Journal of Web Engineering, Vol. 11, No.2 (2012) 146-176


# A NOVEL APPROACH FOR SERVICE PERFORMANCE ANALYSIS AND FORECAST

SID KARGUPTA

*University College London*
*siddhartha.kargupta.10@ucl.ac.uk*

SUE BLACK

*University College London*
*S.Black@cs.ucl.ac.uk*

Received October 10, 2011
Revised April 2, 2012

This research establishes a predictive model to forecast the impact on service performance for changes to the underlying activities of the service's components. It deduces a relational model between a service's performance, its application component latencies and the request load. The major challenge the IT industry is currently facing with the cost associated with repeated performance testing to modify live systems has been addressed. The notion of implicit Operation Impedance gradient ($I_G$) and Operation Potential (V) in Service Provider-Consumer contracts has been introduced. This work establishes that '$I_G$', which impacts the overall Operation Performance (P), is influenced by the underlying *application* components' activities in *distinct* patterns. A high-level *runtime* abstract model is empirically deduced between '$I_G$', 'V' and 'P' by applying established mathematical techniques. Model based *indicative* values of some features are *computed* and associated with the *actual* empirical values of other features against various system configurations. Appropriate regression types are applied to enable trend extrapolation/interpolation. The datasets affirmed effectiveness of the model to assess impact of modifications to the underlying *application* components on the operation's performance *without* repetitive full scale *external* performance/benchmark testing. This also enables fine tuning of application components to *retrofit* prescribed Quality of Services. To address real life applications, this paper describes a Matrix based technique used for the assessment of changes to multiple types of application component activities simultaneously. The method of calibrating the Matrix aided model has also been discussed briefly.

*Key words*: SLA – Service Level Agreement, SOA – Service Oriented Architecture, SLM – Service Level Management, QoS – Quality of Service, ART – Average Response Time
*Communicated by*: G.-J. Houben & G. Rossi

## 1 Introduction

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?"

It is evident from our industrial experience and the corporate reports [3, 4, 5, 6], that a major challenge the Information Technology (IT) industry is currently facing is the cost associated with repeated performance testing of live systems after modification. If performance does not conform to prescribed SLA, a cycle of design, implementation and testing is repeated resulting in significant financial impact to large organizations with complex systems. Industry scale programmes and projects are affected adversely from a resource, time and cost perspective.

This research aims to address the above mentioned critical problems faced by the IT industry today. It explores some generic, application level, bottom-up methods to assess during development the impact of modifications to the application component level activities on the system's performance and other non-functional features. The research has established an empirical relational model between a service's performance, its application components and the service load [1, 2]. It has been established that changes to the underlying activities in the application components of a service impact the service performance. Each of these different types of "*atomic*" activities impacts the performance in particular patterns. We have also worked on a matrix based predictive model to forecast the service performance for simultaneous changes to multiple types of underlying application component activities.

The rest of the paper has been structured under the following sections:

*Background* – this section explains the background of the problem briefly. It describes the motivation for this research to be undertaken.

*Literature Review* – This section describes some relevant research that had already been undertaken. The past research have been grouped under the following categories: Service Performance Evaluation and Optimization, Service Performance Management, Service Performance and SLA, Quality of Service Analysis

*Current Gaps* – In this section we have highlighted the aspects which have remained unaddressed in the context of the identified problem at hand and the research that had already been undertaken.

*Proposed Methodology and Design* – This section describes the empirical approach taken to establish a formula-aided, pattern-based approach for Service Performance analysis and forecast.

*Implementation* – In this section the actual Java implementation of the Service Framework developed to resemble a Service Provider – Service Consumer scenario has been described.

*Deducing the high level, abstract runtime model* – We describe the iterative experiments performed to establish empirically a high-level *runtime* abstract model by applying established mathematical techniques.

*Conclusions and Future Work* – This section summarizes the conclusions from our empirical findings and mentions about the future work that we intend to carry out.

## 2 Background

Due to ever changing business requirements, the underlying software systems supporting evolving business requirements need to be modified in tandem [7, 8]. We witness systems and services being built and modified based on the knowledge, experience and occasionally intuitions of the architects

and designers. Systems are modified or enhanced without prior knowledge of the impact of such modifications. The effect of such modification on the different features may remain unclear and the impact is realized only after implementation. By that time organizations will already have spent a significant amount of time, effort and money [8, 9, 10]. Any adverse effect that has to be addressed may involve substantial re-work. From our industrial experience we have also observed that under these circumstances, at times the engineers have to engage in a trial and error exercise to achieve the service level requirements. This is a significant challenge that the IT industry is facing today.

A system may be made up of one or more application level components [1, 2]. For example, a typical Payroll System may comprise the following: data capture component, pay calculation component, tax calculation sub-component, a National Insurance lookup sub-component, a data access component to interface with the backend databases and possibly a payslip printing component for writing the payslips to a file and sending those to a printer. Due to some changes to the business requirements the Payroll System may be required to be modified and augmented. To address this requirement, we may need to modify one or more than one of the underlying activities of the application components mentioned above. Any change to the application level components will potentially have an impact (positive or negative) on the system's performance [1, 2, 3]. Hence, prior knowledge of the degree of impact on the system's performance will facilitate the following:

1. Architect and design the system's enhancements properly
2. Reduce development and testing time as there will be prior knowledge of impact on performance
3. Build application components to conform to the Service Level Agreements
4. Minimise overall development and testing costs

To achieve the above, it is important to understand how modifications to the different types of application components' activities impact the overall system performance under a given load condition. As one or more than one type of activity of the component(s) may be required to be modified, the knowledge of both of the following is required:

1. Possible impact of modifying every *individual* type of underlying activity of the application components on the overall system performance under a given load condition
2. Possible impact of simultaneously modifying *multiple* types of underlying activities of the application components on the overall system performance under a given load condition

In view of this, there is a distinct need for a method, which will facilitate analysis and prediction of the impact of modifications of the underlying system component's activities on system performance. In order to address this significant requirement of the IT industry, this research focuses on empirically establishing a formula aided, pattern based approach for Service Performance analysis and forecast. The research initially establishes a relational model between Service Performance, Service Impedance Gradient and Service Load. This model is then used to extract the patterns in which the underlying *individual* activities of the application components impact the overall service performance. In real industrial scale systems typical enhancements involve multiple types of underlying activities of the application components being modified simultaneously. Hence, this research has also developed a matrix based predictive model to forecast the service performance for simultaneous changes to *multiple* types of underlying activities of the application components.

## 3 Literature Review

To understand and assess the research already undertaken in the system / service performance, service performance management and forecast of performance due to component modifications, this research has performed exhaustive review of the previous work in this space. Apart from the review of work undertaken towards performance measurement and forecast, the review is broadly organised into four categories: 1) Service Performance Evaluation and Optimization, 2) Service Performance Management, 3) Service Performance and SLA and 4) Quality of Service Analysis.

Significant research has been carried out towards measuring and predicting throughput, response time and congestion using queuing network principles. In particular, ways to model, analyze and plan for web performance problems have been illustrated in detail [11]. High performance website design techniques involving redundant hardware, load balancing, web server acceleration and efficient management of dynamic data have been discussed [12]. Methods are devised for dynamic selection of services based on user specified preferences and to predict performance of component based services depending on the underlying technology platforms [13, 14]. In [15, 16], different methods of generating performance models and prediction have been discussed. An assembler tool and a methodology to automatically generate performance models for component based systems have been explored. A performance prediction approach comprising empirical performance result gathering on Commercial Off The Shelf (COTS) middleware infrastructure, a reasoning framework for understanding architectural trade-offs and relationships to technology features and predictive mathematical models to describe application behaviour on the middleware technology has been investigated. Different model-based software performance prediction approaches have been classified and evaluated in [17]. Queuing network based methodologies, Architectural Pattern based methodologies, software performance analysis through Unified Modelling Language (UML) descriptions and other approaches have been discussed. Further research [18, 19, 20] has explored methods of component based performance evaluation with *top-down* approach focusing on inbound workload, profiling, software containers, UMLs and transactions.

Other related areas explored are Service Performance Evaluation and Optimization, Service Performance Management, Service Performance and SLA and Quality of Service Analysis. The sections below briefly discuss the research work that has been undertaken in these areas and the issues that remain to be addressed in the context of the Problem Statement:

*Service Performance Evaluation and Optimization*

Much research has been undertaken to devise ways of measuring efficiency and performance of services. However, in a Service Provider – Consumer scenario, we lack formula-aided techniques to predict the impact on some of the key features of a system for varying any other feature of the system. It becomes difficult to assess and forecast the impact on performance from a service consumer's viewpoint due to any change to the service providing system. In [21, 22, 23], some attempts have been made to study Web Services performance and its relationship with services availability. Web Services performance is a critical success factor for a service contract between any service provider and consumer. But research highlights the lack of required functionality in existing frameworks to measure service execution performance. The current implementations of Web Services performance have been evaluated and compared with alternative technologies. Present Web Services performance behaviours have been discussed and a simple performance model that could be used to estimate Web Services latencies has been developed in [21]. In [22] an Internet Data Centre (IDC) has been used as an example to illustrate the relationship between performance and availability. It was illustrated that an

IDC should provision enough capacity and redundant resources to ensure that it could meet its performance and availability SLAs. It was stated in [23] that the present web service frameworks did not include the functionality required for web service execution performance measurement from an organization's perspective. As such, a shift to this paradigm of Web Services was at the expense of the organization's performance knowledge. The research introduced an approach to reclaim and improve this performance related knowledge for the organization. This was achieved by establishing a framework that enabled the definition of web services from a performance measurement perspective, together with the logging and analysis of the enactment of web services. This framework utilized web service concepts, Decision Support System (DSS) principles, and agent technologies, to enable feedback on the organization's performance measures through the analysis of the web services.

[24, 25, 26, 27] elaborates on the role of performance evaluation and the significance of automated performance tuning in computer engineering. Evaluation and comparison of performance and recovery time in web services infrastructures based on fault injections have also been discussed in depth in these papers. In [24], Fortier and Michel present with an overview of performance evaluation methods, performance metrics and evaluation criteria. They discuss computer data processing hardware architecture, fundamental concepts and performance measures, general performance measurement principles and system performance evaluation tool selection. Use and analysis of computer architectures, database systems performance analysis and analysis of computer networks components are also discussed. In [25] the theory of "Active Harmony" as a way to automate performance tuning has been advocated. The authors apply the Active Harmony system to improve the performance of a cluster based web system. Performance improvement could not easily be achieved by tuning individual components for such a system. Experimental results showed that there was no single configuration for the system that could perform well for all kinds of workloads. By tuning the parameters, Active Harmony helped the system adapt to different workloads and improve the performance by up to 16%. An approach, based on fault injection, for the evaluation and comparison of performance and recovery time in web services infrastructures is described in [26]. Fault injection is used to validate specific fault handling mechanisms and to assess the impact of faults in actual systems, allowing estimation of measures such as performance in the presence of faults, error detection coverage and recovery time. To compare the performances of different Java based Web Services toolkits, in [27] the authors present an open source utility to automate the performance comparison process. The main purpose of the research is to present a utility named Java Web Services Performance (JWSPerf) to automate the performance evaluation of Java Web Services and facilitate the choice of the "ideal" toolkit to implement an application. The utility supports Java Web Services toolkits – Apache Axis, Java Web Services Developer Pack and Systinet Server for Java.

*Service Performance Management*

There has been substantial research in [28, 29, 30, 31, 32, 33] to explore techniques of Service level monitoring. In reality, however, application developers often find it more convenient to monitor and analyze application level outputs rather than system resource or service level diagnostics. Hence, exploring *generic*, application level methods to assess the impact of modification to the activities at the application component level on other non-functional features is helpful. The notion of a model aided method to facilitate the above through *visual patterns* remains unexplored in the undertaken research. A high level abstract *runtime* model for the key aspects of a Service Operation such as performance, load and latency remains to be discussed. Typically, we still have to recourse to performance/benchmark testing of the whole system for the impact analysis of application component modifications.

A scalable service level monitoring methodology to assess user satisfaction without injecting any measurement traffic is described in [28]. Specifically, web throughput was suggested as a service level metric and possible ways to measure it was outlined. The authors discuss the advantages of passive observations of actual user activity. A statistical data analysis method is proposed that analyzes passive throughput measurements and quantified user satisfaction / dissatisfaction and the confidence that the provider might have on the collected data, i.e. data reliability.

A meaningful pattern for the web throughput was obtained after satisfying the above requirements with numerous passive measurements. It was advocated in [29], that an Argumentation theory, implemented through a set of software agents that reason about Web Services, could improve Web Services' performance through the notion of Web Service Communities. To facilitate and speed up Web Services discovery, Web Services with similar (or equivalent) functionalities could be grouped into communities. Although Web services in a community had a common functionality, they might have distinct non-functional properties. Additionally, a community might describe a desired functionality without explicitly referring to any concrete (or pre-selected) Web service that would implement the functionality at runtime. The authors have discussed in details how to enrich software agents to apply logic based reasoning and argumentation to define the interaction mechanisms for peers in these communities.

A quantitative performance evaluation of Web Services Security (WSS) overhead was conducted in [30]. Based on the evaluation, the existing web services performance model was extended by taking the extra WSS overheads into account. The extended performance model was then validated on different environments with different message sizes and WSS security policies. In [31] an algorithm is described that drew on context mobility elements, such as the user's travel direction and speed, to form personal service areas. As Web service performance depends on the underlying databases, a layered caching scheme for storing environmental data to improve response time was also developed. [32] analysed the impact of an IT Service Provider's process capabilities on the performance of the service it provides. A framework to improve web services performance based on context-aware communication has been proposed in [33]. Two key ideas were introduced to represent a client context - (1) available protocols that the client could handle and (2) operation usage that showed how the client used Web Service operations.

[34, 35 and 36] emphasize Service Oriented Architecture (SOA), its potential flaws leading to degradation of performances and the possible ways to address those issues. An overview of the evolution of Service Oriented Architecture from other technologies such as object oriented programming and distributed computing is provided in [34]. In [35], the author attempts to establish a SOA roadmap unveiling possible traps and pointing out the flaws in the SOA approach. The SOA approach was reviewed critically and the different sections affected within the enterprise were examined. The key middleware technologies for realization of SOA were analysed in [36]. It presented a detailed performance analysis with overhead analysis and identification of optimizations of the web services. The research contributed to the understanding of functional and performance aspects of distributed middleware technologies for realization of SOA.

*Service Performance and SLA*

Methods have been explored to improve service performances and to maintain service models adherence to SLAs. The key focus of [37, 38, 39, 40] has been Service Level Agreement (SLA) guarantee model. Techniques have been discussed which systems might adopt to conform to SLAs. Ways to improve the Web Services runtime environment and the architecture to implement an

effective Web Services performance management system have been explored in detail. In [37], the authors present a SLA-based Web services performance guarantee model to improve Web services runtime environment and expatiated on the key techniques of realizing the model. An architecture and description of a prototype implementation for a performance management system for web services that supports SLAs have been presented in [38]. The authors designed and implemented reactive control mechanisms to handle dynamic fluctuations in service demand while keeping SLAs in mind. The roles of SLA and Service Level Management (SLM) are discussed in [39]. It was suggested that SLM should be more than just a reporting tool. It should be used to identify and remedy process problems in service delivery. [40] discusses the different aspects of SLM and how performance benchmarking creates value by focusing on key performance gaps, creating a consensus to move IT forward and making better decisions from a larger base of facts.

All of the above research focus on SLA in particular but does not attempt to establish any relationship between the key aspects of the service contract between a service provider and a service consumer. These aspects such as performance, load and the underlying service activities have significant influence on SLA.

*Quality of Service Analysis*

Some research has been undertaken in the area of Quality of Service (QoS), of which service performance is a key factor. [41, 42, 43, 44, 45, 46] focus on extending existing QoS models to propose enhanced concepts of evaluating QoS from multiple metrics. Techniques have been discussed to meet the demand for faster and more efficient access to the services to provision QoS. Means of actively monitoring the QoS of services have also been discussed. In [41], an existing QoS model was extended by adding new attributes that reflect performance of services and provide the client a dynamic, on demand service performance prediction. In this way a client might be more capable of finding the best service based both on his/her preferences and on the service performance estimation. An extended web service QoS model has also been proposed in [42]. Web service QoS metrics were evaluated dynamically according to the service context and the overall QoS was evaluated from multiple metrics according to a configurable fuzzy synthetic evaluation system. A QoS requirement description model has been defined to express user's flexible demands on service's performance. An interactive web service choice-making process has also been provided, which included QoS as a key factor. Techniques that have been developed to meet the demand for faster and more efficient access to the Internet to provision QoS have been discussed in [43]. These techniques included caching, pre-fetching, pushing and replication. The concept of developing an ontology for Quality of Service (QoS), also known as QoSOnt has been discussed in [44]. Particular focus was given to its application in the field of service centric systems. QoSOnt was developed to promote consensus on QoS concepts, by providing a model which was generic enough for reuse across multiple domains. As well as the structure of the ontology itself, an example application - Service QoS Requirements Matcher (SQRM) – was also discussed. This application was used to highlight some of the advantages of the ontology including standardisation and the level of machine understanding of QoS specifications which could be achieved. To actively monitor the QoS of Web Services at runtime, a Web Service QoS broker system was designed and developed in [45]. With this information a user could select a Web service best suited for his/her needs. Availability, performance, and reliability were the metrics used for QoS monitoring. Another simple but scalable system to verify Quality of Service in a differentiated services domain was designed and evaluated in [46]. The system used a distributed edge-to-edge monitoring approach with measurement agents collecting information about delays, losses and throughput – reporting to a Service Level Agreement Monitor (SLAM). The SLAM detected potential service

violations, bandwidth theft, denial of service attacks and flagged the need to re-dimension the network domain or limit its users.

In the QoS related research described above, a "top-down" approach of assessing the quality of service due to application of external load was adopted. However, there has not been much research to forecast the possible impact on quality of service due to changes to the system internals. Such a "bottom-up" approach to systems analysis and forecast of performance remains to be explored.

## 4 Current Gaps

None of the above research discusses any model aided technique to predict the impact of modifying the application component level activities of a system/service on system's performance. It is very difficult to assess and forecast the impact on performance from a service consumer's viewpoint due to any change to the service providing system. At times, after modifying the system and observing the degradation of performance, changes have to be rolled back and re-implemented. The entire change process has to be followed again. As a consequence, organizations are required to pay astronomically high recurring Opex bills towards modifying, augmenting and performance testing their systems on a regular basis. All of these highlight the need for a method, which would facilitate analysis and prediction of the impact of modifications of the underlying system components on system performance.

There are gaps existing today for assessing the impact of application layer modifications on the system/service's performance. Currently, there is no derived model between the key aspects of a Service Provider – Consumer contract, which are performance, load and the underlying service activities. In the industry, load/performance testing only happens at the end of application development when adverse results may lead to undoing and redoing a lot of the application development. Every time systems undergo change, load/performance tests are performed by human resources. This current approach of *repetitive* testing is resource, time and cost intensive for any organization. There is also lack of functionality in existing frameworks [21, 22] to measure service execution performance from the *application* layer. As explained in the Literature Review, application developers work at the application level and not at the system level. Hence, monitoring systems resource utilization does not help application developers. At times, they may not even have the systems level expertise to monitor systems resource utilization.

Due to the above, there is little or no way today for application developers to receive rapid feedback of how changes will alter system performance [8, 9].

## 5 Proposed Methodology and Design

This research focuses on empirically establishing a formula-aided, pattern-based approach for Service Performance analysis and forecast. Data collected from the empirical results will be used firstly to establish a relationship between service performance, service load and the relevant underlying application component activity delays. This relational model will then be used along with further data collected from more experiments to plot and derive graph functions of service load and the delays introduced by different application level activities. These plotted graph functions will facilitate visualizing the patterns in which the service load and the different application level activity delays affect the service performance. These graph functions (or graphical patterns) will then be used for future performance forecasts through extrapolation and interpolation.

***'P', '$I_G$', 'V' - Definitions and rationale for use:***

This research explores a novel approach for Service Performance analysis and forecast. Hence a Service Operation Performance ('P') is considered a variable in the proposed model. 'P' is the measure of Service Operation's performance under a given load. The lower the response time, the higher is 'P'. Hence, 'P' is computed as the reciprocal of the Average Response Time (ART) of the Service Operation.

The service load (i.e. number of requests hitting the service per unit time say second) affects service performance [9]. Hence, Service Operation Load Potential ('V') is considered as another variable in the proposed model. 'V' has been defined as the differential between the maximum request load (per second) the Service Operation can cater to maintaining QoS (Service Operation's "stress point") and the Service Operation's contractual request load (per second).
The time taken for all the application level activities cumulatively introduces latency or impedance to a service's performance in some way. Hence an overall Service Operation Impedance gradient ('$I_G$') is considered the third attribute of the model. '$I_G$' has been defined as the *runtime* composite gradient of all the activity delays of the components supporting the Service Operation.

***Service Framework Prototype – Design and purpose:***

To create a prototype, a Java based Service Framework is designed to fulfil the following purposes:

1) For Service Operations, empirically deduce a high level abstract runtime model for Service Operation Load Potential 'V', Service Operation Performance 'P' and overall Service Operation Impedance gradient '$I_G$'. Network latencies (inter-component and Provider-Consumer) contribute to '$I_G$' as well.

2) Decompose the application components supporting the Service Operation into atomic Delay Points i.e. activity nodes in the application component which introduce some delay to the response.

3) Compute model based *indicative* values of '$I_G$' and extract its distinct variation patterns against variability of *actual* component Delay Point impedances and other non-functional features. Use Least Square Fitting (LSF) [47] and appropriate regression types to derive the "best-fit" functions from the collected data set. The function graphs would provide the *graphical patterns* to enable bottom-up and top-down projections of the non-functional features related to service load and performance.

The research introduces the notion of implicit '$I_G$' and 'V' in Service Provider-Consumer contracts. This work establishes that '$I_G$', which impacts the overall 'P', is influenced by the underlying *application* components' activities in *distinct* patterns. A high-level *runtime* abstract model has been empirically deduced between '$I_G$', 'V' and 'P' by applying established mathematical techniques. Model based *indicative* values of some features are *computed* against variability of the operation's components. Lookup datasets against different system configurations are created to associate these *computed* values to the *actual* empirical values of other features. Established mathematical techniques applied with appropriate regression types to enable trend extrapolation/interpolation. The datasets/patterns affirmed effectiveness of the '$I_G$' based model as a means of *decoupled, bidirectional* i.e. top-down and bottom-up impact assessment of modifications to the operation's underlying *application* component activities on 'P' (with 'V' constant) or 'V' (with 'P' constant) *without* repetitive full scale *external* performance/benchmark testing. This also enables fine

tuning of application components to *retrofit* prescribed QoS. The project has also established a matrix based predictive model to forecast the service performance for simultaneous changes to multiple types of the underlying application component activities. Means of model calibration have also been explored.

Service Operations of a Service Provider are catered by underlying application components laid on top of system components. The application components are often modified due to changes in business requirements while the underlying system remains the same. Extending on the fundamentals of some of the previous work done, this research explores one level of abstraction from system resources to application components. It verifies a higher level pattern (extracted from plotted graph functions) based projection of non-functional features like performance, load etc. of a Service Operation for modifications to the different activities of the supporting application components. The Service Operation's application components are decomposed into atomic activities or Delay Points like in-memory Data Processing, File I/O, Database Interaction, XML processing etc., which interface with the system resources (both Queue and Delay). Each of these Delay Points introduces latency to the service execution and contributes to the overall Service Operation Impedance '$I_G$'. The project tries to establish that '$I_G$' is a *function* of the total delay or latency for each type of Delay Point across all the supporting application components i.e.

$$I_G = \boldsymbol{f}(\sum L_{DLPi}) \text{ [i=1 to n]}$$

where $L_{DLP1}$ is the latency (or delay) introduced by a particular type of Delay Point of Component1. This function represents the distinct *pattern* by which the total latency introduced by the particular type of Delay Point across all the components 1 to n ($\sum L_{DLPi}$ [i=1 to n]) influence '$I_G$' and hence 'P' and 'V'.

*Atomicity* of Delay Points is very important as Delay Point types determine their nature of system resource usage, which then manifests as the Delay Point impact pattern. Delay Points should not overlap. '$I_G$' acts as a connector between the Service Operation's internal application Delay Points and external non-functional features. This research focuses on variations to application component Delay Points instead of inbound workload.

## 6 Implementation

To assess the impact of change of the underlying application component activities and request load on the Service Performance, a Service framework is required which will resemble a real life scenario. In the framework, a Service Consumer will request a Service Provider for a particular type of service. The framework is required to be configurable to enable spawning of multiple simultaneously requests, varying the number of requests thus varying the request load and configuring the application level Delay Points. To achieve this, a prototypical Service framework comprising of a Web Service front end with other lower level backend services was created. A configurable Web Service client was developed to serve as the Service Consumer.

As shown in Figure 1 below, the Service Provider – Consumer framework comprising of a multi-threaded Service Consumer, a consumer facing Service Provider (Web Service), other backend services and some utility components have been created. For the purpose of the experiments, some illustrative Delay Points with activities such as Database Interactions, Data Processing, File I/O, Request Authentication and Request Authorization involving XML parsing etc. have also been created.

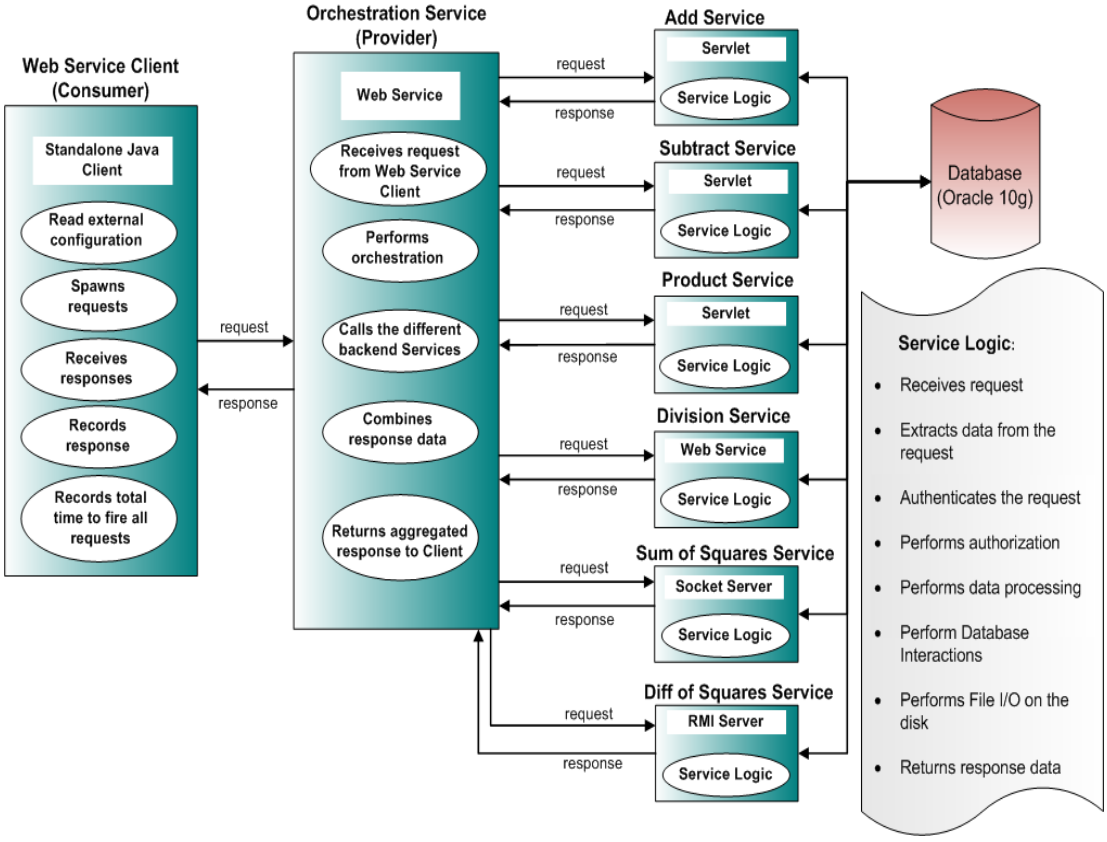


Figure 1: Logical model of the Service Framework

To increase the precision of the model and *standardize* request resource requirements, partitioning of the request load is achieved by constraining the model and method to Service Operation level. Different Service Operations from the same Provider may have different resource requirements.

The Service Consumer framework comprises a multi-threaded Web Service client which implements a Runnable Interface. The Service Provider framework comprises a *Facade* Orchestration Web Service, which orchestrates between the different lower level backend services. The backend services perform different types of data operations. The *Add Service*, *Subtract Service* and *Product Service* extend *HttpServlet*. The *RMIServer* implements a Difference of Square Interface (*DifffSquareIntf*). The *SumOfSquare Service* runs as a *Socket Server* while the *Division Service* has been implemented as another *Web Service*. LatencyActivities is the object responsible for performing all the processing activities to introduce latencies to the response processing. All the services have dependencies on LatencyActivities, which implements the Latencies interface. For authentication and authorization of requests through XML data parsing, two objects namely AuthenticationHandler and AuthorizationHandler are used. These two objects extend the DefaultHandler object.

The Web Service client is driven by an external configuration file. It spawns service requests as per the configuration file and calls the main Orchestration Web Service. This is a Facade which in turn

calls the different back end services one after the other. Depending on the type of service, its processing method is invoked. The service extracts the request data from the input and performs the following activities:

1. *Gets a Singleton instance of the LatencyActivities object* – For overall efficiency, the LatencyActivities class has been designed as a Singleton i.e. only ONE object of the class will be instantiated per virtual machine. The service gets the singleton object and calls the authentication and authorization methods on the LatencyActivities object.

2. *Performs authentication of the request* – The LatencyActivities object has an operation which performs authentication on incoming requests. It reads an XML configuration file and authenticates the User Id and Password supplied with the request.

3. *Performs authorization of the request* - The LatencyActivities object has an operation which performs authorization on incoming requests. It reads an XML configuration file and authorizes the request to perform certain activities.

4. *Performs database interactions* – Upon successful authentication and authorization, the service interfaces with an Oracle 10g database for some read and write operations.

5. *Performs some data processing* - The service then calls the data processing methods on the object, which does some in-memory data processing.

6. *Performs File I/O* – After data processing, the service does some file read/write on the file system.

7. *Performs the relevant service* – At the end, the service calls the appropriate method like addNumbers, subtractNumbers etc. based on the type of service

8. *Returns data back to the calling Service* – After all the backend services are called and processing done, the response is sent back to the Orchestration service. Upon receipt of this response, this service in turn sends the response back to the Web Service Client.

## 7 Deducing the high level, abstract *runtime* model for P, $I_G$ and V (*PIV* model)

Using the prototypical service framework described in Section 5 and 6, iterative experiments were performed to establish empirically a high-level *runtime* abstract model between '$I_G$', 'V' and 'P' by applying established mathematical techniques.

Tests were run by gradually increasing the request load to the Service Operation. Under a particular load configuration, multiple service requests (the number depending on the load configuration) were spawned by the client. The Average Response Time (ART) for all the requests provided an indicative measure of the response time for that particular load condition. Initial experiments showed that the variation of ARTs was *minimal* across test runs for the same load configuration. Hence '5' was considered as a reasonably good number for test run iterations under one given load condition. So, 5 test runs were conducted under a given load condition and the Average Response Time (ART) during each of the runs was recorded. As the impact of varying load conditions was being determined, the *highest* ART was recorded.

Tests were performed against different increasing load conditions. From the Service Consumer, service requests were spawned for a fixed duration of time (5 seconds in our experiment). To increase the number of requests (i.e. increase the request load), the time interval between requests being generated was reduced in steps of 100 milliseconds initially and then 10 milliseconds. Every step represented a particular load condition for which 5 test runs were repeated. The data obtained demonstrated a *finite system queue graph* [9] and reached saturation level after the 9th load condition.

Details of the experiment steps are shown in the flow charts in Figure 2 and 3 below:

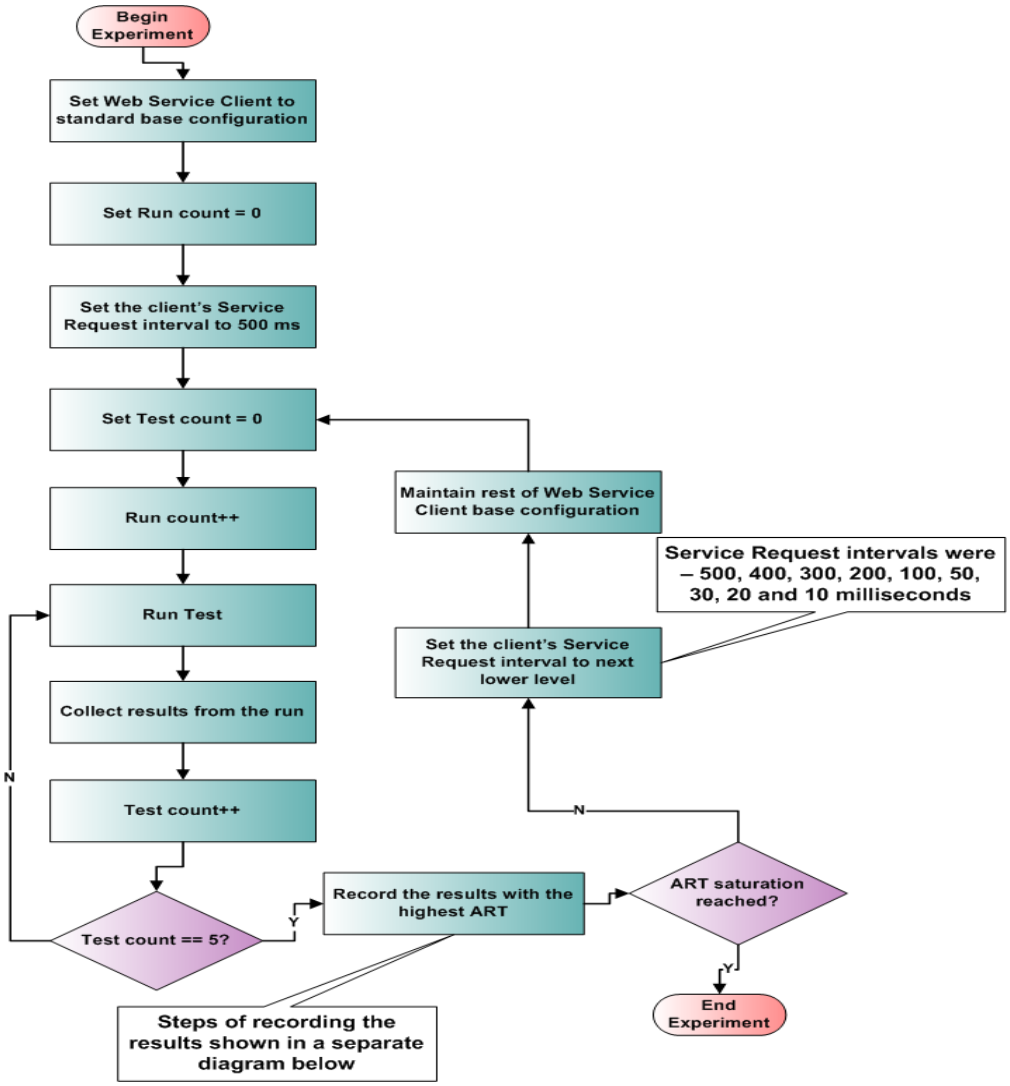


Figure 2: Flow Chart for tests to determine relationship between V and P

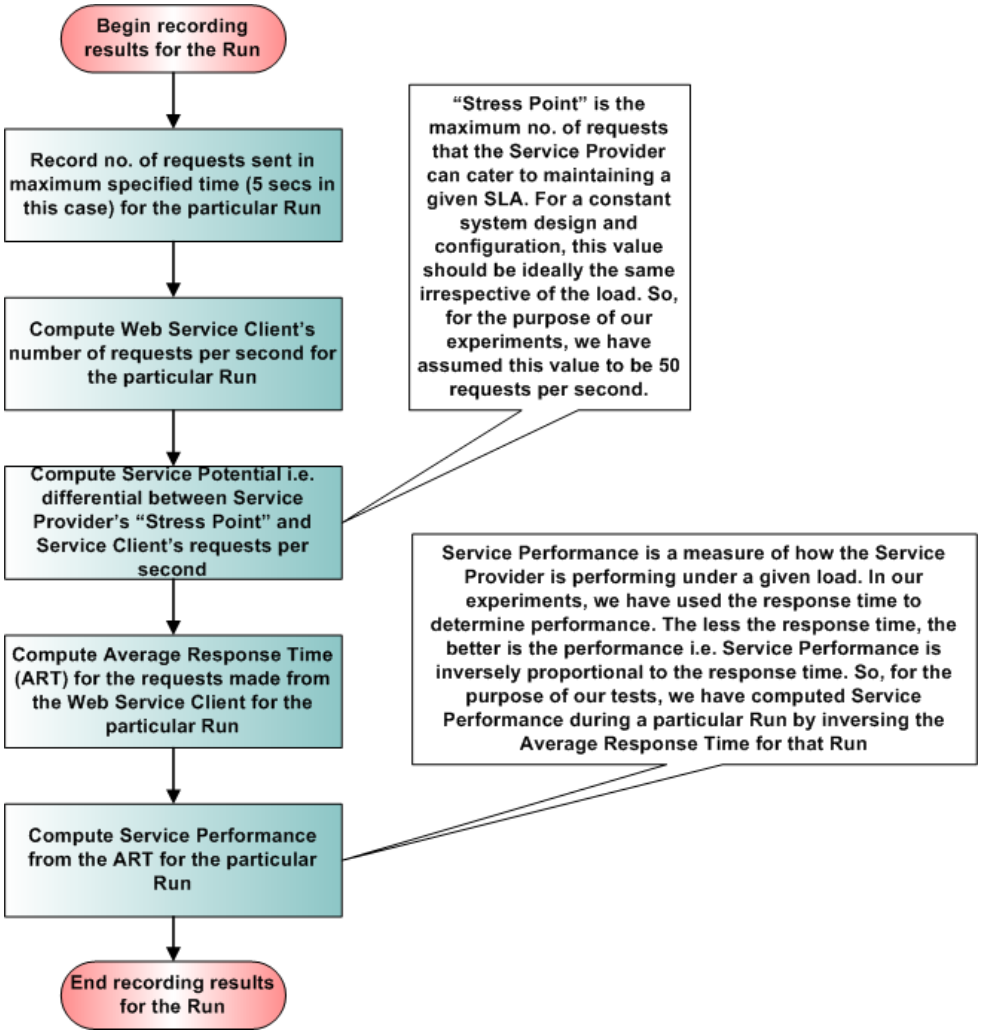


Figure 3: Flow Chart to record the Average Response Time of each run

As shown in Figure 4, the data obtained from the experiments to deduce the high level, abstract runtime model associating 'P', 'V' and '$I_G$' demonstrated a typical finite system queue graph [9].

As the request load was gradually increased and the Service Potential gradually decreased to nearly 45, the abrupt change occurred as the server utilization approached 100% [9]. At this point the throughput of the server gradually approached its maximum throughput.

Assuming a *stress point* for the Service Operation, we observed a typical *finite queue* system curve for 'V' versus 'P'. Accepting approximation error, for simplifying the model, *Piecewise Linear* model is applied to divide the 'V' values into 3 ranges (or *bands*), each with a *linear regression* (affine form)

as the best fit for 'P'. Direct proportionality between 'P' and 'V' is considered for each of the three ranges/bands of 'V' values:

$$P = IV + c$$

where I is the *constant of proportionality* with I and *c* band specific.

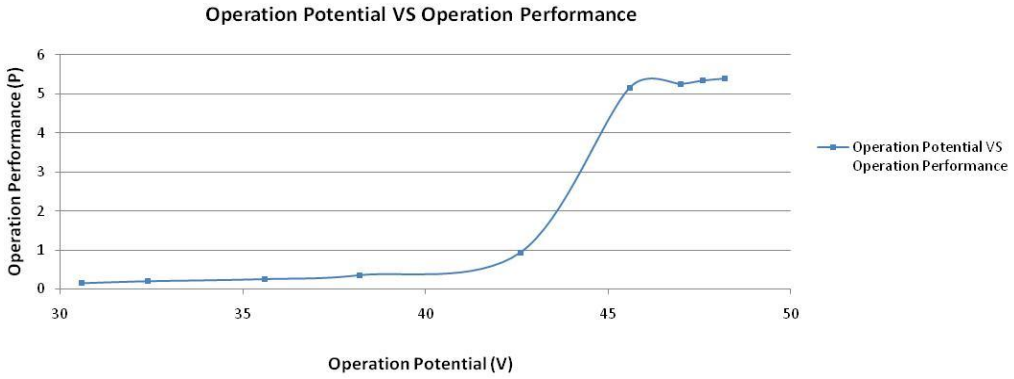


Figure 4: Empirical Data Graph of P for varying V

At a *given* time $T_1$, for requests to the *same* Service Operation, the request/process type, system configuration, resource requirements and contract load condition will be ideally the same. Today, services are run on multi-core, multi CPU servers. So, for simplicity, we assumed Multi-Processor Single Class Queuing Network (open or closed) model approximation. 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* [9]. Under light load, the Residence time ($R_i$') is *D* (proven) and under heavier load, it will be dominated by the single resource queue:

$$R_i' = V_i W_i + D_i$$

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* [9]. As the requests are to the *same* Service Operation, applying all the above constraints, $D_i$ and $V_i$ will ideally be same for all requests. As we used the ART 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*. The experiments had co-located components with local calls between them. Also, only formal Service Contracts are in scope with dedicated, controlled network traffic and *not* any random service access over public network. Hence, at *runtime*, no unpredictable fluctuation of network bandwidth or latency is assumed. Average resource usage effect of other Service Operations on requests of the tested Service Operation is assumed. With *all* the above constraints, we assumed *consistency* of overall impedance for processing requests to the *same* Service Operation at $T_1$ for a 'V' *band* and mapped the *runtime* Operation Impedance to the proportionality constant '$I_G$'.

### *7.1 Data Processing Delay Points - Pattern Extraction and Validation*

Some illustrative components are created with Database Interactions, Data Processing, File I/O, XML Processing and other Delay Points. Keeping the rest of the configuration constant ('V' kept positive),

the Data Processing Delay Point intensities of the components were incrementally varied. Empirical data for *actual* overall 'P', *computed indicative* values of overall '$I_G$' based on the model:

$$P = I_G V + c$$

for the *relevant* 'V' *band*, the *actual* average Data Processing Delay Point impedance ($I_{DP}$) and the Data Processing Impedance Factor ($IF_{DP} = I_G/I_{DP}$) were recorded. The following data models '$I_{DP}$' versus '$I_G$', '$I_{DP}$' versus '$IF_{DP}$' and '$I_G$' versus 'P' showed *distinct* trends in variation, which were consistent but *not* purely linear. Accepting approximation error, for simplicity, LSF for *Linear*, *Exponential*, *Polynomial* and *Power* regression types and *Piecewise Linear* models were verified. For '$I_{DP}$'($x_i$) versus '$I_G$'($y_i$), pattern line with *Polynomial* regression of 3rd order was the best fit:

$$y_i = -91766x_i^3 + 2086.3x_i^2 - 19.139x_i + 0.1043$$

For '$I_G$'($x_i$) versus 'P'($y_i$), a linear regression pattern line was the best fit, which confirmed the piecewise linear model:

$$y_i = 41.652x_i + 0.0727$$

For '$I_{DP}$'($x_i$) versus '$IF_{DP}$'($y_i$) pattern line with *Power* regression was the best fit:

$$y_i = f(x_i) = Ax_i^B \text{ where } B = b, A = e^a, a \text{ and } b \text{ are LSF coefficients}$$

| Data Process Loop | Total Service Requests | Total time for requests in milliseconds | Service Requests per second | Service Stress Point (Requests per second) | Service Potential (V) in Requests per second | Average Orchestration Service Response Time (ART) in seconds | Orchestration Service Performance (P=1/ART) in seconds$^{-1}$ | Calculated overall Service Impedance Gradient ($I_G$ = P/V) | Average actual Data Processing latency ($I_{DP}$) in seconds | Impedance Factor ($IF_{DP}$ = $I_G/I_{DP}$) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 150 | 19672 | 7.625050834 | 50 | 42.37494917 | 0.255626667 | 3.911954929 | 0.092317631 | 0.000618667 | 149.220309 |
| **3** | **150** | **19890** | **7.54147813** | **50** | **42.45852187** | **0.266953333** | **3.745973084** | **0.088226648** | **0.000934667** | **94.39370375** |
| 5 | 150 | 20358 | 7.368110816 | 50 | 42.63188918 | 0.28688 | 3.485778026 | 0.081764569 | 0.001504 | 54.36473973 |
| 10 | 150 | 21108 | 7.106310404 | 50 | 42.8936896 | 0.39452 | 2.534725743 | 0.059093209 | 0.003318667 | 17.80631031 |
| 15 | 150 | 22265 | 6.737031215 | 50 | 43.26296879 | 0.492813333 | 2.029165879 | 0.046903066 | 0.005601333 | 8.373553792 |
| 20 | 150 | 23343 | 6.425909266 | 50 | 43.57409073 | 0.591713333 | 1.69000755 | 0.038784689 | 0.007706667 | 5.032615331 |
| 25 | 150 | 24469 | 6.130205566 | 50 | 43.86979443 | 0.74885333 | 1.335374979 | 0.030439508 | 0.009857333 | 3.088006405 |

Table 1: Empirical Data for varying Data Processing

Tests are performed to validate the extracted patterns. Results affirmed (with some approximation errors) the *distinct* underlying patterns of variations in '$I_G$' due to changes in application component Delay Points under a given load. From a *projected* value of '$IF_{DP}$' corresponding to a given *actual* '$I_{DP}$', we could also *project* '$I_G$':

$$I_G = IF_{DP} \times I_{DP} + e$$

where '*e*' is the error factor. Table 1 and Figures 5, 6 and 7 present the data obtained, the empirical graphs of '$I_{DP}$' versus '$I_G$', '$I_G$' versus 'P' and '$I_{DP}$' versus '$IF_{DP}$'. Pattern validation is highlighted.

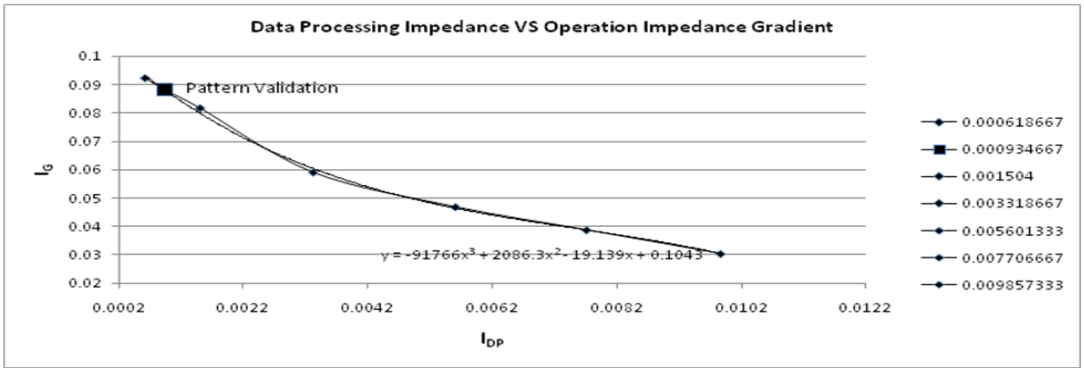


Figure 5: Empirical Data Graph for $I_{DP}$ vs $I_G$ for varying Data Processing

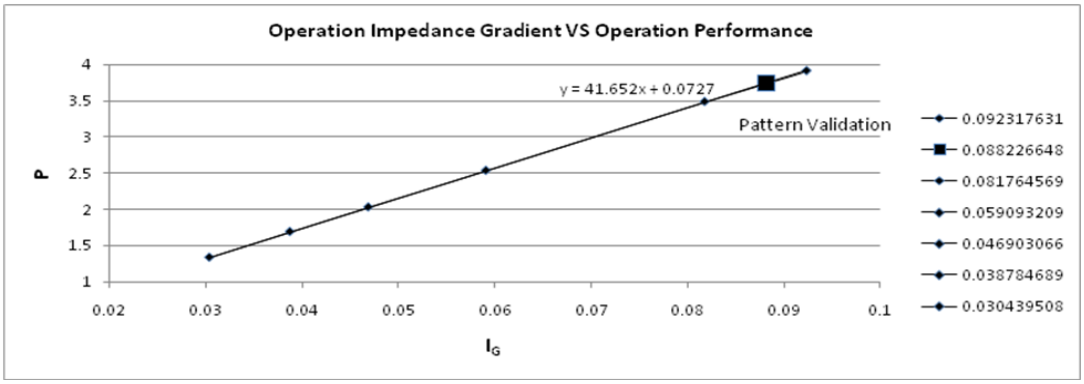


Figure 6: Empirical Data Graph for $I_G$ vs P for varying Data Processing

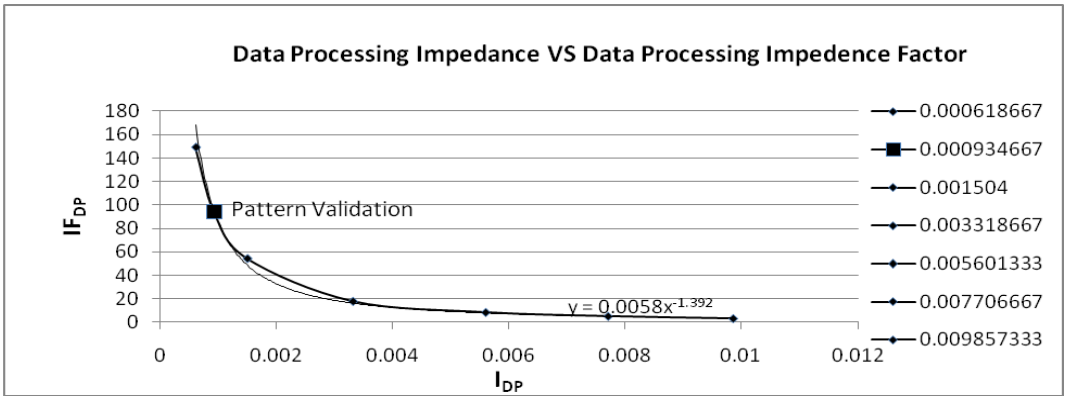


Figure 7: Empirical Data Graph for $I_{DP}$ vs $IF_{DP}$ for varying Data Processing

### *7.2 Database Interaction Delay Points - Pattern Extraction and Validation*

In the same way as the Data Processing Delay Points mentioned above, the Database Interactions Delay Point intensities of the components were incrementally varied keeping the rest of the configuration constant ('V' kept positive). Empirical data for *actual* overall 'P', *computed indicative* values of overall '$I_G$', the *actual* average Database Interactions Delay Point impedance ($I_{DB}$) and the Database Interactions Impedance Factor ($IF_{DB} = I_G/I_{DB}$) were recorded. The data models '$I_{DB}$' versus '$I_G$', '$I_{DB}$' versus '$IF_{DB}$' and '$I_G$' versus 'P' showed *distinct* trends in variation. For '$I_{DB}$'($x_i$) versus '$I_G$'($y_i$), pattern line with *Polynomial* regression of 4th order was the best fit:

$$y_i = 0.1116x_i^4 - 0.4682x_i^3 + 0.7288x_i^2 - 0.5015x_i + 0.1323$$

For '$I_G$'($x_i$) versus 'P'($y_i$), a linear regression pattern line was the best fit, which confirmed the piecewise linear model:

$$y_i = 41.831x_i + 0.001$$

Table 2 and Figures 8 and 9 present the data obtained, the empirical graphs of '$I_{DB}$' versus '$I_G$' and '$I_G$' versus 'P'. Pattern validation is highlighted.

| DB Access Loop | Total Service Requests | Total time for requests in milliseconds | Service Requests per second | Service Stress Point (Requests per second) | Service Potential (V) in Requests per second | Average Orchestration Service Response Time (ART) in seconds | Orchestration Service Performance (P=1/ART) in seconds$^{-1}$ | Calculated overall Service Impedance Gradient ($I_G$ = P/V) | Average actual DB Access latency ($I_{DB}$) in seconds | Impedance Factor ($IF_{DB} = I_G/I_{DB}$) |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 150 | 18953 | 7.914314357 | 50 | 42.08568564 | 6.875533333 | 0.145443263 | 0.003455884 | 0.804137333 | 0.00429763 |
| 6 | 150 | 19109 | 7.849704328 | 50 | 42.15029567 | 7.514593333 | 0.1330744 | 0.00315714 | 0.891390667 | 0.003541814 |
| **8** | **150** | **19312** | **7.767191384** | **50** | **42.23280862** | **8.367553333** | **0.119509247** | **0.002829773** | **1.02896** | **0.002750129** |
| 10 | 150 | 19375 | 7.741935484 | 50 | 42.25806452 | 8.927793333 | 0.112009761 | 0.002650613 | 1.106497333 | 0.002395498 |
| 14 | 150 | 19312 | 7.767191384 | 50 | 42.23280862 | 10.55689333 | 0.094724837 | 0.002242921 | 1.194706667 | 0.001877382 |
| 18 | 150 | 19453 | 7.710892921 | 50 | 42.28910708 | 11.39134 | 0.087785985 | 0.002075853 | 1.337642667 | 0.001551874 |

Table 2: Empirical Data for varying Database Interactions

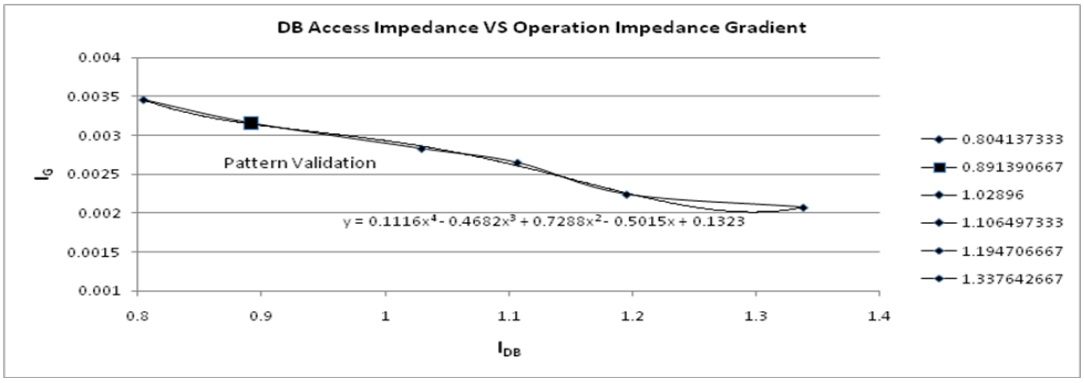


Figure 8: Empirical Data Graph for $I_{DB}$ vs $I_G$ for varying Database Interactions

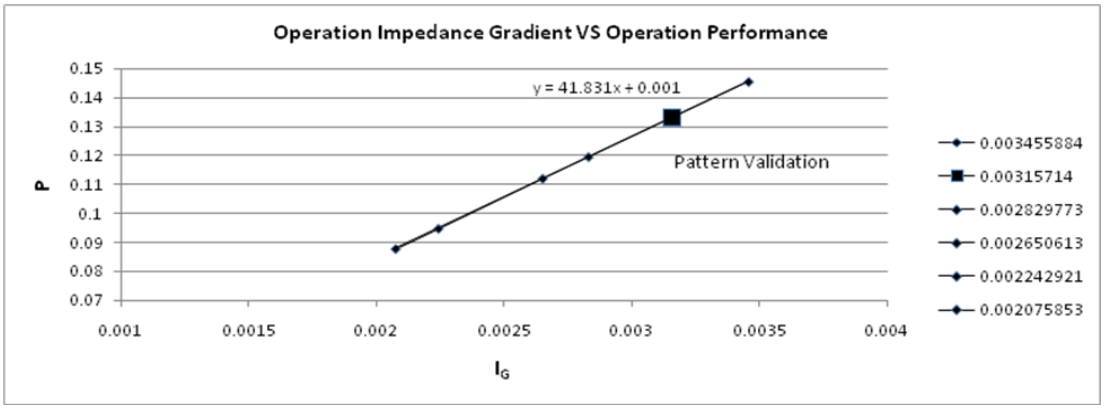


Figure 9: Empirical Data Graph for $I_G$ vs P for varying Database Interactions

### *7.3 File I/O Delay Points - Pattern Extraction and Validation*

In the same way as above, the File I/O Delay Point intensities of the components were incrementally varied keeping the rest of the configuration constant ('V' kept positive). Empirical data for *actual* overall 'P', *computed indicative* values of overall '$I_G$', the *actual* average Database Interactions Delay Point impedance ($I_{FIO}$) and the Database Interactions Impedance Factor ($IF_{FIO} = I_G/I_{FIO}$) were recorded. The data models '$I_{FIO}$' versus '$I_G$', '$I_{FIO}$' versus '$IF_{FIO}$' and '$I_G$' versus 'P' showed *distinct* trends in variation. For '$I_{FIO}$'($x_i$) versus '$I_G$'($y_i$), pattern line with *Power* regression was the best fit:

$$y_i = f(x_i) = Ax_i^B \text{ where } B = b,\ A = e^a,\ a \text{ and } b \text{ are LSF coefficients}$$

For '$I_G$'($x_i$) versus 'P'($y_i$), a linear regression pattern line was the best fit, which confirmed the piecewise linear model:

$$y_i = 42.162x_i + 0.009$$

| File I/O Loop | Total Service Requests | Total time for requests in milliseconds | Service Requests per second | Service Stress Point (Requests per second) | Service Potential (V) in Requests per second | Average Orchestration Service Response Time (ART) in seconds | Orchestration Service Performance (P=1/ART) in seconds$^{-1}$ | Calculated overall Service Impedance Gradient ($I_G$ = P/V) | Average actual File I/O latency ($I_{FIO}$) in seconds | Impedance Factor ($I_G/I_{FIO}$) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 150 | 19187 | 7.817793298 | 50 | 42.1822067 | 0.141966667 | 7.043907004 | 0.166987637 | 0.005017333 | 33.28215141 |
| **3** | **150** | **22016** | **6.813226744** | **50** | **43.18677326** | **0.65048** | **1.537326282** | **0.035597155** | **0.070424** | **0.505469094** |
| 5 | 150 | 21314 | 7.03762785 | 50 | 42.96237215 | 1.5107 | 0.661944794 | 0.015407548 | 0.179541333 | 0.085816159 |
| 10 | 150 | 21171 | 7.085163667 | 50 | 42.91483633 | 3.867106667 | 0.258591264 | 0.006025684 | 0.627218667 | 0.009606992 |
| 15 | 150 | 21104 | 7.107657316 | 50 | 42.89234268 | 5.60749333 | 0.178332802 | 0.004157684 | 0.932293333 | 0.004459631 |
| 20 | 150 | 20640 | 7.26744186 | 50 | 42.73255814 | 7.2279 | 0.138352772 | 0.003237643 | 1.22545333 | 0.002641996 |
| 25 | 150 | 20577 | 7.289692375 | 50 | 42.71030763 | 9.692086667 | 0.103176956 | 0.002415739 | 1.657397333 | 0.00145755 |

Table 3: Empirical Data for varying File I/O

Table 3 and Figures 10 and 11 present the data obtained, the empirical graphs of '$I_{FIO}$' versus '$I_G$' and '$I_G$' versus 'P'. Pattern validation is highlighted.

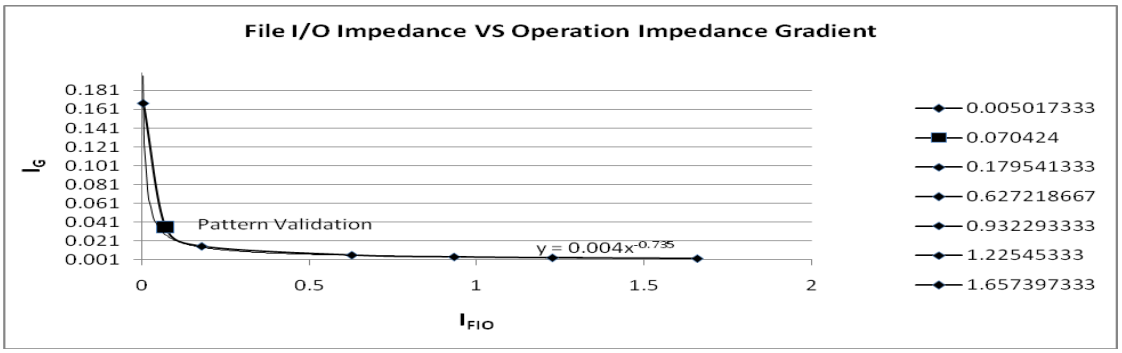


Figure 10: Empirical Data Graph for $I_{FIO}$ vs $I_G$ for varying File I/O

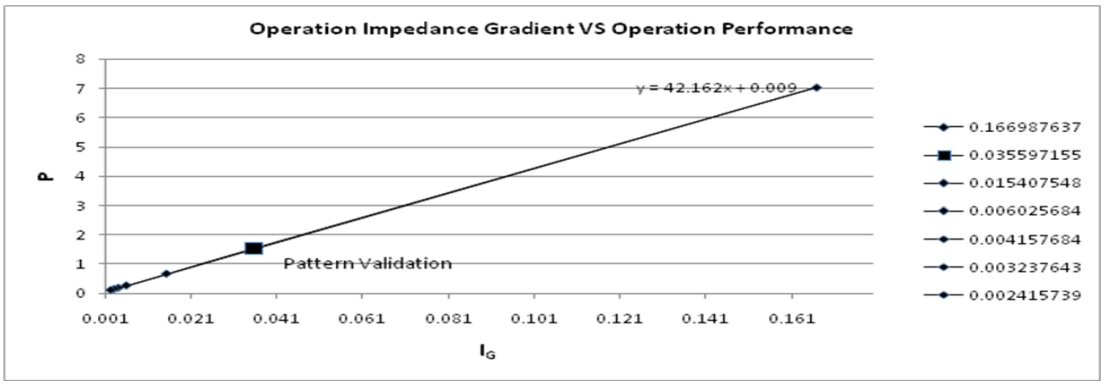


Figure 11: Empirical Data Graph for $I_G$ vs P for varying File I/O

Other types of Delay Points like XML parsing and processing for Request Authentication and Authorization were also tested in similar ways and empirical results were collected. They all showed distinct trends of variation of the overall performance 'P' and Service Operation Impedance Gradient '$I_G$' for variations to the underlying Delay Point Impedances.

### *7.4 Simultaneous changes to multiple 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 multiple Delay Point types modified simultaneously. To address this scenario, multiple Delay Points (Database Interactions, Data Processing, File I/O and XML Processing) were varied simultaneously and experiments were run. For each process load configuration of the set of Delay Points, the aggregate of 5 runs were calculated to smooth out the data and remove any occasional noise. Experiments were run for 5 different process load configuration settings for the set of Delay

Points. The overall Service Operation Performance 'P' and Operation Impedance Gradient '$I_G$' were calculated for every configuration run.

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

| Orchestration Service Performance (P=1/ART) in seconds$^{-1}$ | Calculated overall Service Impedance Gradient ($I_G$ = P/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 4: Calculated $I_G$ and corresponding actual Delay Point latencies

The above dataset represents different combinations of the Delay Point variations and their corresponding calculated '$I_G$'. The atomic Delay Points (DB Access, Data Processing, File I/O etc) are treated as *independent* variables and the data is presented in the form:

$$\boldsymbol{AX = B}$$

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
- *B* is the single column [5x1] matrix of the calculated '$I_G$' for each row in *A*
- *X* is the single column [5x1] Delay Point Impedance Conversion Matrix

The *best fit* value of *X* is calculated through matrix transpose and inverse in the following way:

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

The Conversion Matrix *X* is calculated to facilitate projection of '$I_G$' for any arbitrary combination of Delay Point Impedances.

We present below the step by step process by which the Conversion Matrix X is derived and subsequently how its precision is validated:

**Matrix Conversion Model**

| A | | | | | X | | B |
|---|---|---|---|---|---|---|---|
| $I_{DB}$ | $I_{DP}$ | $I_{FIO}$ | $I_{AN}$ | $I_{AR}$ | Conversion Matrix | | $I_G$ |
| 1.14508 | 0.00104 | 0.063876667 | 0.00722 | 0.00339 | $M_1$ | | **0.002212067** |
| 1.549756667 | 0.001193333 | 0.208513333 | 0.003893333 | 0.002436667 | $M_2$ | = | **0.00176775** |
| 0.879663333 | 0.002133333 | 1.082626667 | 0.006343333 | 0.005226667 | $M_3$ | | **0.001620992** |
| 1.187556667 | 0.002346667 | 1.032653333 | 0.00823 | 0.005473333 | $M_4$ | | **0.001352816** |
| 1.219683333 | 0.00275 | 1.375003333 | 0.003016667 | 0.003183333 | $M_5$ | | **0.001151423** |

**$A^T$ =**

```
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 11945971.37958
```

**$A^TB$ =**

```
0.009709420
0.000014209
0.005245028
0.000047743
0.000031348
```

**Impedance Conversion Matrix $X = (A^TA)^{-1}A^TB =$**

```
-0.004356602
10.074393338
-0.017590749
-1.320577244
 2.177624419
```

To cross check the integrity of X (Impedance Conversion Matrix), we back-calculated matrix B' (single column Impedance Gradient Matrix) by multiplying A with X and compared it with the original matrix B. The values matched up till 6 decimal places. 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
```

**Validation of the Matrix Conversion Model**

The Matrix Conversion model had to be validated to assess the precision of the forecast for overall Service Operation Impedance Gradient ($I_G$) for any given set of Delay Point Impedances.

To achieve this, tests were run on the service framework with the Delay Point process load 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 Impedance Conversion Matrix. Empirical data for ***actual*** overall 'P', computed value of overall '$I_G$' based on the ***actual*** overall 'P' and 'V' of the ***PIV*** model (say $\mathbf{I_{GPIV}}$) and the ***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 $I_{AN}$) were recorded.

The single row Delay Point Impedance matrix comprising of $I_{DB}$, $I_{DP}$, $I_{FIO,}$ $I_{AR}$ and $I_{AN}$ was then multiplied by the previously derived Impedance Conversion Matrix 'X' to calculate the overall '$I_G$' again (say $\mathbf{I_{GMCM}}$), this time based on the Matrix Conversion Model.

The delta between $\mathbf{I_{GMCM}}$ and $\mathbf{I_{GPIV}}$ was 4.459133225%. As a first iteration, this error percentage was considered acceptable. Through calibration of the Matrix Conversion Model, we shall be able to minimise the delta between $I_{GMCM}$ and $I_{GPIV}$ and achieve much higher precision.

Data from the validation test runs are shown below:

| PIV model $I_G$ ($I_{GPIV}$) | $I_{DB}$ | $I_{DP}$ | $I_{FIO}$ | $I_{AN}$ | $I_{AR}$ |
|---|---|---|---|---|---|
| 0.002574027 | 1.034386667 | 0.00078 | 0.04260667 | 0.003173333 | 0.001966667 |

Previously derived Impedance Conversion Matrix X:

| Impedance Conversion Matrix |
|---|
| -0.004356602 |
| 10.07439334 |
| -0.017590749 |
| -1.320577244 |
| 2.177624419 |

Projected $I_{GMCM}$ and error percentage ((|$I_{GPIV}$ – $I_{GMCM}$| / $I_{GPIV}$) * 100):

| Projected $I_G$ ($I_{GMCM}$) | $I_G$ Delta % |
|---|---|
| 0.002694163 | 4.459133225 |

Figure 12 below shows the regression line for Service Impedance Gradient ($I_G$) versus Service Performance (P) as obtained from the data in Table 1 above:

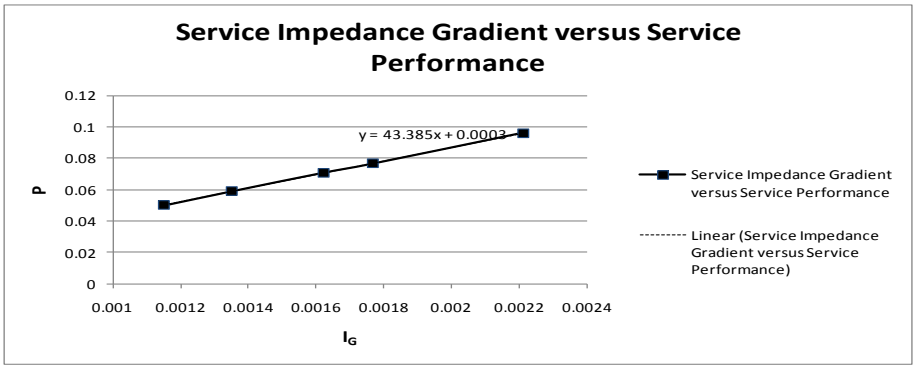


Figure 12: Empirical Data Graph of $I_G$ vs P

A projected value of the Service Performance (P) under the given load is calculated by substituting '***x***' in the regression line with $I_{GMCM}$.

## 8 Conclusions and Future Work

In this paper we have addressed the critical need of a method to facilitate analysis and prediction of the impact on performance due to system modifications. At a high level, the research contribution can be categorized into four broad areas:

a. ***Relational Model for Service Performance, Impedance and Load (PIV Model)*** - Empirically establish an innovative relational model associating the three key non-functional aspects of any system or service operation. These are Service Operation Performance '$\boldsymbol{P}$', Service Operation Load Potential '$\boldsymbol{V}$' (related to service load) and Service Operation Impedance Gradient '$\boldsymbol{I_G}$'(related to all the activities performed by the service's application components internally).

b. ***Service Performance Predictive Model for single Delay Point modification*** - Aided by the deduced relational model, empirically establish that service performance varies in particular patterns depending on the type of activities being performed internally by the service. The possible impact of modifying every *individual* application component Delay Point on the overall system performance under a given load condition is assessed.

c. ***Service Performance Predictive Model for multiple Delay Points modification*** - Using a combination of the deduced relational model and the individual component's impact patterns, empirically develop a matrix based predictive model to forecast the service performance for simultaneous changes to *multiple* underlying application component Delay Points.

d. ***Overall contribution to the Quality of Software as a Service –*** The ability to predict (with acceptable error factor) the impact on Service Performance due to changes to the underlying application component Delay Points will facilitate improving the overall quality of the service. The contribution may be categorized in the following areas:

   i. *Quality Improvement* – The predictive models will facilitate upfront detection of the impact of the changes to the various application component Delay Points. Through skeletal system testing, the developers themselves will be able to analyse the effect of their changes to the Delay Points on the outward performance of the service. If there is any quality degradation from performance perspective, this will enable improving it through measures like code refactoring and optimization.

   ii. *Software Quality Assurance* – Upfront analysis and forecast of the service's performance due to the modifications to the application component Delay Points 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, they'll be able to address any issue upfront without the need to go through overheads like source code control, code reviews, change management, configuration management, testing and release management. To a great extent the need to undo and redo component changes only after performance/load testing at the end will no longer be there. This will ensure timely assurance of the service'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 service will conform to any prescribed SLA or 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 a service. 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. 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.

**Summary of the Empirical deductions**

In the first step of the research, we deduced the relational runtime model between Service Performance (related to response time), Service Potential (related to request load) and Service Impedance gradient related to the latencies due to application components' Delay Points' activities. Based on this model, the subsequent phases of the research are undertaken.

The next phase of the research focussed on establishing the hypothesis that every *individual* application component Delay Point of the service impacts the overall system performance in particular patterns under a given load condition. The service framework is used to create some illustrative application components with Database Interactions, Data Processing, File I/O, XML Processing and other types of Delay Points. Iterative tests were run by varying each type of Delay Point at a time and keeping the rest of the system configuration constant including the request load. Experiments demonstrated different types of Delay Points impacted the overall service performance in distinct patterns. These patterns can be used to extrapolate or interpolate potential impacts on service performance for future modifications to application component activities.

During enhancements, the production systems in the industry typically involve multiple types of Delay Points being modified simultaneously. The research has addressed the impact on overall Service Operation Impedance Gradient (which ties to overall Operation Performance) due to simultaneous changes to underlying application components Delay Points. A matrix based predictive model has been developed to forecast the service performance for simultaneous changes to *multiple* types of underlying application component activities. Using the Matrix Conversion Model, the research has been able to predict the overall Operation Impedance Gradient '$I_G$' for changes to the underlying application components' Delay Points simultaneously. When compared to the '$I_G$' computed from the actual measured Service Operation Performance and Potential, the error factor is found to be 4.459133225%. Given the fact this is only the first iteration of the predictive model without any calibration, the error factor is considered to be within acceptable limits. A projected value of Service Performance (P) under the given load is calculated by using the predicted '$I_G$'.

The new formula-aided pattern based technique and the matrix conversion model will allow upfront impact analysis of application component Delay Point changes by the developers themselves without

the need of additional testing/system admin resources or much *external* tool overhead. The graphical pattern based technique will enable forecast of possible impact on the overall system performance under a given load condition due to modifications to *individual* application components. The matrix based model will enable forecast of possible impact on the overall system performance under a given load condition due to modifications to *multiple* application components Delay Points simultaneously.

This should help address the typical resourcing issues faced during service component enhancements and potentially provide time, resource and significant cost savings to the IT industry.

**Matrix Conversion Model Calibration**

Further work needs to be done towards calibrating the Matrix Conversion Model. During establishing the model and deriving the single column Impedance Conversion Matrix, the Delay Point process load configuration of the components was varied in uniform steps. This yielded forecasts with about 4.4% error approximately. In real life, the process load variations across the components may not be uniform. Hence we need to calibrate the model through other process load configurations. Randomizing the Delay Point load distribution will provide greater coverage of the Delay Point Impedances. This should facilitate increase in forecast precision, reduce percentage error and make the model more robust.

**References**

1. Sid Kargupta and Sue Black, 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, June 22nd – 26th, 2009

2. Sid Kargupta and Sue Black, Visualizing the Underlying Trends of Component Latencies affecting Service Operation Performance, ACTEA 2009, IEEE International Conference on Advances in Computational Tools for Engineering Applications, Lebanon, July 15th – 17th, 2009

3. Optimized Performance Testing (OPT) at Sonata Software, http://www.sonata-software.com /export/sites/Sonata/sonata_en/innovation/resources/brochures/pdfs/Optimized_Performance_Testing_ at_Sonata_Software.pdf [last accessed 22/02/2012]

4. CAA-KPMG Report on BAA Operating Expenditure, http://www.caa.co.uk/docs/5/ergdocs/ ccstanstedh.pdf [last accessed 26/07/2010]

5. B2B Cost Cutting and Cost Saving Portal, CostKiller.net, http://costkiller.net/costs-savings/costs-cutting-The-Migration-to-VoIP.htm [last accessed 22/02/2012]

6. White Paper on Portal Migration Strategy, http://www.chapter26.com/services/Portal Migration Strategy - White Paper v1 1.pdf, [last accessed 22/02/2012]

7. Change Impact Analysis, http://www.changelogic.com/GettingStarted/ImpactAnalysis?v=13ot, [last accessed 30/03/2012]

8. Mitigating the Business Impact of Software Changes with Coverity5, http:// www.coverity.com/library/pdf/Coverity5Brochure.pdf [last accessed 22/02/2012]

9. Change Management: Evaluating CM Tools to Implement a Successful ERP Application CM Strategy, http://www.oracleappsblog.com/index.php [last accessed 22/02/2012]

10. Natalie L. Petouhoff, Tamra Chandler and Beth Montag-Schultz, Graziadio School of Business and Management, Pepperdine University, Graziadio Business Report - The Business Impact of Change Management, http://gbr.pepperdine.edu/063/change.html [last accessed 22/02/2012]

11. Daniel A. Menasce, Virgilio A. F. Almeida: Capacity Planning for Web Services. Metrics, Models and Methods. Prentice Hall PTR, Upper Saddle River, NJ 07458, 2002

12. Arun Iyengar, Jim Challenger, Daniel Dias and Paul Dantzig: High Performance Web Site Design Techniques. Web Design, IEEE Internet Computing, March-April, 2000

13. D. Ardagna and B. Pernici, Adaptive Service Composition in Flexible Processes, IEEE Transactions on Software Engineering, Vol. 33, No.6, June, 2007

14. Yan Liu, Alan Fekete and Ian Gorton, Design-Level Performance Prediction of Component-Based Applications, IEEE Transactions on Software Engineering, Vol.31, No.11, 2005

15. Xiuping Wu, David McMullan and Murray Woodside, Component Based Performance Prediction, Proceedings of the 6th ICSE Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction, 2003

16. Shiping Chen, Ian Gorton, Anna Liu and Yan Liu, Performance Prediction of COTS Component-based Enterprise Applications, Journal of Systems and Software, Vol 74, Issue 1, 2005, Pages 35-43.

17. Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi and Marta Simeoni, Model-Based Performance Prediction in Software Development: A Survey, IEEE Transactions on Software Engineering, Vol.30, No.5, 2004

18. Connie U. Smith and Lloyd G. Williams, Performance Engineering Evaluation of Object-Oriented Systems with SPE.ED, Computer Performance Evaluation: Modelling Techniques and Tools, No.1245, Springer-Verlag, Berlin, 1997

19. Christopher Stewart and Kai Shen, Performance Modeling and System Management for Multi-component Online Services, Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, May2-4, Boston, MA, USA, 2005

20. K.S. Jasmine and R. Vasantha, Design Based Performance Prediction of Component Based Software Products, Proceedings of the World Academy of Science, Engineering and Technology, Vol 24, ISSN 1307-6884, October, 2007

21. Shiping Chen, Bo Yan, Zic J., Ren Liu and Nq A., Evaluation and Modeling of Web Services Performance, IEEE International Conference on Web Services (ICWS'06) 0-7695-2669-1/06, Sept. 2006

22. Menasce D.A., Performance and Availability of Internet Data Centers, Internet Computing, IEEE. George Mason University, Fairfax, VA, USA, May-June 2004, Pages 94 - 96.

23. Carolyn McGregor and Josef Schiefer, A Framework for Analyzing and Measuring Business Performance with Web Services, Proceedings of the IEEE International Conference on E-Commerce (CEC'03) 0-7695-1969-5/03, 2003

24. Paul Fortier, Howard Michel, Computer Systems Performance Evaluation and Prediction, ISBN 1-55558-260-5, Digital Press, 2003

25. I-Hsin Chung and Jeffrey K. Hollingsworth, Automated Cluster-Based Web Service Performance Tuning, Proceedings of the 13th Conference on High Performance Distributed Computing, 2004, Pages 36 – 43.

26. Marco Vieira and Nuno Laranjeiro, Comparing Web Services Performance and Recovery in the Presence of Faults, Proceedings of the IEEE International Conference on Web Services (ICWS 2007), 0-7695-2924-0/07.

27. Ana C. C. Machado and Carlos A. G. Ferraz, JWSPerf: A Performance Benchmarking Utility with Support to Multiple Web Services Implementations, Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), 0-7695-2522-9/06.

28. Manjari Asawa, Measuring and Analysing Service Levels: A Scalable Passive Approach, Proceedings of the 1998 Sixth International Workshop on Quality of Service, 1998. (IWQoS 98), Pages 3 – 12.

29. Jamal Bentahar, Zakaria Maamar, Djamal Benslimane and Philippe Thiran, An Argumentation Framework for Communities of Web Services, IEEE Computer Society, 1541-1672/07, 2007, Pages 75 - 83.

30. Shiping Chen, John Zic, Kezhe Tang, David Levy, Performance Evaluation and Modeling of Web Services Security, IEEE International Conference on Web Services (ICWS 2007), 0-7695-2924-0/07.

31. Ariel Pashtan, Andrea Heusser and Peter Scheuermann, Personal Service Areas for Mobile Web Applications, IEEE Internet Computing, November-December 2004, Pages 34 - 39.

32. Matthew Swinarski, Rajiv Kishore, H. Raghav Rao, Impact of IT Service Provider Process Capabilities on Service Provider Performance: An Empirical Study, Proceedings of the 39th Hawaii International Conference on System Sciences – 2006

33. Ikuo Matsumura, Toru Ishida, Yohei Murakami and Yoshiyuki Fujishiro, Situated Web Service: Context-Aware Approach to High-Speed Web Service Communication, Proceedings of the IEEE International Conference on Web Services (ICWS'06), 0-7695-2669-1/06.

34. Sriram Anand, Srinivas Padmanabhuni, Jai Ganesh, Perspectives on Service Oriented Architecture, Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), 0-7695-2408-7/05

35. Philipp Liegl, The strategic impact of service oriented architectures, Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), 0-7695-2772-8/07.

36. Matjaz B. Juric, Marjan Hericko, Tatjana Welzer, Ivan Rozman, Ana Sasa and Marjan Krisper, Web Services and Java Middleware Functional and Performance Analysis for SOA, Proceedings of the 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2007), Pages 217 – 222.

37. Chuanchang Liu, Zhongwei Li, Junliang Chen, Xiao Lin, Dongtang Ma, A Service Level Agreement-based Web Services Performance, Proceedings of the 16th International Conference on Computer Communications and Networks, 2007, ICCCN 2007, Pages 1279 - 1284

38. Giovanni Pacifici, Mike Spreitzer, Asser N. Tantawi and Alaa Youssef, Performance Management for Cluster-Based Web Services, IEEE Journal On Selected Areas In Communications, Vol. 23, No. 12, Dec 2005, Pages 2333 – 2343.

39. Jean-Pierre Garbani with Robert Zimmerman and Stephan Wenninger, Best Practices For Service-Level Management, 2004 Forrester Research, Inc, TechRepublic White Paper, http://whitepapers.techrepublic.com [last accessed 22/02/2012]

40. End-End Software (Asia Pacific) Pte Ltd, info@end-endsoftware.com, 10, Jalan Besar, 10-03 Sim Lim Tower, Singapore 208787, Service Level Agreements, TechRepublic White Paper, http://whitepapers.techrepublic.com, January 2002 [last accessed 22/02/2012]

41. Zhengdong Gao and Gengfeng Wu, Combining QoS-based service selection with performance prediction, e-Business Engineering, 2005, ICEBE 2005, IEEE International Conference, School of Comput. Eng. & Sci., Shanghai Univ., China, 18-21 Oct., 2005, Pages 611- 614.

42. Mou Yu-jie, Cao Jian, Zhang Shen-sheng, Zhang Jian-hong, Interactive Web Service Choice-Making on Extended QoS Model, Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05), 0-7695-2432-X/05.

43. Marco Conti, Mohan Kumar, Sajal K. Das and Behrooz A. Shirazi, Quality of Service Issues in Internet Web Services, Proceedings of the IEEE Transactions on Computers, Vol. 51, No. 6, June 2002, Pages 593 - 594

44. Glen Dobson, Russell Lock, Ian Sommerville, QoSOnt: a QoS Ontology for Service-Centric Systems, Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05)

45. Gwyduk Yeom, Dugki Min, Design and Implementation of Web Services QoS Broker, Proceedings of the IEEE International Conference on Next Generation Web Services Practices (NWeSP'05), 2005, 0-7695-2452-4/05.

46. Ahsan Habib, Sonia Fahmy, Srinivas R. Avasarala, Venkatesh Prabhakar, Bharat Bhargava, On Detecting Service Violations and Bandwidth Theft in QoS Network Domains, TechRepublic White Paper, http://whitepapers.techrepublic.com, 5th November 2002 [last accessed 22/02/2012]

47. Least Squares Method, http://en.wikiversity.org/wiki/Least-Squares_Method, Last modified on 29th June 2010 [last accessed 22/02/2012]