460536667 1.308123333 1.418370000

0.000513333 0.000940000 0.000986670 0.001276667 0.000520000 0.001453333
0.001393333 0.000723333 0.000840000

0.106400000 0.051790000 0.100396670 0.089766670 0.466443333 0.080206667
0.077826667 0.215200000 0.172850000

0.002530000 0.003656667 0.003440000 0.003523333 0.003440000 0.006246667
0.004726667 0.003440000 0.003116667

0.002970000 0.002283333 0.002293333 0.002393333 0.003810000 0.003366670
0.002250000 0.003043333 0.003426667

A^TA =

16.069509373 0.011393056 1.791179641 0.045505601 0.034480103
0.011393056 0.000009303 0.001085373 0.000035188 0.000024078
1.791179641 0.001085373 0.338388355 0.004872667 0.004348859
0.045505601 0.000035188 0.004872667 0.000138763 0.000098106
0.034480103 0.000024078 0.004348859 0.000098106 0.000076939

$(A^TA)^{-1}$ =

3.040057441 -268.574500134 3.943296874 -60.230654110 -1424.435077688

-268.574500134 4107102.163701040 3415.880168499 -1149700.777149780
107969.089272573

3.943296874 3415.880168499 25.399714825 -315.318957854 -3869.792196445

-60.230654110 -1149700.777149780 -315.318957854 441802.750139248 -
158735.731524393

-1424.435077688 107969.089272582 -3869.792196445 -158735.731524396
1038707.245513630

A^TB =

0.023574995
0.000017540
0.002643049
0.000068121
0.000050987

7.4.2 Calibrated Model's DLPrw Matrix

$X = (A^TA)^{-1}A^TB$ =

0.000650225
1.921521432
0.001221496
-0.416506058
0.232007156

This DLPrw matrix is expected to facilitate wider coverage of random Delay Point algorithm configurations. However, as the processing intensities of the Delay Point algorithms are varied at random and not in uniform steps, the application operation Admittance ('Y') regression is a lot less linear this time. Hence some delta (Least Squares Error) is expected between the original operation Admittance Matrix *B* and the projected operation Admittance Matrix *B'* obtained by multiplying *A* with the derived DLPrw Matrix *X*.

AX = B' =

0.001727203
0.001772580
0.001912085
0.002271326
0.001853111
0.001993434
0.002275392
0.001776634
0.002244374

7.5 Verifying Precision of calibrated DLPrw Matrix

The calibrated DLPrw matrix had to be validated to assess the precision of the forecast for overall application operation Admittance ('Y') for any given random set of Delay Point Impedances.

To achieve this, tests are run on the application framework with the Delay Point algorithm configurations for all the Delay Points ((Database Interactions, Data Processing, File I/O and XML Processing) set to new random values, different from all the values previously used to derive the DLPrw matrix. Experimental data for the measured *actual* overall 'P', *computed* value of overall 'Y' based on the actual overall 'P' and 'V' of the *PALP* model ($Y_{\mathbf{PALP}}$) and the measured *actual* average Delay Point Impedances for Database Interactions (I_{DB}), Data Processing (I_{DP}), File I/O (I_{FIO}) and XML Processing (I_{AR} and I_{AN}) are recorded.

The single row Delay Point Impedance matrix comprising I_{DB}, I_{DP}, $I_{FIO,}$ I_{AR} and I_{AN} is then multiplied by the previously computed DLPrw matrix *X* of the calibrated model to project the overall ‘Y’ again (say $Y_{\mathbf{DLPrw}}$), this time purely based on the matrix model.

The delta between $Y_{\mathbf{DLPrw}}$ and $Y_{\mathbf{PALP}}$ was 3.391236516 %. This is better compared to the delta of 4.459133225 % during the first iteration. The results corroborated the fact that through wider coverage of random Delay Point algorithm configurations, we shall be able to minimise the delta between Y_{DLPrw} and Y_{PALP} and achieve much higher precision for Delay Points being changed at *random*.

Data from the validation test runs are shown below:

PALP **model Y (Y_{PALP})**	**I_{DB}**	**I_{DP}**	**I_{FIO}**	**I_{AN}**	**I_{AR}**
0.00243483	**1.501723333**	**0.000933333**	**0.1073**	**0.002796667**	**0.00338**

Previously derived DLPrw Matrix X:

DLPrw Matrix
0.000650225
1.921521432
0.001221496
-0.416506058
0.232007156

Projected Y_{DLPrw} and error percentage $((|\mathbf{Y}_{PALP} - Y_{DLPrw}| / Y_{PALP}) * 100)$:

Projected Y (Y_{DLPrw})	**Y Delta %**
0.002520299	3.391236516

An error factor of 3.39% is considered acceptable given the conditions under which the experiments are run. However, the *PALP* model and the matrix based method are applicable only when the following constraints are observed:

- it is applied to a particular application operation catering to a type of transaction - ensuring the same class of workload is used and not generically
- geographical location of the operation consumer hasn't changed
- no alteration to hosting environment from infrastructure/hardware and software containers point of view
- application operation provisioning over formal contracts with dedicated network lines and not over general public network to ensure no unexpected fluctuation in network bandwidths

As explained in Section 3.6 with the aid of Figure 7, in real life, the tests will be performed once upfront to populate the matrices (the multi-column Delay Point impedance/latency matrix and the corresponding single column application operation admittance matrix) and create the single column DLPrw matrix. For any change to the underlying application components catering to the application operation, the developers need to perform system testing under the desired load condition (spawning request threads) and record the Delay Point Impedances. Then using the previously created DLPrw matrix, the projected value of the application operation Admittance ($\mathrm{Y}_{\mathrm{DLPrw}}$) will be calculated. This '$Y_{\boldsymbol{DLPrw}}$' can then be fed into the *PALP* model for that particular 'V band' to have an estimate of the possible performance '*P*' (say $P_{\boldsymbol{DLPrw}}$). The predicted performance '$P_{\boldsymbol{DLPrw}}$' can be calculated using the Equation (6.3) i.e. $P_{\boldsymbol{DLPrw}} = (Y_{\boldsymbol{DLPrw}} * V) + c$. The values for 'V' and 'c' are specific to the 'V Band'.

7.6 Threats to the Validity of the Findings

7.6.1 Conclusion Validity

Low Statistical Power – The DLPrw matrix of relative weights initially computed during the evaluation of the third hypotheses of this thesis is subsequently validated to verify the precision of the computed relative weights of the Delay Point latencies. As demonstrated in Section 7.3.1, the statistical power of our experiments is proven to be high, highlighting the ability of the experiments to reveal the true patterns in the collated data. For each application configuration with Delay Point processing intensities set to various levels, 10 test runs are executed with each test run comprising 30 transaction requests of the same class of workload. Data from 5 of the test runs with the highest ARTs are recorded. 5 different application configurations are tested. Hence, to ascertain the relative weights for each of the different types of Delay Points, *1500* sets of responses and probed data are observed. This is a considerable number and signifies good data coverage.

Violated Assumptions of Tests – As explained in Section 6.4.1, several boundary conditions need to be observed for the proposed approach. If any of the boundary conditions is violated, the methodology will yield incorrect results. For example, if the business requirements warrant change in the underlying infrastructure as well, the previous DLPrw matrix may not work upfront. A fresh DLPrw matrix of reference relative weights has to be created for the application on the new platform for the matrix to be applied to the Delay Point latencies to project the operation's Admittance and in turn Performance. Only *formal* application operation consumption contracts are in scope of this thesis with dedicated, controlled network traffic and *not* any random application access over public network. At *runtime*, no unpredictable fluctuation of network bandwidth or latency is assumed. If this assumption is violated, network latency needs to be included in the list of predictor variables.

Understanding and knowledge of the Delay Points is crucial for the accuracy of the results and coverage of Delay Points needs to be complete. Probing the Delay Points accurately is critical to the precision of the performance analysis. Incomplete coverage of the Delay Points may lead to incorrect results, analysis and forecast. In our case, the prototypical application framework was designed and developed by us, which ensured full coverage of the Delay Points. The requirement of having knowledge of the application codebase by the target users may be viewed as a constraint on the proposed methodology.

Fishing and the Error Rate – In our experiments, the processing intensities of the various Delay Points are varied in different steps and the corresponding operation Admittance is observed and recorded with the aim of extracting the relative weights of the Delay Point latencies. The statistical powers of our experiments are demonstrated to be high, highlighting the ability of the experiments to reveal the true patterns in the collated data.

Reliability of Treatment Implementations – The fully automated nature of our experiments with only the instrumentation towards probing the Delay Point latencies and the operation's response times performed manually, mitigates the risk of any difference in type of measurements across different test runs. As the experiments do not involve any human participants, the risk of dissimilar implementations between different persons applying the treatment is eliminated too.

Random Irrelevancies in Experimental Setting – Our experiments are conducted in a controlled environment. Several boundary conditions related to the QN model, application hosting environment, network provisioning and others are observed during the experiments. This reduced the possibility of any external impact on the experiments to a great extent.

Random Heterogeneity of Subjects - Partitioning the workload and classifying the request types increases the predictive power of the model. To classify the request type, we restrict our model and methods to operations of an application. At a given time T_1, for a particular type of transaction request to the same application operation, the request type, the process logic that is followed to serve the transaction, the system configuration, the resource requirements and the contract request load condition will be ideally the same. Hence, the computed DLPrw matrix, containing the relative weights of the Delay Point latencies, is applicable to that particular application operation being requested and associated to a single class of workload. The same DLPrw matrix may not be valid for other operations of the same application.

7.6.2 Internal Validity

In all of our experiments, we had specific independent variables and a group of dependent variables, which eliminated the *Single Group Threat*. As explained in Section 3.3.1, the experimental environment is maintained constant thus mitigating the risk that the *history* will affect the experimental results due to changes in the circumstances. The experiments are performed in a fairly controlled environment without any unexpected or unforeseen factors influencing the resultant data and do not involve any human participants. Hence, effects of motivation, maturation and other such factors do not apply. All of these factors ensured that the *Internal Validity* of our experiments is maintained.

7.6.3 Construct Validity

Design Threats - The application operation provider – consumer prototypical framework comprising a multi-threaded operation consumer, a consumer facing web service acting as the operation provider, other backend application components and some other utility components is built

to resemble a real-life scenario. For the purpose of the experiments, some illustrative application component level Delay Points with activities such as Database Interactions, In-Memory Data Processing, File I/O, Request Authentication and Request Authorization involving XML parsing etc. are also created. While this framework covers a wide range of different types of components supporting an application operation and we have reasons to assume that it simulates many real-world frameworks, it is practically not possible to incorporate all the various types of component complexities in the experimental framework. In this thesis, instrumentation for probing the Delay Point latencies and operation response times has been done manually. As future extension, the instrumentation may be automated through the use of Aspect Oriented Language. However, implementation of the Delay Points of the same type may vary across different real-world applications. This may potentially lead to difficulty in identifying the Delay Points to insert the probes automatically. Tool support will expedite instrumentation otherwise manual insertion of the Delay Point probes may be time consuming. Single core and dual core machines have been used during the experiments. Under certain specified workload conditions, the patterns of variation of the Admittance and Performance of an application operation due to changes to the supporting Delay Points are extracted. However, nowadays, extreme high spec machines with numerous cores are available, which are typically deployed in the production environments. The workload injected during our experiments may not be sufficient to impact the Performance on those types of high spec hosting environments to the extent that patterns of variation can be extracted. More workload may be needed to be injected.
While we have tested various treatment combinations during the experiments, it is possible that we have not explored all the possible combinations of treatments in our experiments. There are many more types of treatments that can be applied to observe the effects. However, in the *Future Work* section (9.4), we have briefly discussed some of the possible further experiments that may be undertaken in the future.

To establish a method to analyse and predict the impact on an application operation’s performance due to simultaneous changes to Delay Points of multiple types across the supporting application components, we treated Delay Points of various types at the same time. A matrix based technique is explored for the assessment of impact of the simultaneous changes on the Admittance and Performance of application operations. This technique applied a heuristic method for estimating the relative weights of predictor variables in multiple regressions. While this formed part of the experiments, the threats of *Interaction of different Treatments* and *Interaction of Testing and Treatment* cannot be fully discarded.

Social Threats – As our experiments do not involve any human participants the *social threats* as explained in Section 5.4.3 are mitigated to a great extent.

7.6.4 External Validity

Interaction of Selection and Treatment – In our experiments we have designed and developed various types of application processing activities, which are commonly found in real life industrial applications. For each application configuration with Delay Point(s) processing intensity set to particular levels, 10 test runs are executed with each test run comprising 30 transaction requests of the same class of workload and representative of real-life scenarios. Data from 5 of the test runs with the highest ARTs are recorded. 5 different application configurations are tested. Hence, to ascertain the patterns and trends for each of the different types of Delay Points, *1500* sets of responses and probed data are observed. This implies good data coverage representative of real-life scenarios. However, even though we have reasons to assume that the experiments covered reasonably good amount of data and types of Delay Points, there still may be other types of Delay Points, which we have not been able to cover within the scope of this thesis.

Interaction of Setting and Treatment – Due to the reasons as explained in Section 6.4.4, it is likely that the performance results on a non-production environment will highlight a “worse-case scenario” under a given load condition for an application operation. Hence, if not better, the production environment’s performance is expected to be at the least the same as the performance observed on the non-production environments for any given workload condition.

Interaction of History and Treatment – Our experiments are fully automated with only the instrumentation towards probing the Delay Point latencies performed manually for the time being. The instrumentation is done once upfront without any further alteration during the course of the experiments, eliminating the risks of any difference in type of measurements across different test runs over time. Also, the experiments do not involve any human participants. Hence the risk of dissimilar implementations towards the treatment over time is eliminated too.

7.6.5 Repeatability

Due to the reasons as explained in Sections 5.4.5 and 6.4.5, replication of our experiments should be possible with different sets of configurations.

7.7 Chapter Summary

In this chapter we proved the third null hypothesis (**$H3_0$**) to be false and evaluated the third hypothesis. The intrinsic and extrinsic goals of the evaluation as described at the beginning of this chapter are achieved. Towards the former one, the evaluation outcome demonstrated that the approach can be used by the software developers without the need of additional quality assurance experts. The software developers will neither be required to learn new, non-standard modelling formalisms and notations nor to specify the software system using process algebras or complex SPE techniques. The approach will not need repetitive performance testing or resource intensive application profilers to assess the impact of the changes to an evolving application and will yield precise results. For the extrinsic goal, it is established that under a given workload condition and hosting environment, the degree of impact of the simultaneous changes to the Delay Points of multiple types, on an operation's Admittance ('Y'), is determined by the implicit relative weights associated with the total cumulative latency of the Delay Points of each type. Determining the *best-fit* relative weights will facilitate predicting the operation's 'Y' for any future set of latencies of the Delay Points.

(Johnson, 2000) proposed a very effective technique to predict the *best-fit* weights of predictor variables in multiple regression by applying a heuristic method for estimating the relative weights. This technique proposed by Johnson is adopted to evaluate the third hypothesis. It is used to determine the *best-fit* relative weights associated with the latencies of each type of Delay Points, which determine the proportionate contribution of those respective latencies to the overall operation Admittance. A matrix based predictive method is established to forecast an application operation's Admittance and thus Performance for simultaneous changes to multiple types of supporting application component activities. The key to this matrix based method is the single column best-fit "Delay Point relative weight" (DLPrw) matrix. Applying the technique proposed by Johnson, this matrix is computed initially from the measured actual Delay Point latencies and the corresponding overall application operation Admittances. The data in this DLPrw matrix represents the best-fit values of the relative weights, which determine the proportionate contribution of those respective Delay Point latencies to the overall operation Admittance. The set of relative weights (i.e. the DLPrw matrix data) is then used to predict the operation Admittance for any future set of latency values for the different Delay Points.

Precision of the DLPrw matrix data will determine the accuracy of the prediction of the impact of simultaneous changes to the various types of Delay Points. For evaluation of the hypothesis, we used the prototype application framework to run experiments and through the heuristic technique advocated by (Johnson, 2000) computed the DLPrw matrix. The precision of the DLPrw matrix is then validated. The percentage error between the operation Admittance computed from the measured actual experimental data and the operation Admittance value projected through the DLPrw matrix is calculated to be 4.459%.

During evaluation, the algorithm configurations of the various Delay Points are varied in uniform steps, which resulted in a more linear regression for the application operation Admittance. In real life, the Delay Point latency variations across the components may not be uniform. Hence the matrix based method need wider coverage of Delay Point latency variations of the components for improved precision of forecasts. The Delay Points in the application framework are subjected to various algorithm configurations at random. This time the number of iterations is increased heuristically than the number of iterations conducted while evaluating the third hypothesis. Following the same steps as before, the method subsequently yielded an error of 3.39%. The statistical power (Wohlin et al., 2012) of the experiments is also derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data. An assessment of the threats to the validity of the experimental findings has also been presented in this chapter.

8 Related Work

In Chapter 2 (*Background and Motivation*), we covered a broad range of past and current research work in the application performance analysis and prediction space. In this chapter we present a more focused discussion of the past and present related work and approaches proposed to address application performance analysis. The approach and findings of this thesis are compared to those related work.

8.1 Comparison of proposed Approach to Related Work

The SPE methodology (Smith, 1990; Smith et al., 2002) was the first comprehensive approach to deal with performance analysis during the software development process. The methodology uses a combination of software execution models and system execution models. The first one represents software execution behaviour in the form of Execution Graphs (EG). The second one is based on Queuing Network models and represents the system platform including hardware and software components. However, despite all its merits, applying SPE techniques in practice is still very challenging (Happe et al., 2010). The SPE analytical models can usually be built only by imposing some structural restrictions on the original system model, depending on the specific modelling formalism as analytical models have often a limited expressiveness. There are many cases in which the significant aspects of the system cannot be effectively represented into the performance model. The SPE techniques are not widespread since they require the software engineers to learn new and non-standard modelling formalisms and notations (Marzolla, 2004). In SPE integration between the early calculations (e.g. by models) and the later measurements is elusive. SPE is constrained by the tight project schedules, poorly defined requirements and over-optimism about meeting those (Woodside et al., 2007). The methodology advocated in this thesis involves matrix computation, which can be achieved either by using readily available online matrix calculators (as done in this thesis) or programmatically. But it does not warrant other complex performance models or requires the software engineers to learn new, non-standard modelling formalisms and notations. The software engineers are not required to specify the software system using process algebras and complex mathematical analysis.

Several Stochastic extensions of Process Algebras (SPA) (Hermanns et al., 2002) associate exponentially distributed random variables to actions and provide the generation of a Markov chain out of the semantic model of a system. PEPA nets – coloured stochastic Petri nets, which is a modelling formalism to clearly capture important features such as location, synchronisation and message passing and a platform support for software performance modelling using the PEPA nets is described in (Gilmore et al., 2004). The drawback of many effective state-of-the-art performance analysis tools which do not differentiate between simple local communication and the migration of processes which may change the allowable pattern of communication is addressed by PEPA nets modelling language (Gilmore et al., 2003). PEPA nets extend the PEPA stochastic process algebra (Hillston, 1996) by allowing PEPA process algebra components to be used as the tokens of a coloured stochastic Petri net. Work has been done to introduce stochastic probes as a means of measuring the soft performance characteristics over software systems (Argent-Katwala et al., 2004). A regular expression language is presented, which specifies the stochastic probe and is then itself converted into a stochastic process algebra component. This is combined with the original SPA model (PEPA used) and analysed to provide soft performance and reliability bounds. This convenient partition allows the three types of soft performance analysis – transient, passage-time and steady-state to be expressed in a unified manner. As an extension of this work, a functional performance specification language (FPS) is developed (Bradley et al., 2006). It allows the modeller to derive quantitative performance functions using stochastic probes. In the context of PEPA, a mechanism is defined for specifying performance queries which combine instantaneous observations of model states and finite sequences of observations of model activities (Clark et al., 2008). These queries are realised by composing the state-aware observers called eXtended Stochastic Probes (XSP) with a model expressed in stochastically-timed process algebra. In (Gilmore et al., 2005), a Multi-Terminal Binary Decision Diagram (MTBDD) based PRISM probabilistic model checker is used to represent the underlying performance model for a high level UML design. A component-based method of linking a collection of software tools is used to facilitate automated processing of UML performance models. One significant practical problem with this approach is that an inexperienced modeller will not be able to use the system to compute any performance measure that they wish without any understanding of the abstraction, modelling and mathematical analysis at work in performance prediction and estimation (Gilmore et al., 2005). Stochastic process algebra based techniques require understanding of complex modelling and mathematical analysis often introducing notational hurdles. This acts as an impediment to the understanding and uptake of modern performance analysis technologies based on stochastic process algebra. This thesis addresses the very crucial issue of software engineers and developers requiring understanding of abstraction, complex modelling, mathematical analysis and unnecessary notational languages for performance evaluation. Using the *PALP* model, the lookup reference datasets are created upfront for the application operation from the initial test runs. During modifications to the application operation, these datasets are subsequently consulted during systems testing by the developers themselves to analyse the possible impact of the changes to the operation's performance. For simultaneous changes to multiple types of Delay Points, the derivation of the DLPrw matrix requires simple matrix formula to be computed as described in Equation (7.2). Although this thesis has used online matrix calculator, this formula can be automated. Hence the developers may not even require any knowledge of matrix calculations.

A tool called ArgoSPE is introduced in (Gomez-Martinez et al., 2006), which translates some performance annotated UML diagrams into SPN models and avoids modelling with SPN since they are obtained as a by-product of their UML models. A formal mapping approach from PCM to QPN models implemented by means of an automated model-to-model transformation as part of a new PCM solution method based on simulation of QPNs is presented in ((Meier et al., 2011). An automatic transformation from PCM to QPNs in the form of a new PCM solver tool is implemented. However, Petri nets do not alleviate the problem with dealing with complex notational hurdles. Petri Net diagrams are complex, contain more nodes and edges than UML 2 activities and are unsuitable for visualization by stakeholders (Staines, 2008; Staines, 2010).

An automatic generation of simulation performance models from high-level UML descriptions of SA is presented in (Marzolla, 2004). The approach considers UML diagrams annotated with a subset of the UML Performance Profile (Object Management Group, 2002). The structure of the simulation model is very similar to the structure of the software model. The drawback of this approach is its dependency on UML modelling. UML is only informally defined, so that software modellers may use different diagrams for the same purpose, or use the same notation with different implicit meaning (Marzolla, 2004). Also, approaches which try to apply performance analysis very early, typically at the software architecture and design level still require much detailed information from the implementation phase to carry out performance analysis (Balsamo et al, 2004). This thesis attempts to address this issue and focuses on the implementation phase directly. The measured data reflects the realistic view of the application running on a specific hosting environment.

Different types of QNs such as Single-Class Open QNs, Multiple-Class Open QNs, Single-Class Closed QNs and Multiple-Class Closed QNs have been discussed in (Menasce et al., 2002). Significant work has been undertaken on various methodologies which propose transformation techniques to derive QN based models like Extended QN (EQN) or Layered QN (LQN) from Software Architecture

(SA) specifications (Kant, 1992; Franks et al., 1995; Rolia et al., 1995; Woodside et al., 1995; Trivedi, 2001; Mania et al., 2002; Petriu et al., 2004). However, in QN models it is difficult to handle situations arising from finite capacity of queues and subsequent blocking behaviour. Only approximate techniques can be used in some cases and simulation is the only approach in general (Balsamo et al., 2003). Other difficulties arise when analyzing simultaneous resource possession, fork and join systems, synchronous versus asynchronous communications and many queuing disciplines.

An integrated coverage of performance modelling, workload forecasting, load testing and benchmarking of web applications has been carried out (Menasce et al., 2002). The perception of performance from web infrastructure, server architecture and network point of view has been explained. To achieve precision of performance analysis and prediction, steps are proposed to characterize and partition the workload into a series of classes such that their populations are formed by quite homogeneous components. The work of Menasce et al. forms some of the basis of the *PALP* model of this thesis. However, their work does not delve into the application layer and how changes to an existing application impact the performance. The system level measured data is obtained from the hosting platform for which we may need additional performance experts as the application developers/software engineers may not have the requisite expertise (Marz, 2005). With the approach to automatically improve software architectures with respect to performance, reliability and cost as presented in (Martens et al., 2010), the design space spanned by different design options (e.g. available components and configuration options) can be systematically explored using meta-heuristic search techniques. Based on an initial architectural model of a system, new candidates are automatically generated and evaluated for the quality criteria. However, the approach is complex, time consuming and doesn't guarantee globally optimal solution. For the results, uncertainty of estimations, uncertainty of the workload and the resulting risks are not taken into account.

Presently, the tools used by performance analysts range from load generators for supplying the workload to a system under test, to monitors for gathering data as the system executes (Woodside et al., 2007). Instrumentation is either built into the host operating system and minimally indicates the utilization of the various devices including the CPU or probes are added manually to applications. Frameworks such as the Application Response Measurement (ARM) (Johnson, M. W., 1998) are beneficial as they form a common platform to which disparate programs can gather performance information. Instrumentation can be added automatically as well. Aspect-Oriented programming can be used to automatically insert instrumentation code into applications (Debusmann et al., 2003).

Quantify (IBM, 2002) adds probes at the beginning and end of basic blocks of object code to count the number of cycles executed. The Paradyn tool (Merson et al., 2005) carries this one step further by instrumenting the actual executables dynamically. Profiling can be achieved using instrumentation through statistical sampling or by running the program on a virtual machine and counting the execution of the actual instructions (Nethercote et al., 2003). However, a key problem with these performance tools is that these are not well established at earlier stages in the software life cycle. Also, various tools have various forms of output which makes interoperability quite challenging and the measurement data collected requires expert interpretation to fix the problems (Malony et al., 2001).

Profiling and monitoring tools deeply affect performance (Duggan et al., 2011). Profilers add overhead to the executing application it is measuring and to the machine it is running on. The amount of overhead depends on the type of the profiler. In the case of a performance profiler, the act of measurement may itself impact the performance being measured. This is particularly true for an instrumentation profiler, which has to modify the application binary to insert its own timing probes to every function. As a result there is more code to execute, requiring additional CPU and memory, causing increased overhead. The profiler also has to deal with lots of data it and for a detailed analysis it may require a lot of memory and processor time just to cope. If the application is already memory and CPU intensive, things will only get worse and it could be that it is just not possible to analyse the application properly (Farrell, 2010). Also, there are no specific rules to interpret the data and users depend on experience to use the results. As highlighted in (Woodside et al., 2007), better methods and tools are a future requirement for interpreting the results and diagnosing performance problems. The approach proposed in this thesis circumvents the additional overhead issues both from memory and CPU perspectives. Through probing of Delay Point response times (either a particular type or multiple types depending on the type of change) the variation in the operation’s Admittance and Performance is detected, which do not add much processing or memory overheads. This thesis also attempts to alleviate the problem of leaving the collected measurement data to subsequent expert interpretation by attempting to formulate an approach applicable during the implementation phase of the development life cycle.

Performance profiling through synthetic workloads or benchmarks is useful at the initial stages of design and development of a future system, but it may not be adequate for analysis of performance issues and observed application behaviour in existing production systems (Cherkasova et al., 2007). Frequent software releases and application updates make it extremely difficult and challenging to perform a thorough and detailed performance evaluation of an updated application. The traditional *reactive* approach to performance analysis is to set thresholds for observed performance metrics and raise alarms when these thresholds are violated. It is acknowledged that this approach is not adequate for understanding the performance changes between application updates. Cherkasova et al suggests a *proactive* approach that is based on upfront continuous application performance evaluation may assist enterprises in reducing loss of productivity by time-consuming diagnosis of essential changes in application performance. The approach advocated in this thesis will facilitate proactive fine-tuning of the application code and configuration by the software developers upfront during local systems testing phase to attain the prescribed performance level instead of recourse to a repetitive and *reactive* approach of setting thresholds for observed performance metrics and raising alarms when these thresholds are violated

CPM implements performance and scalability testing within a CI environment. Automated load tests are run within the CI test harness to baseline and track the application's scalability during development (Haines, 2008). JUnit (JUnit 4, n.d.) covers component level unit test analysis. For load testing, HttpUnit extension of JUnit is typically used. It requires a different test bed because these are more like business use cases rather than functional tests. These types of tests do not provide insight into how the internal processing activities impact the timeliness of responses of the components or operations supported by multiple components. To achieve this and take fullest advantage of CPM, scriptable performance analysis tools like code profilers and memory debuggers are needed (Haines, 2008). That is, an engine that will run the performance, integration, scalability tests and capture the results is needed. But these will have significant limitations as profiling and monitoring tools deeply affect performance (Duggan et al., 2011). The performance of a memory and CPU intensive application will only get worse and it may not be possible to analyse the application properly (Farrell, 2010). To test performance of functional units of code, tools like JUnitPerf are used. These tests verify the performance of the functional units as a black box. The drawback is if we execute performance tests against lower level units of code, it will only measure the performances of the different units of code but may not indicate the overall performance of a higher level operation or transaction being catered by the underlying units of code. Another drawback of CPM as it is practised currently is that although it is done on a continuous basis, it follows the traditional *reactive* approach to performance analysis. Load tests are performed after every build and alarms are raised when set thresholds for observed performance metrics are violated. Remedial measures are then adopted and performance is re-tested. This cycle continues till the set thresholds are met. As highlighted by Cherkasova et al., this approach is not adequate for understanding the performance changes between application updates. Due to this inherent nature of CPM, it is not possible to *target* a specific performance level and fine tune the application code proactively during application code development and modification to achieve the target performance level. Using the novel concepts of operation Admittance and Load Potential, this thesis will enable proactive fine-tuning of the application code and configuration by the software developers upfront during code development and modification (local systems testing) to attain the prescribed performance level instead of recourse to the reactive CPM approach.

Cherkasova et al. presented a novel framework for automated anomaly detection and application change analysis. It is based on a regression-based transaction model that reflects a resource consumption model of the application and an application performance signature that provides a compact model of run-time behaviour of the application. Through the resource consumption model, the framework detects anomaly in CPU utilization across different transactions. The application performance signature facilitates identification of unusual variations to transaction service times with changes to the application. However, the focus of their work is on server level (App Server and DB server) transaction latencies and identifying unusual variations to overall transaction service times with changes to the application. It does not delve into the details of how the various lower level transaction processing activities impact the service time of the transaction. Without this knowledge, it is difficult to fine tune an application's algorithm and configuration to meet any prescribed performance criteria.

A new generation of monitoring tools, both commercial and research prototypes, provide useful insights into transaction activity tracking and latency breakdown across different components in multi-tier systems. Some of them measure end-to-end latencies observed by clients (Rajamony et al., 2001). Typically they provide a latency breakdown into network and server related portions. While these tools are useful for understanding the client network related latencies and improving overall client experience, this approach does not offer sufficient insight in the server-side latency as it does not provide a latency breakdown into application and database related portions (Cherkasova et al., 2007). Some other tools focus on measuring server-side latencies using different levels of transaction tracking, which are useful for drill-down performance analysis and modelling (Barham et al., 2004; Quest Software Inc.). Unfortunately, such monitoring tools typically report the measured transaction latency and provide additional information on application versus database server latency breakdown. Using this it is often difficult to ascertain whether an increased latency is a result of higher load in the system or whether it is an outcome of the recent application modifications (Cherkasova et al., 2007).

The approach proposed in this thesis enables detection of the possible impact on performance of an application operation under a given workload condition due to changes to the lower level transaction processing activities. Instead of measuring the application latency as a whole, the approach enables identifying the patterns in which the granular Delay Points of a particular type or multiple types when modified simultaneously impact a transaction's Performance.

Run-time monitoring approaches such as (Westermann et al., 2010) use systematic measurements to build mathematical models or models obtained with genetic optimization. The models serve as interpolation of the measurements. The main idea of the approach is to abstract from system internals by applying a combination of systematic goal-oriented measurements, statistical model inference and model integration. The measured functional dependencies are integrated in the PCM. This approach intrinsically has a few drawbacks. Intuitive interpretation of the measure values is applied. If there do not exist more information about how the numbers are produced, it is difficult to interpret them. Behind every software measure (resource, process or product measures), a qualitative model is hidden (Zuse, 1998). Doing software measurement we have to quantify qualitative properties of objects in reality. In a software application it can be quite challenging to ascertain how the various processing activities are contributing to the measured systematic data. It takes a lot of effort, time and most importantly the right technical insight to identify these hidden attributes. The other significant limitation of this approach is that it integrates into the PCM. Depending on the approach, it may not follow an industry standard, therefore widespread use may be a long term goal. Existing UML tools cannot be used to create PCM instances. The learning curve for developers familiar with UML is potentially higher. Existing UML models are not supported by the PCM analysis tools and have to be transformed to PCM instances to carry out performance predictions. Implementing such transformations is complicated (Becker et al., 2009). The approach proposed in this thesis provides insight into how each of the various transaction processing activities impacts the application operation's performance. It is easy to interpret the results and proactively fine tune the processing activities accordingly without recourse to build, load test, performance measurement and code change cycles. Software engineers will not need any learning curve as no non-standard modelling formalism is required.

8.2 Chapter Summary

In this chapter we have brought together the various approaches explored previously, which are related to the performance analysis space that this thesis focuses on. The previous techniques and approaches are discussed in light of the methodology proposed by this thesis. Some of the limitations and shortcomings of the previous approaches and how the methodology proposed in this thesis attempts to address those have been discussed.

The SPE methodology and how it combines the software and system execution models to comprehensively deal with performance analysis during the software development process has been discussed. Despite all the features, the aspects which attribute to the severe challenges in applying and adopting the SPE techniques in practice have been reviewed. It is acknowledged that the SPE techniques are not widespread since they require the software engineers to learn new and non-standard modelling formalisms and notations. Several SPAs and PEPA nets - coloured stochastic Petri nets have been discussed. It is discussed how PEPA nets extend the PEPA stochastic process algebra. A functional performance specification language (FPS) alongwith the concept of XSP is explored with a model expressed in stochastically-timed process algebra. A probabilistic model checker to represent the underlying performance model for a high level UML design is discussed. While the benefits of PEPA nets have been reviewed, the significant practical problems associated with SPA and PEPA based techniques have been discussed as well. An inexperienced modeller will not be able to use the system to compute any performance measure without the understanding of the abstraction, complex modelling and mathematical analysis at work in performance prediction and estimation. SPA based techniques often introduce notational hurdles, which act as an *impediment* to the uptake of performance analysis based on SPA. This thesis's approach to address the issue of software engineers requiring understanding of abstraction, complex modelling, mathematical analysis and unnecessary notational languages for performance evaluation is reviewed.

The benefits of SPN based tools are discussed alongwith the complexities of Petri nets and how those do not alleviate the key issue of dealing with notational hurdles. The drawback of simulation performance models from high-level UML descriptions of SA is discussed. It is reviewed how the need of detailed information from the implementation phase to carry out performance analysis is addressed in this thesis by measuring data reflecting the realistic view of the application running on a specific hosting environment. The benefits and limitations of the different types of QNs have been discussed in this chapter. An integrated coverage of performance modelling, workload forecasting, load testing and benchmarking of web applications has been reviewed. To achieve precision of performance analysis and prediction, steps have been analysed to characterize and partition the workload into a series of classes such that their populations are formed by quite homogeneous components. The work of Menasce et al., which forms some of the basis of the *PALP* model has been discussed.

It is reviewed how all the benefits of profiling and monitoring tools come at a price. Most of these tools are not well established at earlier stages in the software life cycle and their output varies in form, which makes interoperability quite challenging and the measurement data collected requires expert interpretation to fix the problems. The various ways in which profiling and monitoring tools adversely impact performance by adding system overheads (CPU and memory) are reviewed. Also, there are no specific rules to interpret the data and users depend on experience to use the results. It is highlighted that better methods and tools are a future requirement for interpreting the results and diagnosing performance problems. The ways in which this thesis attempts to address these issues through the concept of Admittance and measuring Delay Point responsiveness have been discussed.

Frequent software releases and application updates make it extremely difficult and challenging to perform a thorough and detailed performance evaluation of an updated application. Upfront application performance evaluation may assist enterprises in reducing productivity loss in time-consuming diagnosis of changes in application performance The different types of tests like JUnit, JUnitPerf and load tests during CPM, their purposes, benefits and the gaps that currently exist for a proactive approach towards performance analysis have been discussed. The problems of using code profilers and memory debuggers and how these may worsen an application's performance have been reviewed. The inability of tests like JUnitPerf to assess the overall performance of higher level operations supported by various underlying units of code is highlighted. The drawback of CPM owing to the traditional *reactive* approach to performance analysis and its inability to *target* a specific performance level and fine tune the application code proactively during code development and modification to achieve the target is explained. This chapter discusses how this thesis will facilitate *proactive* fine-tuning of the application code and configuration by the software developers during local systems testing phase to attain the prescribed performance level instead of recourse to a repetitive and *reactive* approach of load testing and raising alarms when set thresholds are violated.

Limitations of the novel framework for automated anomaly detection and application change analysis presented by Cherkasova et al is discussed. The drawbacks of a new generation of monitoring tools of not being able to offer sufficient insight in the server-side latency are discussed. Using these tools it is often difficult to ascertain whether an increased latency is a result of higher load in the system or whether it is an outcome of the recent application modifications. The issues with the run-time monitoring approach of Westermann et al. using systematic measurements to build mathematical models and integrating the measured functional dependencies with the PCM intrinsically are reviewed.

9 Conclusion

In this chapter we summarise the context and motivation of this thesis, the work undertaken, the results obtained and this thesis's overall contribution to the advancement of the body of knowledge on performance engineering. We have presented an evaluation of the proposed approach against the research objectives. We have also discussed how the methodology proposed in this thesis shall complement the cloud paradigm and elasticity. Finally, we have discussed some of the potential work that may be carried out in the future to augment the precision of the proposed methodology and address real life production systems.

9.1 Thesis Summary

9.1.1 Context & Motivation

Significant research involving various modelling approaches including SPE techniques, QNs and their various extensions, SPA and SPNs have been undertaken towards performance analysis of systems. Currently with CPM, automated performance tests are embedded in the CI process. Performance is assessed by either invoking automated load tests on the high level business use cases or measuring the performance of the functional units of code during the CI process. Framework such as JUnit is used for unit tests of the code developed and implemented. Integration and load testing are more like business use cases rather than functional tests and HttpUnit extension of JUnit is typically used here. Tools like JUnitPerf are used to verify performance of functional units of code during the CI process.

The various approaches mentioned above all have their respective drawbacks. Software performance modelling is challenging for different reasons. Static analysis of code doesn't yield meaningful performance measures. Process Algebra based approaches have their own problems. From the performance evaluation viewpoint, the analysis usually refers to the numerical solution of the underlying Markov chain which can easily lead to numerical problems due to state space explosion. The software designer is required to be able to specify the software system using process algebras and to associate the appropriate performance parameters to actions, which may not possible all the times. Applying SPE techniques in practice is still very challenging. The SPE analytical models can usually be built only by imposing some structural restrictions on the original system model. There are many cases in which the significant aspects of the system cannot be effectively represented into the performance model. The SPE techniques are not widespread since they require the software engineers to learn new and non-standard modelling formalisms and notations. It is an acknowledged fact that the lack of performance requirement validation in current software practice is mostly due to the knowledge gap between software engineers/architects and quality assurance experts. Also, most of these approaches try to apply performance analysis very early, typically at the software architecture and design level and hence still require much detailed information from the implementation phase to carry out performance analysis. Besides these, frequent software releases and application updates make it extremely difficult and challenging to perform a thorough and detailed performance evaluation of an updated application. Hence, for software systems which are evolving continuously due to changes to business requirements, these techniques are not effective from resource, time and cost perspective.

CPM is reactive and does not help to understand how the different granular low level application processing activities impact the performance of the operation. If profilers and monitoring tools are used during CPM, it adds CPU and memory overheads to the executing application it is measuring and to the machine it is running on. If the application is already memory and CPU intensive, things will only get worse and it could be that it is just not possible to analyse the application properly. The JUnitPerf decorators introduce significant overheads. The elapsed time measured is not reflective of the actual elapsed time of the method. These tests are also reactive in nature as these set observed performance thresholds and flag alarms when these thresholds are violated. These are also not intended to be a full-fledged load testing or performance profiling tool. This testing neither helps to understand how each of the low level granular application processing activities impacts the overall performance of the higher level operation nor does it facilitate proactive fine-tuning of the various application processing activities during code development and modification to achieve a pre-specified target performance level.

Software performance engineering in its essence is not directly applicable to the continuously evolving software applications due to changing market requirements and short innovation cycles. Many approaches focus on early lifecycle phases assuming that a green-field scenario and all the details are known. The problem is that detailed information about the internal structure, which is required for performance prediction, may not be available during the early phases. Many approaches have been published in the context of software performance engineering but none has achieved widespread industrial use due to the above mentioned issues.

Software developers work at the application layer being developed and not at the underlying systems layer. They develop code and often recourse to manual ways to measure the performance of their code. At times, they do not even have the requisite expertise to monitor systems resource utilization. Hence, monitoring systems resource utilization may not help application developers. Typically, system administrators monitor systems level resource utilization.

The drawbacks of the various performance assessment techniques highlight the need for a pragmatic approach to software performance analysis and forecast for *existing* software applications undergoing frequent software releases and updates. The approach shall not require repetitive load/performance testing, complex SPE techniques, performance models and resource intensive application profilers. The software engineers shall not be required to learn new, non-standard modelling formalisms and notations or to specify the software system using process algebras.

This thesis sets out to answer the above question affirmatively. It attempts to establish an approach that will address the above mentioned constraints and facilitate proactive fine-tuning of the application code and configuration by the software developers upfront during code development and modification to attain a prescribed performance level instead of recourse to a repetitive and *reactive* approach of setting thresholds for observed performance metrics and raising alarms when these thresholds are violated.

9.1.2 Methodology and Hypotheses

While adopting the approach advocated by (Westermann et al., 2010) and (Cherkasova et al., 2007), this thesis aligns with the future direction as proposed by (Woodside et al., 2007). The thesis augments the work done in (Cherkasova et al., 2007), which adopts a very linear, sequential approach to compute the overall latency of a transaction relating to the transaction's performance. This thesis focuses on the application layer of the system and proposes a combination of model based and measurement based performance analysis approach. While getting into the application algorithm and configuration i.e. a "white-box" approach (TestPlant, 2011) to the application layer, the thesis uses "black-box" performance models, which do not contain any information about the underlying system's internal structure. The performance of the application operation is captured by a function of its usage. The reason for adopting this approach is manifold. Changes in business requirements warrant the application components to be modified. However, typically the system layers underneath the application comprising the operating system, hardware infrastructure and network remain unchanged. Technology refresh for IT systems, which are typically undertaken every 3-5 years (Archstone Consulting, 2013) are outside of the scope of this thesis.

This thesis introduces three new concepts in the context of an application operation namely Admittance, Load Potential and Delay Point. Chapter 3 discusses the rationale behind the performance analysis methodology proposed in the thesis. It then states the three hypotheses which form the core of this thesis. The hypotheses are then evaluated in the subsequent chapters. The *first hypothesis* is stated in the context of the relational model between an operation's *P*erformance, *A*dmittance and *L*oad *P*otential. It is referred to as the *PALP* model. The hypothesis states that, for a given class of workload and a hosting environment, an application operation's Performance is directly proportional to its Load Potential at runtime and the proportionality constant is the runtime Admittance of that operation. This is for a given range of the Load Potential. All the other boundary assumptions for the applicability of the hypothesis are discussed in this context and the constancy of runtime operation Admittance is explained and justified.

The s*econd hypothesis* is formulated in the context of the impact on operation Admittance for changes to Delay Points of one particular type. The hypothesis states that under a given workload condition and hosting environment, the operation Admittance can be expressed as a *function* of the total delay or latency of the Delay Points of a particular type across all the supporting application components catering to the operation. Determining the *best-fit* function will facilitate predicting the operation Admittance and hence the operation Performance for any future value of the cumulative latencies of the Delay Points of a particular type.

In real life production systems, it is unlikely that Delay Points of only one particular type will be modified at a time due to changes in business requirements. In view of this, the *third hypothesis* of this thesis is formulated in the context of the impact on operation Admittance for simultaneous changes to Delay Points of multiple types. The hypothesis states that under a given workload condition and hosting environment, in the event of simultaneous changes to Delay Points of multiple types, each type has an implicit relative weight associated to it, which determines the degree of its impact on the operation Admittance. Determining the *best-fit* relative weights will facilitate predicting the operation Admittance and hence the operation Performance for any future set of latency values of the Delay

Points. This thesis adopts the technique proposed by (Johnson, 2000) to evaluate the *best-fit* relative weights associated with the latencies of each type of Delay Points, which determine the proportionate contribution of those respective latencies to the overall operation Admittance. The set of relative weights is then used to predict the operation Admittance for any future set of latency values for the different Delay Points.

The hypotheses evaluation procedures suggested that the approach will involve manipulation of one or more variables and control of all the other variables at fixed levels for particular configurations. The effect of the manipulation is measured and statistical analysis is performed based on the measured values. Model based indicative values of some features are computed against variability of the Delay Point activities of the underlying application components supporting the application operation. Lookup datasets against different system configurations are created to associate these computed values to the actual measured values of other features and established mathematical techniques applied with appropriate statistical regression types to enable trend extrapolation and interpolation. The research method aligns itself with the "Controlled Experiment" empirical strategy as described by (Wohlin et al., 2012).

The proposed methodology comprising the *PALP* model and the associated methods is applied during the SDLC. The lookup reference data sets are initially created upfront for all the Delay Points supporting an application operation. During development, the developers are able to fine-tune the application code (Delay Points) to retrofit a specified performance level under a specified load condition by consulting the lookup reference data sets with systems testing instead of recourse to the cycles of Load Testing, Performance Measurement, Code Modification and Build during CI. In CPM, if there is any performance problem, load testing follows a repetitive and *reactive* approach of setting thresholds for observed performance metrics and raising alarms when these thresholds are violated. This thesis enables assessing the impact on performance due to modification of application code under a specified load condition by probing only the modified Delay Points. This can be achieved before the execution of the full build of the application and subsequent load testing, which has its own overhead from time, resource and cost perspective.

This thesis aims to allow the software developers to undertake performance analysis for modifications to the application layer. The software engineers who have knowledge of the application codebase and its behaviour are the target users of the proposed methodology. Applying the methodology warrants the Delay Points supporting the application operation of the existing application to be modified and their respective latencies probed through instrumentation to create the initial reference data sets. The thesis in its current form does not provide tool support for automated instrumentation and recourses to manual insertion of probes. There was always the possibility that modifying the application code might impact the behaviour of the operation in some way. Hence, knowledge of the application operation's codebase, its behaviour and implication of behavioural changes was required for the current version of this thesis. Due to these constraints, an application framework resembling a real life scenario of an application operation supported by several backend components and consumed by a multi-threaded client is designed and developed to conduct an *experimental* research for the evaluation purposes in this thesis.

9.1.3 Evaluation Experiments and Results

Details of the evaluation of the first, second and third hypotheses are presented in Chapters 5, 6 and 7 respectively. The goal of the first evaluation as presented in Chapter 5 is to validate the first hypothesis and establish the relational *PALP* model. Then, based on the deduction of constancy of 'Y' at runtime in Section 3.3.1, it attempts to demonstrate that 'P' may be expressed as a linear function of 'V' for a given range of 'V' (referred to as "V Bands" in Figure 17) and 'Y' is the gradient constant for that function of 'V' for a given class of workload and a given hosting environment including the network. In Chapters 6 and 7, we divide the evaluation of our approach into intrinsic and extrinsic ones. For both the evaluations, the intrinsic goal is to demonstrate that our approach can be used by the software developers without the need of additional quality assurance experts. The software developers will neither be required to learn new, non-standard modelling formalisms and notations nor to specify the software system using process algebras or complex SPE techniques. The approach will not need repetitive performance testing or resource intensive application profilers to assess the impact of the changes to an evolving application. It will yield precise and accurate results. The statistical power (Wohlin et al., 2012) of the forecasts shall be as high as possible. The method will be applied during the implementation phase or later to ensure more precise performance measures. However, the extrinsic parts of the evaluations are different. In Chapter 6, it is concerned with quantifying the effects of the changes to the Delay Points of a particular type across the application components catering to an operation and establishing the fact that under a given workload condition and hosting environment, the operation's Admittance ('Y') can be expressed as a statistical *function* of the total cumulative latency of the Delay Points of that type. This establishes the second working hypothesis of this thesis. In

Chapter 7, it is concerned with quantifying the effects of the changes to the Delay Points of multiple types across the application components catering to an operation and establishing the fact that under a given workload condition and hosting environment, the degree of impact of the total cumulative latency of the Delay Points of each type on the operation's Admittance ('Y') is determined by an implicit relative weight associated to the total cumulative latency of the Delay Points of that type. The *best-fit* relative weights will help in predicting the operation's 'Y' for any future set of latencies of the Delay Points. This establishes the third working hypothesis of this thesis.

For all the evaluations, we follow the guidelines on experimentation in software engineering as they are presented in (Wohlin et al., 2012). While keeping some subjects constant, different treatments are applied to other subjects and the effects on the outcome variables are measured. Controlled experiments are performed in a laboratory environment. When experimenting, the objective is to manipulate one or more variables and control all the other variables at fixed levels. The effect of the manipulation is measured and based on this measurement, statistical analysis is performed.

A prototype application framework, resembling our example TCG application, is developed to perform the experiments in a controlled environment. The overall framework comprises the application operation's provider sub-framework, consumer sub-framework and the application components forming the respective sub-frameworks. The operation provider framework exposes the application operation for consumption. It comprises a *facade* Orchestration Web Service, which orchestrates between the different lower level backend application components. The backend components perform different types of data operations. The Orchestration Web Service receives the transaction request, extracts the request data, calls the relevant backend application components, amalgamates the different responses from the backend components to create the response and sends the response back to the operation consumer, which is a web service client.

The operation consumer framework generates the transaction requests with appropriate request data and invokes the application operation on the provider's orchestration web service. It then receives the responses from the application operation and measures the response times. The measurements are recorded in the way as defined in an external configuration file. Based on parameters set on the external configuration file, the consumer framework is able to vary the inbound workload to the operation.

A call to the exposed operation triggers the operation's transaction process and the associated sequence of actions. The facade orchestration web service interacts with the various backend application components to cater to the operation's transaction. As explained in Chapter 3, certain environmental constraints are observed during the experiments. The same prototype application framework is run separately on two hosting environments to compare the results and observe the impact of augmented hardware on the various measured data under similar workload conditions.

Experiments are run on the prototype application framework to prove the first null hypothesis (**$H1_0$**) to be false and evaluate the first working hypothesis, which states that for a given class of workload, a given hosting environment including the connecting network between the application and the operation consumer and a given range of 'V', the application operation's 'P' can be expressed as a linear function of 'V' with 'Y' as the gradient constant. This is the *PALP* model. Previous work by Menasce et al. and Cherkasova et al. formed some of the basis of the key assumption that a particular application operation's admittance is constant at runtime under a given load band. Through controlled experiments on the application framework, it is proved that given certain constraints related to the inbound workload and the hosting environment, the runtime relationship between the three key attributes 'P', 'Y' and 'V' may be represented in the form: $P = YV + c$. Accepting approximation error, the non-linear function of 'P' against 'V' is simplified through the use of *Piecewise Linear Functions* by dividing the 'V' values into 3 ranges (or *bands*), each with a *linear regression* as the best fit for 'P'. It is also proved mathematically that an increase in the operation's 'Y' will boost its 'P' for any given value of 'V'. The statistical power (Wohlin et al., 2012) of the experiments is also derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data.

Through controlled experiments, the second hypothesis of the thesis is evaluated and the second null hypothesis (**$H2_0$**) is proved false and rejected. Over a number of iterations, the processing intensities of the Delay Points of one particular type are varied at a time in each iteration across all the components supporting the application operation. The impact of this variation of the processing intensities of the Delay Points of that particular type on the operation's Admittance and Performance is measured. From the experimentally measured and model based computed data, various types of statistical function graphs like the operation's 'Y' versus the variation of latencies introduced by a particular type of Delay Point, the operation's 'P' versus its 'Y' etc. are generated. The graphs produced are non-linear but consistently showed *distinct* trends in variation. The function graphs are verified against various statistical regression types (*Linear*, *Exponential, Polynomial* and *Power*) and accepting approximation error, the best-fit LSF is identified for the function. The statistical regression functions for every individual type of Delay Point were distinct. For every type of Delay Point, pattern integrity validation is performed subsequently to confirm the correctness of the extracted functions. The observed data sets affirmed the second hypothesis which states that under a given workload condition and hosting environment, the operation's 'Y' can be expressed as a statistical *function* of the total delay / latency introduced by the Delay Points of a particular type across all the supporting application components catering to the operation. The statistical power (Wohlin et al., 2012) of the experiments is derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data.

The application framework is implemented on two different hosting environments, one of which comprised hardware of higher specification than the other. The same experiments are run on both the environments. The graph functions derived from the two separate hosting environments revealed that the rate of decrease in the operation's Admittance with the increase of the latency/impedance of the Delay Points is more on the hosting environment of lower hardware specification. Given the inherent definition of Admittance, lesser rate of decrease in Admittance implies improved performance on the hosting environment with augmented hardware. This observation confirms that hosting provision with higher specification under same workload condition has a positive impact on the performance of the application operation.

A very effective technique proposed by Johnson to predict the *best-fit* weights of predictor variables in multiple regression by applying a heuristic method for estimating the relative weights is adapted to prove the third null hypothesis (**H3$_0$**) to be false and evaluate the third hypothesis of this thesis. The hypothesis states that under a given workload condition and hosting environment, in the event of simultaneous changes to multiple types of Delay Points, the cumulative latency of the Delay Points of each type has an implicit relative weight associated to it, which determines the degree of its impact on the operation Admittance. Johnson's technique is used to determine the best-fit relative weights associated with the cumulative latencies of the Delay Points of each type, which determine the proportionate contribution of those respective latencies to the overall operation Admittance. A matrix based predictive model is established to forecast an application operation's Admittance and thus Performance for simultaneous changes to multiple types of supporting application component activities. The key to this matrix based method is the single column best-fit DLPrw matrix. This matrix is computed initially from the set of measured actual Delay Point latencies and the corresponding overall application operation Admittances. The data in this DLPrw matrix represents the best-fit values of the relative weights, which determine the proportionate contribution of the respective Delay Point latencies to the overall operation Admittance. The set of relative weights (i.e. the DLPrw matrix data) is then used to predict the operation Admittance for any future set of latency values for the different Delay Points.

Precision of the DLPrw matrix determines the accuracy of the prediction of the impact of simultaneous changes to the various types of Delay Points. The prototype application framework is used to run the experiments and the DLPrw matrix is computed. The precision of the DLPrw matrix is subsequently validated. The percentage error between the operation Admittance computed from the experimental data and the operation Admittance projected through the DLPrw matrix is found to be ***4.459%***. To address real life scenarios and cover non-uniform variation of Delay Points, the application framework is subjected to various process load configurations at random. This time the number of iterations is increased heuristically than the number of iterations conducted while evaluating the third hypothesis. Following the same method as before, the calibrated model subsequently yielded an error of ***3.39%***. The statistical power (Wohlin et al., 2012) of the experiments is also derived to demonstrate the ability of the experiments to reveal the true pattern in the collated data.

9.1.4 Evaluation of the proposed Approach against Research Objectives

In Section 1.3 (Research Objective), we described the research question and defined a set of criteria that this thesis aims to satisfy in order to address the gaps and drawbacks of the previous and current related work in application performance analysis space. Table 10 below provides a snapshot of how the approach proposed in this thesis compares against the aforesaid set of criteria defined in the introduction:

Criteria defined by the Research Question	***The PALP Model***	***Method to analyse changes to Delay Points of one type***	***Method to analyse changes to Delay Points of multiple types***
Repetitive performance testing for changes to evolving applications not required.	Performance is calculated upfront during code modification from ‘*Y’* and ‘*V’*. ‘*Y’* looked up from Delay Point latencies and ‘*V’* computed from workload. Repetitive load testing is not required as in CPM, which is *reactive* i.e. load testing after every build; code modified if performance not met and re-tested. This cycle is repeated till thresholds are met. This repetitive approach not adequate to understand performance changes between application updates.	For existing application operations, reference s*tatistical functions* are first created (Section 3.6). For subsequent application changes involving Delay Points of a specific type, cumulative latency of Delay Points of that type probed by software developers through systems testing and looked up against the reference statistical functions to project ‘*Y*’ and in turn ‘*P*’. Can be done before build and load testing. Doesn’t need repetitive CPM cycles.	For existing application operations, reference *Relative Weights Matrix* of Delay Point latencies (DLPrw matrix) is created first (Section 3.6). For subsequent application changes involving Delay Points of multiple types, cumulative latencies of Delay Points of those types probed by software developers through systems testing. DLPrw matrix applied to measured latencies to project ‘*Y*’ and in turn ‘*P*’. Can be done before build and load testing. Doesn’t need repetitive CPM cycles.
Complex SPE technique, performance models or resource intensive application profilers not required.	*PALP* model is simple and linear within a workload band. It associates ‘*P*’, ‘*Y*’ (related to algorithm) and ‘*V*’ (related to workload). Any two attributes allow computing the third one enabling a three-way predictive model. ‘*Y’* is computed from probing Delay Point latencies. No profilers required. In contrast, applying SPE in practice very challenging where analytical models are built by imposing structural restrictions on the original system model. Significant aspects of the system cannot be effectively represented into the performance model.	The lookup datasets created by the method comprise *statistical functions*. These do not require any complex SPE technique or performance models. These statistical functions are looked up with the probed data during application modification to project ‘*Y*’ and ‘*P*’ for the operations. As the Delay Points represent the internal processing of the operation, their latencies highlight how each type of processing impact ‘*P*’. Hence intensive application profilers not required.	The reference datasets created comprise *Relative Weight Matrices* of Delay Point latencies (DLPrw matrices). A heuristic method for estimating the relative weights of predictor variables in multiple regression is applied to compute the relative weights through simple matrices. No complex SPE technique or performance models required. During application modification DLPrw matrix applied to measured latencies to project ‘*Y*’ and ‘*P*’ for the operations. Delay Point latencies highlight how different types of processing impact ‘*P*’. Intensive application profilers not required.

Table 10: Summary of the Criteria satisfied by the Thesis’s Approach

Criteria defined by the Research Question	***The PALP Model***	***Method to analyse changes to Delay Points of one type***	***Method to analyse changes to Delay Points of multiple types***
No need of additional quality assurance experts. Software developers not required to learn new, non-standard modelling formalisms or use of process algebras.	Simple relational model associating '*P*', '*Y*' and '*V*' used by software developers during creating the initial lookup datasets. No need of experienced modellers. New, non-standard modelling notations, process algebra not used.	Reference *statistical functions* created by software engineers by varying Delay Point (specific type) intensities, probing latencies and plotting against computed '*Y*'. Later on, these functions consulted to project '*Y*' and '*P*' for measured Delay Point latencies. No non-standard modelling formalism or process algebra required. Also, software engineers will be able to analyse without additional quality assurance experts.	Reference *DLPrw matrices* created by varying Delay Point (multiple types) intensities, probing latencies, mapping against computed '*Y*' and applying simple matrix based calculation (use of freely available calculators or automated). During application changes, DLPrw matrices applied to measured latencies to project '*Y*' and '*P*'. No non-standard modelling formalism or process algebra required. Software engineers will be able to analyse without additional quality assurance experts.
Yields result with high precision and accuracy	Corresponding null hypothesis $\boldsymbol{H1_0}$ proved wrong in Section 5.3. Statistical power of the evaluation experiments is calculated to be very high demonstrating ability to reveal true pattern in collated data.	Corresponding null hypothesis $\boldsymbol{H2_0}$ proved wrong in Sections 6.1 and 6.2 for 2 specific types of Delay Points. Precision of the extracted *statistical functions* are validated. The statistical powers of the evaluation experiments calculated for specific Delay Point types. Powers demonstrated very strong ability to reveal true pattern in collated data.	Corresponding null hypothesis $\boldsymbol{H3_0}$ proved wrong in Section 7.3.1 for changes to multiple types of Delay Points. Precision of the extracted *DLPrw matrix* validated. The statistical power of the evaluation experiments is calculated. The power demonstrated very strong ability to reveal true *Relative Weight*s of the Delay Point latencies in collated data.

Table 10: Summary of the Criteria satisfied by the Thesis's Approach (contd)

Criteria defined by the Research Question	***The PALP Model***	***Method to analyse changes to Delay Points of one type***	***Method to analyse changes to Delay Points of multiple types***
Precise performance measures during implementation phase	Model applied while creating reference lookup datasets by probing application operation's Delay Point latencies. For subsequent operation changes, Delay Point latencies probed again through systems testing and measured data looked up against initially created datasets. Applying the model on an existing application or during implementation of its changes ensures precision of measured data. This addresses the drawbacks of performance analysis very early at the architecture and design stages still requiring much detailed information from the implementation.	Lookup *statistical functions* are created by varying the Delay Point (single type) intensities of existing operations, measuring their latencies and plotting against computed '*Y*' and measured '*P*'. These functions are looked up with the probed data during application modification to project '*Y*' and '*P*' for the operations. Using the method on an existing operation or during its modification satisfies the defined criteria related to higher precision of measurement during implementation phase.	*DLPrw matrices* are created by varying the Delay Point (multiple types) intensities of existing operations, measuring their latencies, mapping against computed '*Y*' and applying matrix based computation (use of freely available calculators or automated). During application changes, DLPrw matrices applied to measured latencies to project '*Y*' and '*P*'. Using the method on an existing operation or during its modification addresses the drawbacks of some of the previous performance analysis approaches applied early during the architecture and design phases and lacking precise measurement due to immature implementation.
Proactively target pre-specified performance criterion under a given workload condition in a bottom-up way.	For pre-specified *target* values of '*P*' and '*V*', '*Y*' can be calculated. Delay Point latencies then adjusted to yield the computed '*Y*'. This enables proactive fine-tuning of algorithm and configuration to target a pre-specified performance level, which is not possible in the current CPM paradigm.	*Statistical functions* of '*P*' versus '*V*', '*Y*' versus cumulative Delay Point (single type) latencies and others are created initially. For pre-specified value of '*P*' we compute corresponding '*Y*' and in turn relevant Delay Points' cumulative latency. With this prior knowledge, software engineers can fine-tune the Delay Points to achieve the desired latencies and in turn the target '*P*' for a given workload.	*DLPrw matrices* are created through matrix manipulation using measured Delay Points (multiple types) latencies and corresponding values of '*Y*'. For a pre-specified '*P*', we calculate '*Y*' by using *PALP* model. Combining previously created *DLPrw matrix* and *'Y'*, desired Delay Point latencies are calculated. With this knowledge, software engineers adjust various Delay Points to achieve desired latencies and in turn the target '*P*' under a given workload.

Table 10: Summary of the Criteria satisfied by the Thesis's Approach (contd)

Criteria defined by the Research Question	***The PALP Model***	***Method to analyse changes to Delay Points of one type***	***Method to analyse changes to Delay Points of multiple types***
Proactively assess performance for changes to application operation under a given workload in a top-down way.	For given values of '*V*' and '*Y*' (from workload and Delay Point latencies respectively), '*P*' can be calculated upfront by software developers during systems testing with synthetic workload. Iterations of code change, build, load test, measure not required.	During code modification, latencies of Delay Points (single type) are probed by software developers through system testing. These latencies are then used to extrapolate/interpolate '*Y*' and in turn calculate '*P*' for a particular workload. This can be proactively performed before executing full build and load testing as done in CPM.	During code modification, latencies of Delay Points (multiple types) are probed by software developers through system testing. Applying *DLPrw matrix* to these latencies, '*Y*' and in turn '*P*' are calculated for a particular workload. This can be proactively performed before executing code build and load testing as practised in CPM.
Ascertain inbound workload for an application operation for a target performance level and given application algorithm.	For given values of '*P*' and '*Y*' (from Delay Point latencies), '*V*' is calculated. Difference of the *Stress Point* and '*V*' is the projected workload.	From reference *statistical functions*, '*Y*' is projected for the probed latencies of Delay Points of a specific type. If '*P*' is specified, '*V*' is calculated from '*P*' and projected '*Y*' (*PALP* model). Workload is the difference between Stress Point and calculated '*V*'.	Applying *DLPrw matrix* to probed latencies of Delay Points of multiple types, '*Y*' is calculated. If '*P*' is specified, '*V*' is calculated from '*P*' and projected '*Y*' (*PALP* model). Workload is the difference between Stress Point and '*V*'. This can be performed before executing full build and load testing as done in CPM.

Table 10: Summary of the Criteria satisfied by the Thesis's Approach (contd)

9.2 Contribution of the Thesis

9.2.1 The overall Approach

This thesis proposes a proactive approach that combines model based and measurement based performance engineering techniques to evaluate the performance of existing software applications undergoing frequent updates. The approach does not require repetitive performance testing, complex SPE techniques, performance models and resource intensive application profilers. The software engineers are not required to learn new, non-standard modelling formalisms and notations or to specify the software system using process algebras. The *novel* concepts of operation Admittance, Load Potential and the *PALP* model enable proactive fine-tuning of the application code and configuration by the software engineers upfront during code development and modification to achieve a target performance level instead of recourse to the reactive approach currently used in CPM, which requires *repetitive cycles* of load testing, performance measurement, code modification and build to fix any performance issue.

The methodology proposed in this thesis will also allow the software engineers to assess the impact of the changes to an application operation upfront during code modification without the need to involve additional quality assurance experts. It attempts to circumvent the unnecessary notational hurdle, which acts as an impediment to the understanding and uptake of modern performance analysis technologies. No system level utilization diagnostics and measurement will be required for which the software developers may not have adequate expertise.

9.2.2 *PALP* Model and the novel concepts of Admittance and Load Potential

To analyse and predict the performance of an application operation and to associate the operation's performance to its various internal processing activities and its inbound workload, this thesis introduces two *novel* concepts namely Admittance and Load Potential in the context of runtime consumption of an application operation. Associating the performance of an application operation to the above two novel concepts this thesis establishes an *innovative* relational model referred to as the *PALP* model.

In Electrical Engineering, Admittance is a measure of how easily a circuit or device allows current to flow and is defined as the inverse of the device's Impedance. The concept of Admittance is introduced in software applications. It is defined as the measure of ease with which a request to an application's operation is processed and response sent back. All the granular transaction processing activities required to process a request, cumulatively introduce latency or impedance to the operation transaction's performance. Admittance is the inverse of this impedance created. An overall application operation Admittance 'Y' is thus considered as a key attribute influencing the operation's performance. This is a numeric measure only.

The operation workload (i.e. number of requests hitting the application operation per unit time) affects the performance of the operation. On a specific hosting environment, an application operation has an upper workload limit up till which it is able to maintain a prescribed QoS. If the workload exceeds that threshold, the application fails to maintain the requisite performance level for that operation. This threshold is referred to as the operation's "*Stress Point*" in this thesis. From performance perspective, it is significant to understand whether the current workload is approaching the Stress Point. Hence the *novel* concept of application operation Load Potential 'V' is introduced related to the inbound workload. 'V' is defined as the differential between the operation's Stress Point and its contractual request load (per unit time) e.g. 1000 requests/sec. The unit of this attribute is (unit of time)$^{-1}$.

9.2.3 Bi-directional Method to fine-tune Delay Points of a specific type

The underlying application components catering to an operation's transaction perform various types of processing activities through the Delay Points. There can be different types of Delay Points like in-memory data processing, file I/O, database interaction etc. Using the deduced *PALP* model, this thesis establishes a *bi-directional* method to achieve a *target* performance level during code modification. Firstly, in a *bottom-up* way, the method enables software developers to fine-tune the response times of the Delay Points of a particular type during application code modification to *retrofit* a specified target operation performance level. Secondly, in a *top-down* way it enables software developers to predict the possible change of an operation's performance due to variation of the operation's Delay Points of a specific type. Both of these will be achieved without the need of repetitive load/performance testing (even if the load testing is performed during a CPM cycle), complex SPE techniques, performance models and resource intensive application profilers. As the method is applied during the implementation phase during software development, it addresses the issue highlighted by Balsamo et al., which states that most of the modelling approaches try to apply performance analysis very early, typically at the software architecture and design level and hence still require much detailed information from the implementation phase to carry out performance analysis. The thesis establishes that given the hosting environment, the inbound workload and the other Delay Points remaining constant, changes to the Delay Points of a specific type influences the application operation's Admittance and hence Performance following statistical regression functions.

9.2.4 Bi-directional Method to fine-tune Delay Points of multiple types

Using a combination of the *PALP* model and the method described above, this thesis establishes a second *bi-directional* method to facilitate fine-tuning of Delay Points of *multiple types* simultaneously to achieve a *target* performance level during application code modification. Firstly, in a *bottom-up* way, the method allows the software developers to fine-tune the response times of the Delay Points of various types to achieve a target (specified) operation performance level without the need of repetitive load/performance testing (even if the load testing is performed during a CPM cycle), complex SPE techniques, performance models and resource intensive application profilers. This enables targeting a specific performance level upfront during code development and modification rather than going through a cycle of Load Testing, Performance Measurement, Code Modification and Build in the event of any performance issue. Secondly, in a *top-down* way it enables software developers to predict the possible change of an operation's performance due to changes to multiple types of Delay Points simultaneously. The cumulative latencies of the Delay Points of each individual type are treated as predictor variables for the application operation's admittance and hence the performance. A heuristic method is applied for estimating the best-fit relative weights of predictor variables in multiple regressions.

The *PALP* model along with the two methods established above are very much complementary in nature and in combination facilitates addressing the thesis question as described in Section 1.3 affirmatively. Used jointly, the resulting framework facilitates a flexible ***three-way*** predictive technique involving an application operation's Performance, Admittance and Load Potential.

9.2.5 Other general contribution to Quality of Software

Besides the above *novel* contributions, the framework will facilitate general improvement of the overall quality of software in the following areas:

i. *Quality Improvement* – The established approach will enable the software engineers to proactively fine-tune the application algorithm and configuration to achieve a target performance level under a given load condition. This will facilitate attaining a specified software quality sooner than the current reactive approach of load/performance testing during the CPM cycle. Performance targeted algorithm and configuration will minimise the need of measures like code refactoring and optimization.

ii. *Software Quality Assurance* – Upfront analysis and forecast of the application operation's performance will feed into the quality assurance process early in its lifecycle. As the developers themselves will be able to forecast the variation in performance due to their alterations, issues will be addressed upfront without the need to go through overheads like source code control, code reviews, change or configuration management, testing and release management. To a great extent the need to undo and redo component changes only after performance/load testing will no longer be there. This will ensure timely assurance of the application's performance in addition to reducing the overall costs of software development.

iii. *Verification and Validation* – Early detection of impact of changes will help to verify whether the performance of the modified application operation will conform to any prescribed QoS requirements at an early stage while the development process is still on. This will ensure corrective measures (if warranted) to be adopted in a timely manner avoiding wastage of resource time and cost.

iv. *Defect Characterization* – Performance (in terms of response time) is a quality carrying property of an application or system operation. Ability to analyse and forecast performance anomalies will help in characterization of the defects and early remedial actions may be taken.

v. *Software Quality Management (SQM) Techniques* – The predictive models may prove to be very effective SQM techniques. Proactive fine-tuning of Delay Points serving an application operation to achieve a pre-specified performance level under a specified workload, early detection of performance anomalies and remedial measures during the development lifecycle will ensure that the required level of performance quality is achieved in the software product. It will also ensure conformance to Software Quality Plan (SQP).

vi. *Software Quality Measurement* – The predictive models should help in measuring software efficiency by facilitating identification and prediction of potential operational performance bottlenecks and future scalability problems. Corrective measures may be taken following best coding practices.

9.3 Complementing Cloud and Elasticity

The basic value proposition of cloud computing is to pay as you go and to pay for what you use (Harbert, 2011). An application can expand and contract on demand, across all its tiers (presentation layer, services, database, security and others). This also implies that application's components can grow independently from each other. So, if more storage for the database of an application is needed, it should be able to grow that tier without affecting, reconfiguring or changing the other tiers. Basically Cloud applications behave like a sponge. When water is added to the sponge, it grows in size. In the application world, the more load is put on the application/system, the system grows across extended hardware. The smallest elasticity unit of an Infrastructure as a Service (IaaS) provider and a virtual machine environment is a server (physical or virtual) (Harbert, 2011). For a purely elastic application, as the load increases, more servers are added to the server farm during runtime and the application is run on the augmented server farm. The process of adding computing resources remains transparent to the application.

(Chapman et al., 2010) discusses the implications of software architecture definitions for distributed applications that are to be deployed on Cloud environments. The paper proposes ways for service providers to meet their QoS objectives by examining how software architectures may be described to take full advantage of the capabilities introduced by the Cloud platforms like deployment, management and execution of services across multiple physical locations in a seamless manner. The work identifies a list of requirements and constraints that a provider must declare when deploying and hosting a multi-component application on a Cloud. It presents a language for the description of such requirements and introduces new abstractions like specifying ***runtime*** on-demand scaling. Using a model denotation approach, an architecture is defined to support the abstractions and specify clear behavioural semantics for the presented language with respect to the architecture.

The method established in this thesis will complement the paradigm of Cloud, Elasticity and particularly the kind of work as mentioned above. Our method will facilitate ***development (build time)*** detection of any possible bottlenecks at the application level and remedy if required. Any negative impact may be addressed upfront and potential performance issues can be averted before rolling out the application to production. One constraint of the *PALP* model is the operation's Load Potential band. The performance predictions are made for a given range of inbound workload. This is because we assumed constancy of the operation's overall Impedance and hence the overall Admittance for processing transactions of the same application operation within a given range / band of inbound workload determined by the operation's Load Potential.

Cloud and Elasticity on the other hand enables ***runtime*** extension or contraction of the hardware and the application being hosted on those. The workload is assessed at runtime. If the load warrants provisioning of more computer resources, it is done transparently at the background. The application is then extended to run on the newly provisioned hardware resources. If the load requires less computer resources, then usage of the computer resources is reduced according and the application is run on the reduced infrastructure.

For a given operation Load Potential, where the workload is contracted and doesn't fluctuate to a great extent, Elasticity may not be required for an application. However, for the same Load Potential, the approach proposed in this thesis may be applied during the code development and modification of the application to assess the impact of the Delay Point modifications of the application components.

9.4 Future Work

9.4.1 JIT compilers, Query Optimization, Resource Pooling and Caching

As presented in Section 3.3.1, various types of optimisation techniques like JIT compilation, intelligent query optimisation and other such techniques may be applied to boost performance. Prior to optimisation, the initial few requests will result in relatively slower responses and then there will be improved performance post optimisation for the rest of the application's life. As our controlled experiments are run repeatedly for a number of times (optimal number determined through Standard Deviation of the response times) for the same application configuration and the average latency of each type of Delay Point is recorded, it is assumed that the impact of any performance technique (if applied) should be smoothened out i.e. the data recorded reflects the *warm state*. As such, this thesis does not delve into the details of such performance optimisation techniques. This thesis also does not deal with any finite resource pools in the application layer explicitly. However, to reduce any possibility of resource contention, the application container is configured in a way so that the maximum inbound workload remains within the limit of the maximum number of requests that can be handled concurrently by the container. All of these aspects mentioned impact the application performance either positively or negatively depending on the way those are implemented. As a future extension to the work carried out in this thesis, it would be worth implementing some of these optimisation techniques in our controlled experiments to verify the extent of their impact on the outcome. Different types of caching strategies may also be adopted to improve the performance of application operations. We do not deal with any proactive or reactive caching strategy explicitly in this thesis. To address any implicit caching in the background, the controlled experiments are run repeatedly for a number of times (optimal number determined through Standard Deviation of the response times as explained earlier) for the same application configuration and the average latency of each type of Delay Point is recorded. However, it will be very relevant to introduce both proactive and reactive caching (as explained in Section 3.3) in the prototypical application framework and assess the impact on the application operation's Admittance and Performance.

9.4.2 Tool Support

Using the *novel* concepts of operation Admittance and Load Potential, the thesis enables proactive fine-tuning of the application algorithm and configuration by the software developers upfront during code development and modification to attain the target performance level instead of recourse to the reactive CPM approach. The proposed methodology involves probing the Delay Point latencies to ascertain their impact on the operation's Admittance and Performance. In this thesis, probing of the Delay Point latencies is accomplished through manual instrumentation. The code of the supporting application components is modified to insert the probing / tracing code blocks to record the time taken by the various processing activities (Delay Points).

However, when any application codebase is touched for the purpose of modification, it gets "*polluted*" and bugs may get introduced in many cases. This may potentially impact the timescales and costs of the projects. In view of these, we need to try to achieve a means of inserting the Delay Point probes without polluting the base application code. Hence, in due course we will need some tool support in order to automate the process of instrumentation. In this way, the method of performance analysis and prediction will remain decoupled from the main application. This will mitigate the potential risks mentioned above.

An Aspect Oriented Programming (AOP) language aims to increase modularity by allowing the separation of cross-cutting concerns (Kiczales et al., 1997). Logging exemplifies a cross-cutting concern because a logging strategy necessarily affects every logged parts of the system. Logging thereby *crosscuts* all logged components, classes and methods.

In view of this, we may consider AOP paradigm to develop the tool support for the proposed approach. Through *aspects*, we will be able to perform all the logging activities by applying *advice* at various *join points* specified in a query called *pointcut*.

9.4.3 Delay Point Atomicity and Workload Partitioning

Precision in partitioning the workload is critical to the success of the proposed approach. It has been explained in Section 3.3.1 all the attributes that are taken into consideration for classifying and partitioning workloads and the rationale explained for classifying an application operation as a single class.

However, within an application operation, depending on the decision logic, processing different requests to the same operation may warrant different components to be used based on conditional scenarios, which determine the “process path”. While creating the lookup data sets and models, we will need to consider the different process paths that may be used for serving the operation request based on any conditional logic, if relevant. An application operation may have more than one process path based on certain conditions to serve the request. Each of those process paths may contain different types of Delay Points for processing the request. The systems resource usage requirements for those various Delay Points may be different, thus manifesting themselves in different patterns of delays. In view of these, it is critical to identify all the possible process paths that may be taken by the transactions to serve a request to the application operation. For each of these process paths, the reference lookup data sets and models need to be created initially for analysis and forecast of future operation performances.

The approach proposed in this thesis is founded on the concept of Delay Points. For the various processing activities, the Delay Points interface the system resources as needed. For the integrity of the forecasts it is important that the processing activities and the type of systems resource usage of every individual Delay Point is consistent all the time. To achieve this, the types of Delay Points should be *atomic* and there should not be any *sub-types* of the Delay Points. Also, ideally, there should not be any overlap of activities or systems resource usage between two types of Delay Points at a given point in time. In our experiments, we have created Delay Points for in-memory Data Processing, Database Interactions, File Input / Output and some other types of application level activities. In the instance of Data Processing, there can be different sub-types of data processing like pure numeric, pure textual, pure graphics, combination of numeric and textual etc. The usage of systems resources for pure numeric data processing will be very different from the usage of systems resources for processing of pure textual contents. For example, any type of arithmetic and logical operations will use the Arithmetic and Logic Unit (ALU) of the Central Processing Unit (CPU) of the machine. In computing, an ALU is the digital circuit that performs arithmetic and logical operations (Arithmetic Logic Unit, 2013). However, ALU will not be used typically in a pure textual contents processing. In view of this, if we require different types of data processing to serve the operation request, we will need to have different types of Data Processing Delay Points like Numeric Data Processing Delay Point, Textual Data Processing Delay Point etc. Impedances/latencies for these Delay Points should be measured independently like I_{DPN} for Numeric Data Processing Delay Points, I_{DPT} for Textual Data Processing

Delay Points etc. In our application framework we only used textual data processing. Hence this way of categorizing of the Delay Points is not required.

Figure 34 below shows diagrammatically how the different process paths of an application operation need to be identified for creating the respective data sets and models.

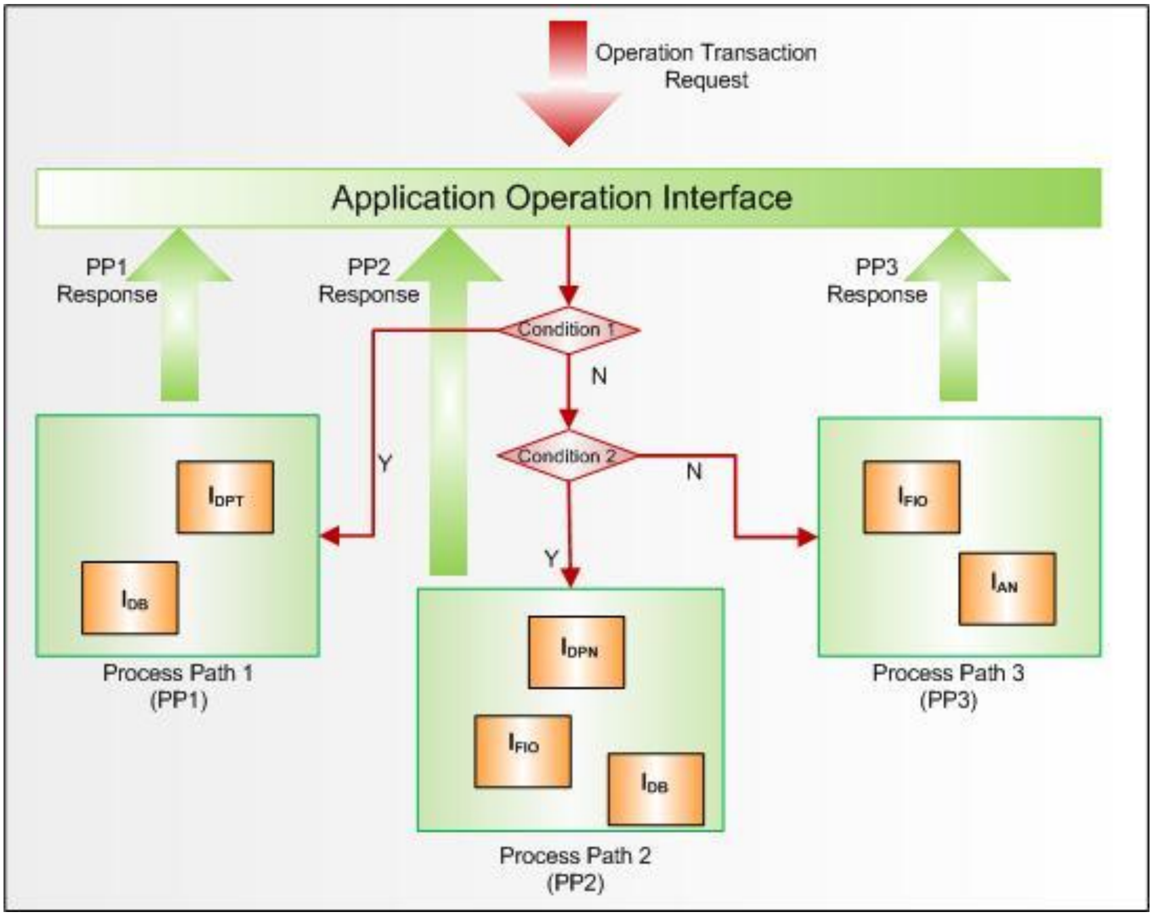


Figure 34: Different Process Paths for the same application operation

In our prototype application framework, the application operations didn’t have much complex conditional logic. Hence, we didn’t have to segregate the process paths for creating the data sets.

9.4.4 Hosting the Framework on a multi node environment

This thesis focuses on the application layer of a system. All the experiments designed and executed are run on an application framework. The framework comprised various types of components like Web Services, Servlets, RMI Server and low level Socket Server. The application framework is run on two different types of hosting environments of different specifications. However, each environment was implemented on a single hardware. This was one of the constraints. In typical heavy duty production systems, applications are hosted on fault tolerant, load balanced environments with multiple physical servers. Hosting the framework on such a multi node environment was beyond the scope of this thesis. In distributed applications, the Delay Points in the highest application layer may need to be measured while the underlying layers may need to be treated as black boxes in a layered way. This method has not been verified in this thesis and may form part of the future work.

This thesis may be extended in the future to be hosted on a multi node environment to simulate real production scale scenario. In a load balanced configuration, the load balancer receives the consumer's request, applies an algorithm to determine the individual loads of each of the physical nodes and directs the request to the node with minimal process load at that point in time. Once the request hits a particular node, from that point onwards, the request is processed by that particular node/physical server and the response is sent back via a proxy server back to the consumer. This is similar to the way we executed our experiments on a single physical server. Hence, the *PALP* model and the methods for performance analysis for modifications to Delay Points of a single type and multiple types will apply in the same way as described in this thesis.

10 References

Anderson, C. (2012), EITL Analyst Insight: Best Practices in Application Testing, Gartner, G00239376.

Archstone Consulting (2013), Optimizing IT Technology Refresh Policies, IT Effectiveness, http://www.archstoneconsulting.com/services/it-effectiveness/white-papers/optimizing-it-technology.jsp

Argent-Katwala, A., Bradley (2006), Functional Performance Specification with Stochastic Probes, Formal Methods and Stochastic Models for Performance Evaluation, LNCS Volume 4054, pp 31 – 46.

Argent-Katwala, A., Bradley, J.T., Dingle, N.J. (2004), Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models, "Proceedings of the 4th Intl. Workshop on Software and Performance", WOSP'04, pp 49 – 58.

Arithmetic Logic Unit (2013), http://en.wikipedia.org/wiki/Arithmetic_logic_unit

Balbo, G. (2001), Introduction to Stochastic Petri Nets, Lectures on Formal Methods and Performance Analysis, LNCS, Volume 2090, pp 84-155.

Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M. (2004), Model-Based Performance Prediction in Software Development: A Survey, "IEEE Transactions on Software Engineering", Volume 30, No.5.

Balsamo, S., Persone, V.D.N., Onvural, R. (2003), Analysis of Queuing Networks with Blocking, Kluwer Academic Publishers.

Barham, P., Donnelly, A., Isaacs, R., Mortier, R. (2004), Using Magpie for request extraction and workload modelling, "Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'2004).

Becker, S., Dencker, T., Happe, J. (2008), Model-Driven Generation of Performance Prototypes, Performance Evaluation Metrics, Models and Benchmarks, LNCS, Volume 5119, pp 79-98.

Becker, S., Koziolek, H., Reussner, R. (2009), The Palladio Component Model for model-driven performance prediction, The Journal of Systems and Software 82, pp 3-22.

Bertolino, A., Mirandola, R. (2004), CB-SPE Tool: Putting Component-Based Performance Engineering into Practice, Component-Based Software Engineering, LNCS, Volume 3054, pp 233-248.

Brosig, F., Huber, N., Kounev, S. (2011), Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems, "Proceedings of 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)", pp 183-192.

Budescu, D.V. (1993), Dominance Analysis: A new approach to the problem of relative importance of predictors in multiple regression, Psychological Bulletin, 114, pp 542 – 551.

Buhr, R.J.A., Casselman, R.S. (1996), Use Case Maps for Object-Oriented Systems. Prentice Hall.

Burges, C.J.C. (1998), A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery 2, pp 121 – 167.

Chapman, C., Emmerich, W., Garlan Marquez, F., Clayman, S., Galis, A. (2010), Software Architecture Definition for On-demand Cloud Provisioning, "Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC)", Chicago.

Chen, S., Liu, Y., Gorton, I., Liu, A. (2005), Performance prediction of component-based applications, The Journal of Systems and Software 74, pp 35 – 43.

Cherkasova, L., Ozonat, K., Ningfang, M., Symons, J., Smirni, E. (2007), Automated Anomaly Detection and Performance Modeling of Enterprise Applications, ACM Transactions on Computer Systems 12, 27(6).

Cidade, M. (2008), "What does a JIT compiler do?", http://stackoverflow.com/questions/95635/what-does-a-just-in-time-jit-compiler-do

Clark, A., Gilmore, S. (2008), State-Aware Performance Analysis with eXtended Stochastic Probes, Computer Performance Engineering, LNCS, Volume 5261, pp 125 – 140.

Clark, M. (2009), “JUnitPerf”, Clarkware Consulting Inc, http://www.clarkware.com/software/JUnitPerf.html#limitations.

Compuware, (2006), Applied Performance Management Survey.

Cortellessa, V., Marco, A.D., Inverardi, P. (2005), Transformations of software models into performance models, ICSE

Cortellessa, V., Mirandola, R. (2000), Deriving a Queuing Network Based Performance Model from UML Diagrams, “Proceedings of ACM Intl. Workshop on Software and Performance, WOSP’2000, pp 58-70.

D’Ambrogio, A. (2005), A model transformation framework for the automated building of performance models from UML models, “Proceedings of the 5th Intl Workshop on Software and Performance”, WOSP’05, pp 75-86.

Debusmann, M., Geihs, K. (2003), Efficient and Transperant Instrumentation of Application Components Using an Aspect-Oriented Approach, Self-Managing Distributed Systems, LNCS Volume 2867 Springer, pp 209-220.

Denaro, G., Polini, A., Emmerich, W. (2004), Early Performance Testing of Distributed Software Applications, Proceedings of 4th Intl Workshop on Software and Performance, WOSP’04, pp 94-103.

Di, M., Joo, E.M., Beng, L.H. (2008), A comprehensive study of Kalman filter and extended Kalman filter for target tracking in Wireless Sensor Networks, “Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp 2792 – 2797.

Duggan, J., Hotle, M., Murphy, T. E. (2011), “Determining the ‘Right’ Number of Nonproduction Environments”, Gartner, G00210456.

Easy Forex (2014), “Forex Volatility”, http://www.easy-forex.com/int/volatility/

Econterms. (2013), “Regression Function”, About.com Economics, http://economics.about.com/od/economicsglossary/g/regressionf.htm.

Ehlers, J., Hoorn, A.V., Waller, J., Hasselbring, W. (2011), Self-adaptive software system monitoring for performance anomaly localization, “Proceedings of the 8th ACM Intl Conference on Autonomic Computing”, pp 197-200.

eHow (2014), “Technical Analysis of Stocks and Commodities”, http://www.ehow.com/about_5431905_technical-analysis-stocks-commodities.html

Farrell, C. (2010), “The Fast Guide to Application Profiling”, Simple-talk, http://www.simple-talk.com/dotnet/performance/the-fast-guide-to-application-profiling/

Fleury, M. (2002), JMX: Managing J2EE with Java Management Extensions, Sams.

Folger, J. (2011), Investopedia, “Should You Trade Forex Or Stocks”, http://www.investopedia.com /articles/forex/11/forex-or-stocks.asp

Fourer, R., Gay, D.M., Kernighan, B.W. (2003), Piecewise-Linear Programs, AMPL: A Modeling Language for Mathematical Programming, chapter 17.

Franks, G., Hubbard, A., Majumdar, S., Petriu, D.C., Rolia, J., Woodside, C.M. (1995), A Toolset for Performance Engineering and Software Design of Client-Server Systems, Performance Evaluation, Volume 24, No. 1-2, pp 117-135.

Gilly, K., Alcaraz, S., Juiz, C., Puigjaner, R. (2009), Analysis of burstiness monitoring and detection in an adaptive web system, Computer Networks.

Gilmore, S., Hillston, J. (1994), The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling, “Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation”, Springer-Verlag, No. 794, pp 353 – 368.

Gilmore, S., Hillston, J., Kloul, L., Ribaudo, M. (2004), Software performance modelling using PEPA nets, “Proceedings of the 4th Intl Workshop on Software and Performance”, WOSP’04, pp 13 – 23.

Gilmore, S., Hillston, J., Ribaudo, M., Kloul, L. (2003), PEPA nets: A structured performance modelling formalism, Performance Evaluation, 54(2), pp 79 – 104.

Gilmore, S., Kloul, L. (2005), A unified tool for performance modelling and prediction, Reliability Engineering and System Safety 89, pp 17 – 32.

Gomaa, H., Menasce, D.A. (2000), Design and Performance Modeling of Component Interaction Patterns for Distributed Software Architectures, “ACM Proceedings of Intl Workshop on Software and Performance”, pp 117-126.

Gomaa, H., Menasce, D. (2001), Performance Engineering of Component Distributed Software Systems, Performance Eng., eds, pp 40-55.

Gomez-Martinez, E., Merseguer, J. (2006), ArgoSPE: Model-Based Software Performance Engineering, Petri Nets and Other Models of Concurrency – ICATPN, LNCS, Volume 4024, pp 401 – 410.

Grassi, V., Mirandola, R. (2001), UML Modelling and Performance Analysis of Mobile Software Architectures, The Unified Modeling Language. Modeling Languages, Concepts and Tools, LNCS Volume 2185, pp 209 - 224.

Grassi, V., Mirandola, R. (2004), Towards Automatic Compositional Performance Analysis of Component-based Systems, “Proceedings of the 4th Intl Workshop on Software and Performance”, WOSP’04, pp 59-63.

Grassi, V., Mirandola, R., Sabetta, A. (2005), From Design to Analysis Models: a Kernel Language for Performance and Reliability Analysis of Component-based Systems, "Proceedings of the 5th International Conference on Software and Performance", WOSP'05, pp 25 – 36.

Grassi, V., Mirandola, R., Randazzo, E., Sabetta, A. (2008), KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability, The Common Component Modeling Example, LNCS, Volume 5153, pp 327-356.

Gu, G., Petriu, D.C. (2002), XSLT Transformation from UML Models to LQN Performance Models, "ACM Proceedings of Intl Workshop on Software and Performance", pp 227-234.

Gunther, N.J. (2005), Hit-and-Run Tactics for Website Scalability, Guerrilla Capacity Planning, Part I, Springer-Verlag.

Haines, S. (2008), "Continuous Integration and Performance Testing", http://www.drdobbs.com/tools/continuous-integration-and-performance-t/206105978?pgno=2

Hansen, K.H. (2011), "Load Testing your Applications with Apache JMeter", http://www.jguru.com/article/server-side/load-testing-with-apache-jmeter.html

Happe, J., Becker, S., Rathfelder, C., Friedrich, H., Reussner, R.H. (2010), Parametric performance completions for model-driven performance prediction, Performance Evaluation 67, pp 694-716

Happe, J., Westermann, D., Sachs, K., Kapova, L. (2010), Statistical Inference of Software Performance Models for Parametric Performance Completions, "Research into Practice – Reality and Gaps", Lecture Notes in Computer Science, Volume 6093, pp20-35.

Harbert, T. (2011), Cloud value proposition, PwC, http://www.pwc.com/gx/en/technology/cloud-computing/assets/Article_4_--_Cloud_Value_Proposition.pdf

Harbitter, A., Menasce, D. A. (2001), Performance of Public Key-Enabled Kerberos Authentication in Large Networks, Proceedings of IEEE Symposium on Security and Privacy, Oakland, California.

Hermanns, H., Herzog, U., Katoen, J. P. (2002), “Process Algebra for Performance Evaluation”, Theoretical Computer Science, 274(1-2), pp 43-87.

Hill, M. C. (1998), “Methods and Guidelines for effective Model Calibration”, Water-Resources Investigations Report 98-4005, US Geological Survey.

Hillston, J. (1996), A Compositional Approach to Performance Modelling, Cambridge University Press.

IBM (2002), IBM Rational PurifyPlus, Purify, PureCoverage and Quantify: Getting Started, G126-5339-00.

IBM (2011), Factors that affect resource utilization, http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp.

Investopedia (2014), “Fundamental Analysis”, http://www.investopedia.com/terms/f/fundamentalanalysis.asp

ITBusinessEdge (2014), “Cache”, http://www.webopedia.com/TERM/C/cache.html
Iyengar, A., Challenger, J., Dias, D., Dantzig, P. (2000), High Performance Web Site Design Techniques, Web Design, IEEE Internet Computing.

Johnson, J.W. (2000), A Heuristic Method for Estimating the Relative Weight of Predictor Variables in Multiple Regression, Multivariate Behavioral Research, 35(1), pp 1 – 19, Lawrence Erlbaum Associates, Inc.

Johnson, M. W. (1998), Monitoring and diagnosing application response time with ARM, “Proceedings of IEEE 3rd International Workshop on Systems Management”, Newport, RI, USA, pp 4-13.

JUnit 4 (n.d.), A programmer-oriented testing framework for Java, JUnit, http://junit.org/

Kalman, R. E. (1960), A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering, Volume 82 No. 1, pp35–45

Kant, K. (1992), Introduction to Computer Systems Performance Evaluation, McGraw-Hill.

Kargupta, S., Black, S. (2009a), Service Operation Impedance and its role in projecting some key features in Service Contracts, "Proceedings of the 9th International Conference on Web Engineering", ICWE, San Sebastian, Spain.

Kargupta, S., Black, S. (2009b), Visualizing the Underlying Trends of Component Latencies affecting Service Operation Performance, "Proceedings of the IEEE International Conference on Advances in Computational Tools for Engineering Applications", Lebanon.

Kargupta, S., Black, S. (2011), A Novel Approach for Service Performance Analysis and Forecast, Journal of Web Engineering, Volume 11 No. 2, pp146 - 176.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, Jean-Marc., Irwin J. (1997), Aspect-Oriented Programming, "Proceedings of the European Conference on Object-Oriented Programming (ECOOP)", Springer-Verlag LNCS 1241.

Kleinrock, L. (1975), Queuing Systems, Volume 1. J. Wiley.

Knuth, D. E. (1971), An empirical study of FORTRAN programs, Software Practice and Experience, Volume 1, No. 2.

Kounev, S. (2006), Performance modelling and evaluation of distributed component-based systems using queuing petri nets, IEEE Transactions on Software Engineering, Volume 32, No. 7, pp 486 – 502.

Kounev, S., Buchmann, A. (2003), Performance modelling of distributed e-business applications using queuing petri nets, Proceedings on ISPASS'03, pp 143 – 155.

Koziolek, H. (2010), Performance evaluation of component-based software systems: A survey, Performance Evaluation, Volume 67, Issue 8, pp 634-658.

Kuperberg, M., Krogmann, K., Reussner, R. (2008), Performance Prediction for Black-Box Components Using Reengineered Parametric Behaviour Models, "Proceedings of the 11th Intl Symposium on Component-Based Software Engineering", CBSE'08, pp 48 – 63.

Lindeman, R.H., Merenda, P.F., Gold, R.Z. (1980), Introduction to bivariate and multivariate analysis, Glenview, IL: Scott, Foresman and Company.

Liu, Y., Fekete, A., Gorton, I. (2005), Design-Level Performance Prediction of Component-Based Applications, "IEEE Transactions on Software Engineering", Volume 31, No.11.

Lloyd, O. (2012), "JMeter vs LoadRunner in terms of vusers", http://www.stackoverflow.com/questions/10648491/jmeter-vs-loadrunner-in-terms-of-vusers

Lo'pez-Grao, J.P., Merseguer, J., Campos, J. (2004), From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering, "4th Intl Workshop on Software and Performance", WOSP'04, CA, pp 25-36.

Malony, A. D., Helm, B. R. (2001), A theory and architecture for automating performance diagnosis, Future Generation Computer Systems, 18(1), pp 189-200

Mania, D., Murphy, J. (2002), Framework for predicting the performance of component-based systems, "Proceedings of the IEEE 10th Internation Conference on Software, Telecommunications and Computer Networks, SoftCOM, pp 46-50.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G. (1995), Modeling with Generalized Stochastic Petri Nets, J. Wiley.

Martens, A., Koziolek, H., Becker, S., Reussner, R. (2010), Automatically Improve Software Architecture Models for Performance, Reliability and Cost Using Evolutionary Algorithms, Proceedings of the first joint WOSP/SIPEW Intl Conference on Perfrmance Engineering, pp 105-116.

Marz, T. (2005), Diagnostic Software – What your developer Doesn't Know, Integrated Diagnostics: Operational Missions, Diagnostic Types, Characteristics and Capability Gaps, Carnegie Mellon University.

Marzolla, M. (2004), Simulation-Based Performance Modeling of UML Software Architectures, PhD Thesis: TD-2004-1, Dipartimento Di Informatica, Universita Ca Foscari Di Venezia.

Menasce, D. A., Virgilio A. F. Almeida, V. A. F. (2002), Capacity Planning for Web Services. Metrics, Models and Methods, Prentice Hall PTR.

Merson, P., Hissam, S. (2005), Predictability by construction, Psters of OOPSLA, San Diego, ACM Press, pp 134-135.

Meier, P., Kounev, S., Koziolek, H. (2011), Automated Transformation of Component-Based Software Architecture Models to Queuing Petri Nets, Intl Symposium on MASCOTS, pp 339 – 348.

Mos, A., Murphy, J. (2002), Performance Management in Component-Oriented Systems Using a Model Driven Architecture Approach, "Proceedings of the 6th Intl Enterprise Distributed Object Computing Conference, pp 227-237.

MSDN (2014a), "Synchronous and Asynchronous Operations", http://msdn.microsoft.com/en-us/library/ms734701(v=vs.110).aspx.

MSDN (2014b), "Loading the Cache", http://msdn.microsoft.com/en-us/library/ff648446.aspx

Murphy, T. E., Kowall, J. (2012), "Leverage Your Application Performance Monitoring Through the Application Life Cycle", Gartner, G00227165.

Nethercote, N., Seward, J. (2003), Valgrind: a program supervision framework, Electronic Notes in Theoretical Computer Science, 89(2), pp 1-23.

Niski, J., Selip, S. (2008), "To Err Is Human, So Test That Software", Burton Research, G00203503.

Object Management Group (OMG). (2001), Model Driven Architecture, OMG document number ormsc/2001-07-01.

Object Management Group (OMG). (2002), UML Profile for Schedulability, Performance and Time Specification, OMG Adopted Specification ptc/02-03-02.

Object Management Group (OMG). (2006), UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE).

Oracle (2014), “Thread Pools”, http://docs.oracle.com/javase/tutorial/essential/concurrency/pools.html

Petriu, D.B., Woodside, M. (2004), A Metamodel for Generating Performance Models from UML Designs, “The Unified Modeling Language – Modelling Languages and Applications”, LNCS, Volume 3273, pp 41-53.

Petriu, D.B., Woodside, M. (2005), Software performance models from system scenarios, Performance Evaluation 61, pp 65 – 89.

Petriu, D.B., Woodside, M. (2007), An intermediate metamodel with scenarios and resources for generating performance models from UML designs, Software & Systems Modeling, Springer, Volume 6, Issue 2, pp 163-184.

Petriu, D.C., Shen, H. (2002a), Applying UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications, “Proceedings of Seventh Intl Conf. Modelling Techniques and Tools for Performance Evaluation”, pp 159-177.

Petriu, D.C., Wang, X. (2000), From UML Descriptions of High-Level Software Architectures to LQN Performance Models, Applications of Graph Transformations with Industrial Relevance, LNCS, Volume 1779, pp 47-63.

Petriu, D.C., Woodside, C.M. (2002b), Software Performance Models from System Scenarios in Use Case Maps, “Proceedings of 12th Intl Conference on Modelling Tools and Techniques for Computer and Comm. System Performance Evaluation”, pp 141 – 158.

Pfleeger, S.L. (1994), Experimental design and analysis in software engineering part 1-5, ACM Sigsoft Software Engineering Notes.

Quest Software Inc., Performance, http://www.quest.com/performance.

Rajamony, R., Elnozahy, M. (2001), Measuring Client-Perceived Response Times on the WWW, “Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)”.

Robson, C. (2002), Real World Research: A Resource for Social Scientists and Practitioners-Researchers, 2nd Edition, Blackwell, Oxford/Madden.

Rolia, J.A., Sevcik, K.C. (1995), The Method of Layers, “IEEE Transactions in Software Engineering, Volume 21, No. 8, pp 682-688.

Runeson, P., Host, M., Rainer, A.W., Regnell, B. (2012), Case Study Research in Software Engineering, Guidelines and Examples, Wiley, Hoboken.

Saltzman, M. (2013), A Little Set Theory (Never Hurt Anybody), Department of Mathematical Sciences, Clemson University.

SmartBear. (2012), “Why Automated Testing”, http://support.smartbear.com/articles/testcomplete/manager-overview/.

Smith, C.U. (1990), Performance Engineering of Software Systems, Addison Wesley.

Smith, C.U., Llado, C.M., Cortellessa, V., Marco, A.D., Williams, L.G. (2005), From UML models to software performance results: an SPE process based on XML interchange formats, “Proceedings of the 5th International Workshop on Software and Performance”, WOSP’05, ACM, NY, pp 87- 98.

Smith, C. U., Williams, L. G. (1997), Performance Engineering Evaluation of Object-Oriented Systems with SPE.ED, “Computer Performance Evaluation: Modelling Techniques and Tools”, No.1245, Springer-Verlag.

Smith, C.U., Williams, L.G. (2002), Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software, Addison Wesley.

Soliman, A. F. (2007), Intelligent Query Answering with Data Mining techniques, Proceedings of the International Conference on Computer Engineering and Systems 2007, pp427 – 432.

Southwire. (2013), “Solving for Conductor Impedance and Admittance”, http://www.southwire.com/support/ solving-for-conductor-impedance-and-admittance.htm

Sonata Software. (2012), “Optimized Performance Testing”, http://www.sonata-software.com/web/sonata_en/services/testing_services/.

Spec. (2011), SPECjEnterprise2010 Design Document, Standard Performance Evaluation Corporation, http://www.spec.org/jEnterprise2010.

Staines, A.S. (2008), Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets, Engineering of Computer Based Systems, pp 191 – 200..

Staines, A.S. (2010), A triple graph grammar (TGG) approach for mapping UML 2 activities into Petri nets, “Proceedings of the 9th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems”, pp 90 – 95.

Taylor, W. A. (1995), Process Tolerancing: A Solution to the Dilemma of Worst-Case versus Statistical Tolerancing, Taylor Enterprises Inc., http://www.variation.com/techlib/ta-2full.html

TestPlant (2011), Black-box vs. White-box Testing, TestPlant.com, http://www.testplant.com/wp-content/uploads/downloads/2011/06/BB_vs_WB_Testing-1.pdf

Thakkar, D., Hassan, A.E., Hamann, G., Flora, P. (2008), A framework for measurement based performance modelling, “Proceedings of the 7th Intl Workshop on Software and Performance”, WOSP’08, pp 55-66.

Tipping, M.E. (2000), The Relevance Vector Machine, Microsoft Research.

Trivedi, K.S. (2001) Probability and Statistics with Reliability, Queuing and Computer Science Applications, John Wiley and Sons.

Tubias, O. (n.d.), “B2B Cost Cutting and Cost Saving Portal”, CostKiller.net, http://costkiller.net/costs-savings/costs-cutting-The-Migration-to-VoIP.htm.

Vapnik, V.N. (1998), Statistical Learning Theory, Wiley, New York.

Weisstein, E.W. (2013), Least Square Fitting, MathWorld – A Wolfram Web Resource, http://mathworld.wolfram.cm/LeastSquaresFitting.html.

Welch, G., Bishop, G. (2006), An Introduction to the Kalman Filter, Technical Report, ACM Digital Library.

Westermann, D., Happe, J. (2010), Towards performance prediction of large enterprise applications based on systematic measurements, “Proceedings of the 15th Intl Workshop on Component-Oriented Programming”, WCOP’10.

Williams, L.G., Smith, C.U. (2002), PASA: A Method for the Performance Assessment of Software Architectures, “Proceedings of the ACM Intl Workshop on Software and Performance”, WOSP’02, pp 179-189.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A. (2012), Experimentation in Software Engineering, Springer.

Woodside, C.M., Neilson, J., Petriu, S., Majumder, S. (1995), The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software, “IEEE Trans. Computer, Volume 44, pp 20-34.

Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J. (2005), Performance by Unified Model Analysis (PUMA), “Proceedings of the 5th Intl Workshop on Software and Performance WOSP’05.

Woodside, M., Franks, G., Petriu, D. C. (2007), The Future of Software Performance Engineering, “Future of Software Engineering, FOSE’07”, pp 171-187

Wu, X., McMullan, D., Woodside, M. (2003), Component Based Performance Prediction, “Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction”.

Wu, X., Woodside, M. (2004), Performance Modeling from Software Components, “Proceedings of the 4th Intl Workshop on Software and Performance”, WOSP’04, pp 290 – 301.

Zuse, H. (1998), A Framework of Software Measurement, de Gruyter, ISBN 3-11-015587-7

Annexes

Annex A

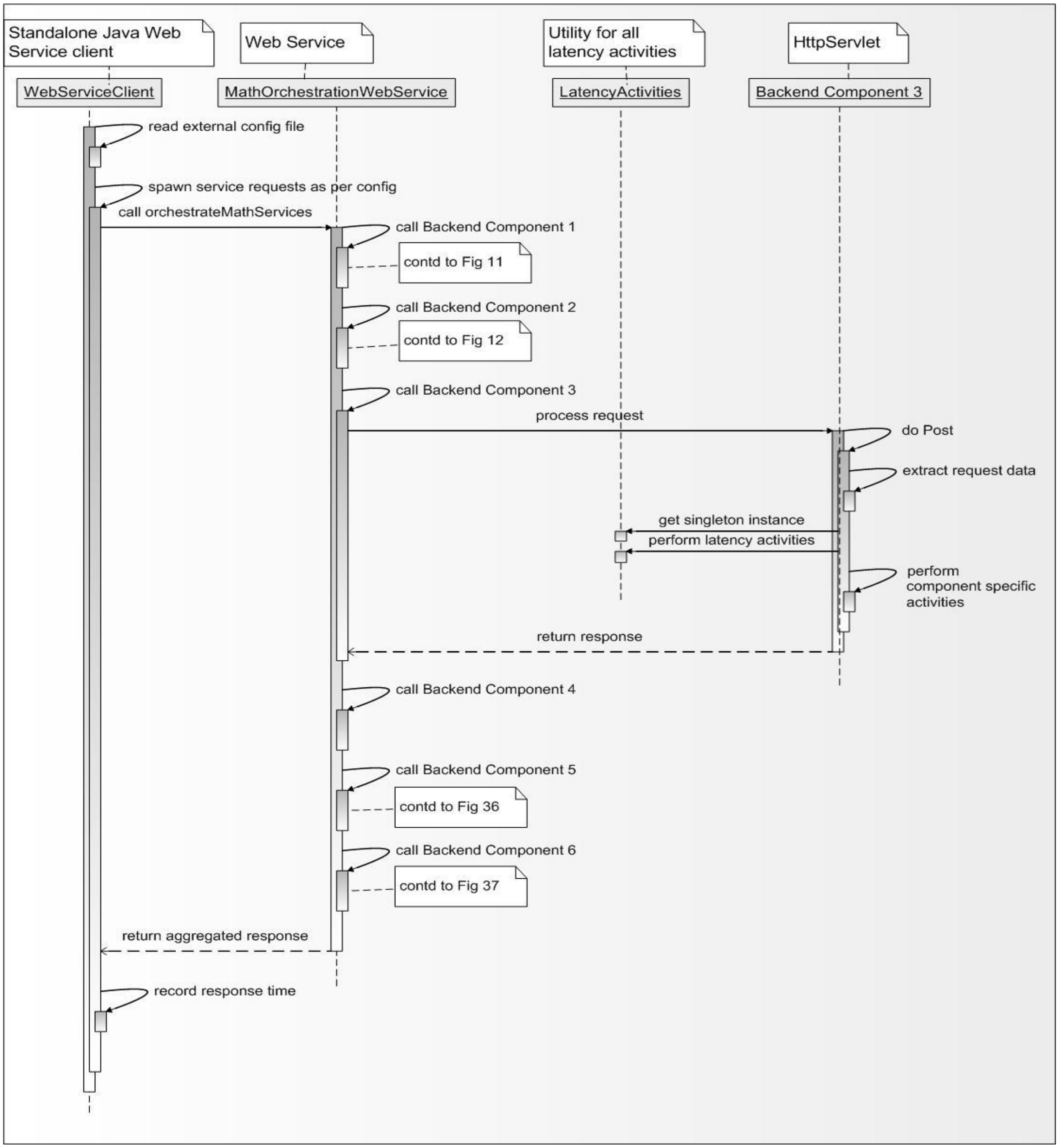


Figure 35: **Sequence model for Backend Component 3 activities**

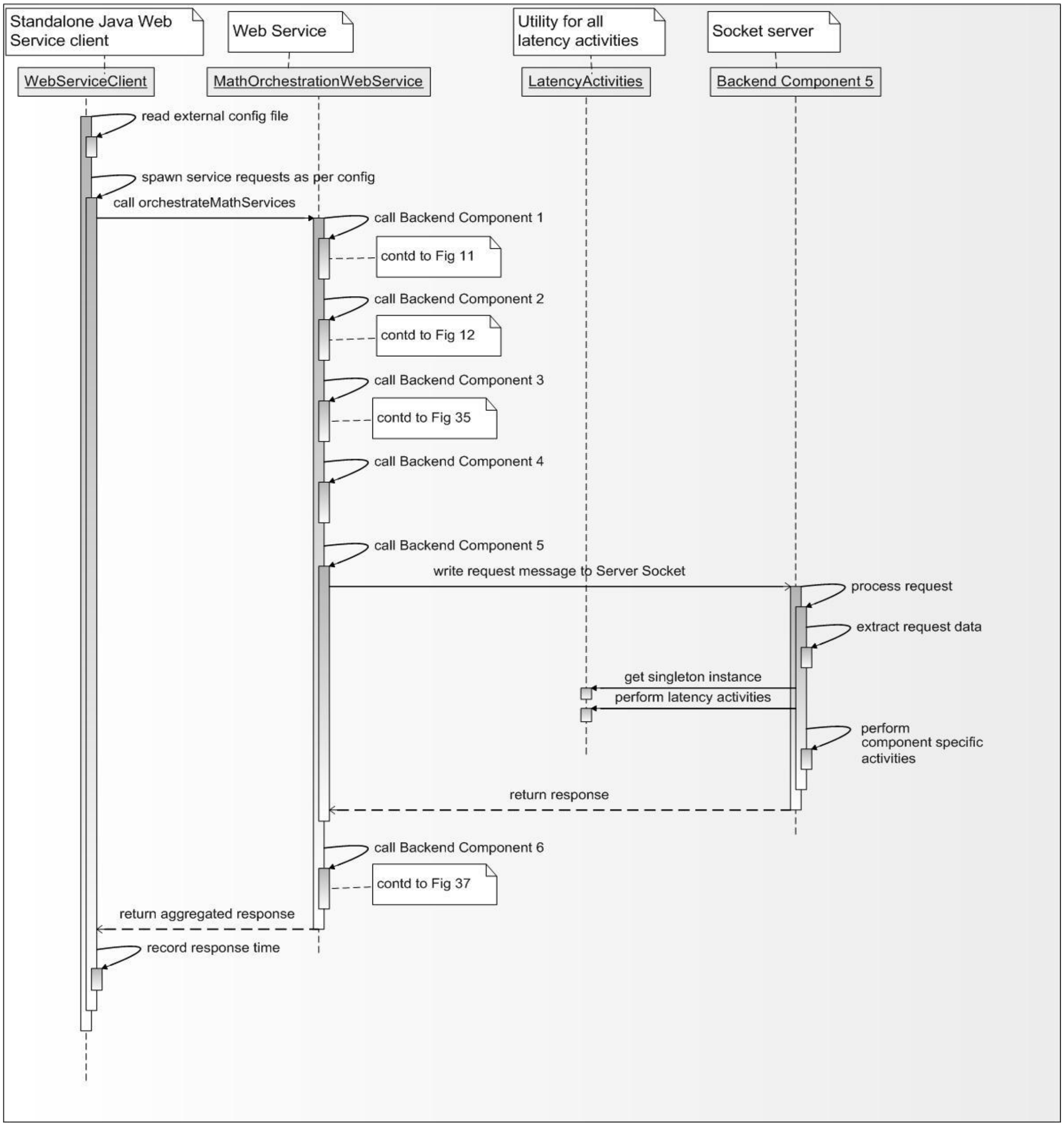


Figure 36: **Sequence model for Backend Component 5 activities**

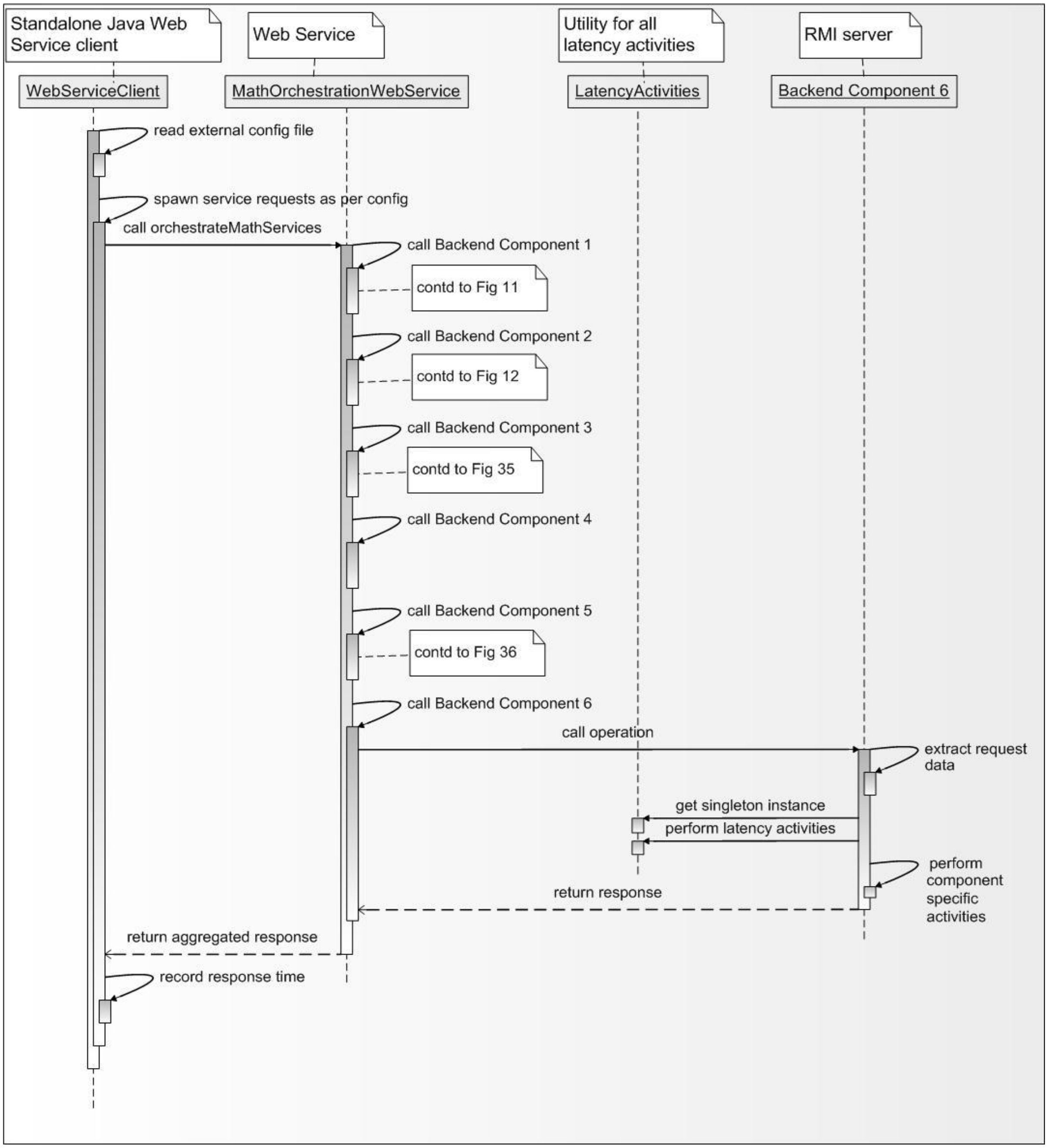


Figure 37: **Sequence model for Backend Component 6 activities**

Annex B

Source Code of WebServiceClient.java

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;


/**
 * @(#)WebServiceClient.java
 *
 *
 * @author Sid Kargupta
 * @version 1.00 2008/9/1
 */

public class WebServiceClient implements Runnable {
        private int counter;
        private Service service;
        private Call call;
        private static String endPoint =
        "http://localhost:9090/axis/MathOrchestrationWebService.jws";
        private static String operation = "orchestrateMathServices";
        private static long SECOND_IN_MILLIS = 1000;
        private int dataLoop;
        private int fileLoop;
        private int dbLoop;
        private String authorizeIndicator;
        private String authenticationIndicator;
        private String whatToPrint;
        private String userId;
        private String password;
        private String data = RESEARCHOFSIDATUNIVERSITYCOLLEGELONDON";

        public WebServiceClient(int count,
                                                int dataProcessLoop,
                                                int fileIOLoop,
                                                int dbAccessLoop,
                                                String authorization,
                                                String authentication,
                                                String id,
                    String pwd,
                    String toPrint) {
      this.counter = count;
      this.dataLoop = dataProcessLoop;
        this.fileLoop = fileIOLoop;
        this.dbLoop = dbAccessLoop;
        this.authorizeIndicator = authorization;
        this.authenticationIndicator = authentication;
        this.whatToPrint = toPrint;
        this.userId = id;
        this.password = pwd;

        try {
            service = new Service();
        call = (Call) service.createCall();
            call.setOperationName(new QName(endPoint, operation));
            call.setTargetEndpointAddress( new java.net.URL(endPoint) );
        }
        catch (Exception e) {
            System.out.println("Orchestration Service Encountered Exception Or Not
            Available");
            System.err.println("Execution failed. Exception: " + e);
        }
    }

    public void run() {

        String output = "Orchestration Service Not Available";
        String inputToServices = "100,200" + "," + dataLoop + "," + fileLoop + "," +
        dbLoop + "," + authorizeIndicator + "," + authenticationIndicator + "," + userId + ","
        + password + "," + data + counter + "," + whatToPrint;

        try {
            long beginTime = new Date().getTime();
            output = (String) call.invoke( new Object[] {inputToServices} );
            long responseTime =  new Date().getTime()-beginTime;
            System.out.println(responseTime);
        }
        catch (Exception e) {
            System.err.println("Execution failed for Request "                  +
counter + "\n Reason : " + e);
            System.exit(1);
        }
    }

    public static void main (String[] args)  {
        Thread client = null;
        int count = 100;
        int secs = 1;
        long timeLimit = SECOND_IN_MILLIS;
        int sleepTime = 1;
        int numOfReq = 0;
        int dataProcessLoop = 0;
        int fileIOLoop = 0;
        int dbAccessLoop = 0;
        String authorization = null;
        String authentication = null;
        String id = null;
        String pwd = null;
        String toPrint = null;
        Properties props = new Properties();

        try {
props.load(new FileInputStream("latency.properties"));
        count = Integer.parseInt( (String)props.getProperty("numOfRequests"));
        secs = Integer.parseInt( (String)props.getProperty("totalDurationOfRequests"));
        sleepTime = Integer.parseInt( (String)props.getProperty("requestIntervalInMS"));
        dataProcessLoop = Integer.parseInt( (String)props.getProperty("dataProcessLoop"));
        fileIOLoop = Integer.parseInt( (String)props.getProperty("fileIOLoop"));
        dbAccessLoop = Integer.parseInt( (String)props.getProperty("dbAccessLoop"));
        System.out.println("dbAccessLoop = " + dbAccessLoop);
        authorization = (String)props.getProperty("authorization");
        authentication = (String)props.getProperty("authentication");
        toPrint = (String)props.getProperty("toPrint");
id = (String)props.getProperty("userId");
pwd = (String)props.getProperty("password");
}
catch (Exception e) {
e.printStackTrace();
}

        timeLimit = secs * SECOND_IN_MILLIS;
        long elapsedTime = 0;
        long startTime = new Date().getTime();


        for (int i=0; i < count; i++) {

                client = new Thread(new WebServiceClient(i, dataProcessLoop,
                fileIOLoop, dbAccessLoop, authorization, authentication, id, pwd,
                toPrint));

                client.start();

                try {
                        Thread.currentThread().sleep(sleepTime);
                }
                catch(InterruptedException ie) {
```

```
                                        System.out.println("Sleep interrupted!");
                                        System.exit(0);
                              }
                              numOfReq = i;
                    }

                    elapsedTime =  new Date().getTime() - startTime;
                    System.out.println("" + ++numOfReq + " SERVICE REQUESTS MADE in " +
                    elapsedTime + " ms!");
          }
}
```

Source Code of Latencies.java

```
public interface Latencies {
          public void someDataProcessing(int dataLoop, String service, String data, String toPrint);

          public void someFileIO (String fileName, int fileLoop, String service, String toPrint);

          public boolean someAuthorization(String authorizeIndicator, String user, String service,
          String toPrint);

          public boolean someAuthentication (String authenticationIndicator, String userId, String
          password, String service, String toPrint);

          public void someDBAccess(int dataLoop, String service, String toPrint);
}
```

Annex C

Screen Shots of command prompt responses while running the WebServiceClient to establish the *PALP* model

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=500
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
406
125
125
188
109
234
125
234
125
9 SERVICE REQUESTS MADE IN 5 SECONDS!!

AVG RESPONSE IN 185.66 MILLI SECS
***************************************************************
```

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=400
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
391
125
125
203
125
281
125
219
125
203
125
203
12 SERVICE REQUESTS MADE IN 5 SECONDS!!

AVG RESPONSE IN 187.5 MILLI SECS
**************************************************************
```

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=300
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
500
219
109
140
218
203
250
125
109
141
188
203
203
125
125
15 SERVICE REQUESTS MADE IN 5 SECONDS!!

AVG RESPONSE IN 190.53 MILLI SECS
**************************************************************
```

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=200
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
375
657
312
141
156
125
125
141
141
172
172
156
157
172
156
172
156
156
157
188
156
125
22 SERVICE REQUESTS MADE IN 5 SECONDS!!
AVG RESPONSE IN 194 MILLI SECS
**********************************************************************
```

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=100
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
875
1172
969
891
1047
1235
1000
890
1016
1375
1266
1109
1375
1516
1204
1328
1266
1375
1406
1406
1313
1110
1437
1172
```

```
1031
1188
1032
1265
953
938
37 SERVICE REQUESTS MADE IN 5 SECONDS!!
922
797
765
516
672
329
438

AVG RESPONSE IN 1070.24 MILLI SECS
************************************************************
```

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=50
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
1093
1219
1562
1890
2063
1437
1968
```

2844
2609
3391
3343
3344
3172
3094
3297
3219
2938
2797
2734
2985
59 SERVICE REQUESTS MADE IN 5 SECONDS!!
2750
2703
2906
2484
2281
2812
2797
3125
2562
2453
2610
2500
2859
812
3047
2515
2578
1359
1234
3000
2250
2562
3563
3546
4031
3063
2688

```
4031
3594
3156
2734
3234
4110
3453
3656
3938
3296
4312
3266
```

AVG RESPONSE IN 2794.39 MILLI SECS
****

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=30
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.

1781
2125
2593
3063
2797
3141
3406
3437
```

```
3453
4141
3641
4172
3718
```

72 SERVICE REQUESTS MADE IN 5 SECONDS!!

```
3234
3703
3610
3890
3641
3719
4203
4375
3765
3265
3625
3359
2797
2187
1891
4172
4187
4875
2828
3172
4547
3687
4719
4297
4609
3625
4671
4625
2546
5890
3593
4406
4000
4313
4125
```

```
4625
4422
4922
3156
3953
3703
4688
3641
4375
4343
3703
3859
4219
5078
4141
5172
4219
4188
4547
4672
4922
4594
4593
4984
```

AVG RESPONSE IN 3893.58 MILLI SECS
****

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=20
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
```

```
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.

1640
1656
1890
2297
2672
2734
2688
2969
3219
2812
3610
3360
2829
4000
2766
3484
3688
3735
88 SERVICE REQUESTS MADE IN 5 SECONDS!!
3985
3485
2421
3343
3234
3344
3156
3234
3078
3391
3031
4453
4235
4328
3015
2875
5000
5125
```

4687
5578
4687
4906
4578
4984
5375
5000
5156
5750
6125
5141
4969
4766
5500
5547
6078
5125
5047
5594
4969
6204
6015
6531
6188
6188
5922
5265
5531
6718
7360
5500
6000
5593
6765
6625
6500
5921
7703
7062
6500

```
7328
8203
7422
7516
6828
6765
7485
7891
7250
7547
6390
```

AVG RESPONSE IN 4921.14 MILLI SECS
****

Client Configuration:

```
numOfRequests=500
totalDurationOfRequests=5
requestIntervalInMS=10
dataProcessLoop=10
fileIOLoop=2
authorization=N
authentication=N
userId=sid.kargupta
password=wmin30
```

Command Prompt Responses:

```
C:\PhD\project>
C:\PhD\project>runWebServiceClient.bat
- Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.

2031
2109
3328
3391
3484
3890
3750
97 SERVICE REQUESTS MADE IN 5 SECONDS!!
4265
```

4812
4391
4859
5016
4781
4594
5110
5281
5266
5453
5141
6125
5719
5500
5516
5219
5875
5844
6437
6312
5546
6078
6062
6312
6468
6688
7094
6891
7110
6594
6734
6250
6953
6953
7203
6328
6859
7781
6860
6734
6907

7203
7046
7344
7641
7250
7172
7469
7187
7125
7391
7328
7359
7484
6266
7610
7438
7781
7797
7329
7609
7407
7407
7469
7797
6375
8250
8000
7515
8140
7406
7562
8063
6562
7375
7735
8469
6703
7703
7484
7719
7578

8079
7672
7640
6953
7265
7297
7297

AVG RESPONSE IN 6511.91 MILLI SECS
****