A Load Balancing Framework for SMS Invocation of Services in SOA Environment

A dissertation submitted by:

Mandla Trevor Nene

(Reg No: 20065585)

To:

Department of Computer Science,

Faculty of Science and Agriculture,

University of Zululand,

Kwadlangezwa 3886, RSA

Supervisor: Prof MO Adigun

Co-Supervisor: E Jembere

2013

Declaration

I, Mandla Trevor Nene, declare that this dissertation represents my own work and that this work has not been previously submitted to any university or other institution of tertiary education. All sources of information used in this work have been acknowledged.

Mandla T Nene

Signature_____

Dedication

To

Busisiwe Rose Zulu

Acknowledgements

I would like to thank my supervisor, Prof. M.O. Adigun, for the opportunity and his guidance, advice and support.

I would also like to thank Edgar Jembere and cluster members for their support and advice. I would like to thank my family for their understanding and support.

Finally, I thank Telkom for funding me throughout my research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations & Acronyms	Description
CaLLF	Client-aware Least Loaded Framework
HTTP	Hyper Text Transfer Protocol
SOA	Service Oreinted Architecture
SOC	Service Oreinted Computing
SOAP	Simple Object Access Protocol
SMS	Short Message Service
XML	Extensible Markup Language

Abstract

The drastic increase in the usage of SMS technology has led to solutions that seek to enable users of mobile devices to access services through SMSs. This has resulted in the proposal of solutions towards SMS-based service invocation in service-oriented environments. However, the dynamic nature of service-oriented environments, coupled with sudden load peaks generated by service requests, poses a performance problem to the service provisioning environment such as service provider's servers.

In this research work we adopted load balancing techniques to address this problem. In addition, we observe that service-oriented environments deal with different service clients requiring different service quality. So it was deemed necessary to design a client-aware load balancing approach for SMS invocation of services. This is realisable through the Client-aware Least Loaded Framework (CaLLF). CaLLF allows service clients to be served and distributed across service provider's servers (infrastructure) according to their class of importance. CaLLF achieves this with the combination of prioritisation, over dimensioning of resources and a least loaded approach. The evaluation of CaLLF for scalability showed it can cope with increasing traffic while providing better performance compared to Round robin (RR), the currently used scheme. However, even though CaLLF achieves better results than RR, it does so at a higher cost in terms of the computational resources it uses. We also evaluated CaLLF for utility and observed that CaLLF can provide better utility for premium clients compared with non-client aware least loaded load balancing approach but this was achieved with a trade off on the utility of regular clients.

Chapter 1

Introduction

1.1 Introductory Background

The University of Zululand, Computer Science Department is currently working on the Grid-Based Utility Infrastructure for Small, Medium and Micro Enterprises (SMMEs) Enabling Technologies (GUISET) (Adigun *et al*, 2006). GUISET takes advantage of the affordability and popularity of mobile devices as well as advancements in Service Oriented Architecture (SOA) to address the hardware and software acquisition problems for resource constrained enterprises such as SMMEs. In addressing these problems, GUISET is implemented in SOA technology and hence the challenges towards realising SOA supporting mobile devices technology are also GUISET issues.

Service-oriented computing (SOC) is a paradigm that takes advantage of web services as building blocks, in order to support the development of affordable and easy composition of applications in distributed systems (Vanrompay *et al*, 2009). An SOC model consists of three aspects which are:

- ***service providers*** (organisations that provide the web service implementations, supply their service descriptions, and provide related technical and business support);
- ***service clients*** (end-user organisations that use some web services);
- ***service registry*** (a searchable directory where web service descriptions can be published and searched for).

In SOC, service clients find service descriptions in the registry and obtain binding information for services. This information is sufficient for the service client to contact, or bind to, the service provider and thus service clients make use of the services provided by a service provider. If taking into account that service clients are exposed to services through various interfaces such as desktop PCs via web browsers and mobile devices (such as cell phones) via Short Message Service (SMS) which brings an advantage of mobility to service clients.

The advancements in mobile computing and the development of SOC have enabled mobile devices to participate in service-oriented environments both as service providers and as service clients (Doulkeridis *et al*, 2006). This has made mobile devices which implement SMS technology, not only to be used for sending and receiving text messages among mobile devices users, but also for something more useful than unidirectional communications. For example, service clients can reach services from a service provider by requesting and retrieving services through SMSs (Ramana *et al*, 2005; Risi and Teofilo, 2009). This is known as SMS based service invocation. In each case, a service client sends an SMS containing service parameters. This request is forwarded to the back-end of a service provider's server that hosts that particular service which in turn responds to the request accordingly. This is due to the fact that more users are getting interested in finding effective ways to access public web services through their mobile devices as they usually do with their desktop PCs (Lin *et al*, 2009). Moreover, the success of web services, open standard based technology (Rendon *et al*, 2005) and frameworks like Open Services Gateway Initiative (OSGI) (Lin *et al*, 2009) have complemented the existence of SMS based service invocation.

Despite the major success of the adoption of SMS-based service invocation taking place in service oriented environments, there are still some challenges. To begin with the intrinsic advantage of SMS based service invocation is the low cost of SMS usage (Wan, 2008). However, a challenge arises when a large number of mobile users (service clients) take advantage of low cost SMS based service invocation. It can result in service providers back end receiving burst loads of request messages from mobile users, which may lead to overloading on the service providers' server resources. In turn, the performance in service providers' server resources is degraded, i.e. the response time will increase leading to service clients waiting too long, hence leading to dissatisfaction among service clients. This, coupled with the dynamic nature of the service-oriented environment and autonomy of the entities involved, results in unexpected behaviour from the entities involved. During SMS invocation of services, an abnormal behaviour from mobile users (service clients) can occur which may create unexpected traffic resulting in service provider servers being overloaded. There must be special precautions taken for unexpected request patterns that drive servers into overloading leading to poor performance. There are a number of approaches (Ramana *et al*, 2005; Risi and Teofilo, 2009 ; Lin *et al*, 2009; Rendon *et al*, 2005)proposed for SMS based invocation of services which focus on improving technology and architecture. Examples of these approaches include the web service access from low end mobile devices using SMS and Mobile deck which are based on enhancing SMS user interface so that service clients receive and retrieve content which is meaningful. Lastly, the other efforts include Pandora (Rendon et al, 2005) and WSA client architectures for accessing services via SMS (Rendon *et al*, 2005).

Lin *et al* (2009) acknowledge that even though SMS based service invocation is made possible in service oriented environments; there are performance problems with respect to the number of SMS requests that can be handled by the system without affecting QoS.

Load balancing is widely accepted as a solution to reduce the performance problem mentioned above (Bourke, 2001; Chandra, 2002). Load balancing is a process of distributing workload evenly across multiple computers or a computer cluster, network links, central processing units, or other resources, to achieve optimal resource utilisation, maximise throughput, minimise response time, and avoid overload. However, this work considers load balancing in the context of sharing load across multiple computers (servers). Scholars like Ramana *et al* (2005) proposed, among other load balancing solutions, the queuing approach to load balancing. The queing approach implements a store and forward mechanism. The forwarding mechanism acts as a load dispatcher that implements simple load balancing schemes (Ramana *et al*, 2005) such as the round-robin (RR) scheme that distributes loads around service provider servers iteratively. The RR may perform well in cases where the servers are homogeneous and assumed to be always available. In cases where the servers may be heterogeneous, the RR might end up overloading the servers with low processing capacity while servers with higher processing capacity have fewer loads to execute and this results in poor performance. In this work, we argue that using the simple queuing approach with RR as the load distribution mechanism, leads to poor load distribution owing to the fact that RR does not consider any runtime system state information in making load distribution decisions. In view of the fore-going, this work aims to investigate the applicability of existing load balancing approaches for dynamic environments, in developing load balancing solutions for SMS invocation of services in service-oriented environments that deal with unexpected traffic in a scalable manner while providing adequate performance.

1.2 Statement of the Problem

Service-oriented environments are by nature dynamic and composed of autonomous entities, which makes them unpredictable and difficult to manage (Papazoglou, 2007). Further to the dynamic nature of such environments, the SMS-invocation of services in SOC makes the environment more unpredictable and difficult to manage due to unexpected traffic and sudden load peaks which are common in such environments. This poses a performance problem to the infrastructure supporting SMS invocation of services. In light of addressing this performance problem in SMS-based invocation of services in SOA environments, this work investigated the following issues:

1. How can we come up with a load balancing approach that monitors and handles unexpected load in the SMS invocation of service in SOA environments?
2. How should load balancing solutions be designed to guarantee different client priority if we have premium and regular clients?
3. What performance metrics are needed for the purpose of load monitoring?

1.3 Motivation

SMS enables the use of mobile devices for accessing web services. This is advantageous to communities of people who are still trying to bridge the digital gap. The problem arises when many people take advantage of the low cost of SMS-based service invocation resulting in the service providers back end receiving a burst load of request messages from service consumers. This may result in the overloading of the infrastructure supporting invocation of services. In turn there may be a drop in service providers' servers performance as a result of an increase in response time, This may lead to client disatisfaction.

Given this background, we envisage an approach that fosters a load balancing solution, which will contribute to SMS invocation of service environments that enable service providers to deal with stress loads or sudden load peaks without affecting the quality of service. By providing a load balancing approach that tries to address performance issues occurring in such environments, the service providers and service clients will benefit in such a way that service provider infrastructure will provide better performance hence leading to service clients' satisfaction.

1.4 Research Goal

The goal of this study is to come up with a load balancing framework that would be able to function in SMS based service invocation environments and also serve clients based on their categories.

1.5 Objective

In order to achieve our goal, we formulated some equivalent lower-level objectives which were:

1. To survey the existing load balancing solutions and group them according to common approaches.
2. To investigate the common performance metrics considered for monitoring load, with the aim of adapting them for SMS-based service invocation environments.
3. To develop a load balancing framework that adapts to the nature of SMS based service invocation. Moreover, we need to incorporate client classification mechanisms in the framework that will influence load balancing and be able to serve clients according to client categories.

4. To evaluate the load balancing framework developed in objective (3) for efficacy and utility.

1.6 Research Methodology

The achievement of the above mentioned objectives was possible through the employment of the following methodologies:

1.6.1 Literature Survey

The aim of the literature survey was to establish a good theoretical background for this research work and also to explore trends in load balancing approaches in distributed environments such as SOA, Publish/subscribe and grid environment with special focus on how they handle load in dynamic environments. Moreover, we also explored approaches for requests classification and prioritisation to cater for the fact that clients require different service quality in SMS based service invocation environments. Thereafter, a survey of the common performance metrics for monitoring load in distributed environments was carried out.

1.6.2 Modelling

Modelling was our primary method which depended on secondary methods such as the literature survey mentioned in Section 1.6.1 and also experimentation for the purpose of crafting and validating the artefact produced. Modelling was done as follows: based on the insights gained from literature on approaches and trends in the area of load balancing and requests classification or prioritisation in various applications; a load balancing framework for monitoring and handling load, combined with requests classification and prioritisation approach, was designed. This includes taking common performance metrics to monitor load used by load balancing algorithm for decision making inside the framework and

incorporating client category mechanisms that will differentiate between premium and regular clients. After the model was developed there was a need to evaluate the model using experimentation which is discussed in Section 1.6.3.

1.6.3 Experimentation

As proof of concept, we implemented and evaluated our load balancing approach combined with the prioritisation approach which was formed with a framework for efficacy and utility. In evaluating the effectiveness of the proposed load balancing approach, we started with concept implementation and then it was fitted to service oriented environments so that we could observe how it handles load (load testing) compared to the popularly used round-robin approach. We created two scenarios to test the efficacy of how our load balancing approach performs and relevant metrics were recorded. We also evaluated the utility of the proposed load balancing approach combined with prioritisation against the the same load balancing approach without prioritisation. This was done in order to investigate whether prioritisation improves client satisfaction or not. Premium clients are clients that require higher service quality over and above regular clients. Based on the satisfaction level given by each client, whether premium or regular, the utility of each client was observed and the percentage of satisfied clients was recorded. The generated data was analysed for each experiment and conclusions were drawn.

1.7 Dissertation Synopsis

In this Chapter we highlighted SMS based service invocation issues associated with SOA. In SOA, services can be published, discovered and utilised. Web services are one of the most promising technologies in the implementation of SOC (Papazoglou, 2007). Web services provide the basis for the development and execution of business processes that are distributed by the network and are available via standard interfaces and protocols. After services are published by providers, they need to be discovered and utilised by service clients. Service clients can utilise services through various devices such as Desktop PCs and mobile phones etc. In our case, we are describing mobile phones through the SMS platform known as SMS based service invocation.

In this Chapter we indicated that this project aims to come up with a load balancing framework for SMS based service invocation environments in order to deal with overloading in service providers, by providing a mechanism that handles and monitors load in a dynamic environment such as ours. In addition, the framework should be able to serve clients according to their category. For the realization of this goal, we explored various types of load balancing techniques, client classification techniques and then mapped them to our situation because we need to find one of the techniques that will fit our environment.

Chapter 2, therefore, presents a background on the important concepts of SMS based service invocation, load balancing and prioritisation based client classification. Chapter 3 examines the approaches proposed by other researchers in addressing the issue of performance in service provider's infrastructure in various disciplines. Related literature is discussed according to the following aspects: *load balancing techniques and algorithms and clients prioritisation techniques*. Chapter 3 also outlines lessons learnt from the evaluation of these approaches using an evaluation framework.

Chapter 4 presents the model development. It firstly presents the load balancing approach preferred in our model and then shows how the model is made to take note of clients' classes of importance. Finally, the load balancing framework is developed to improve its load handling capability. The third research method adopted in this study is the experimentation. To evaluate the effectiveness and utility of the solution, experiments had to be conducted. The implementation and evaluation result of our load balancing framework are presented in Chapter 5.

Chapter 6 concludes our work and also presents gaps for future work.

Chapter 2

Background Concepts

2.1 Introduction

This Chapter presents a background of key concepts in load balancing and SMS invocation of services in SOA environments.With the drastic increase in SMS usage, SMS technology has developed to enable users of mobile devices to access services through SMSs. The popularity of the use of SMSs for service invocation and the performance challenges it presents to the service provisioning infrastructure have generated a lot of interest in the research community. This Chapter presents the underlying concepts of the solutions that have been proposed for the provision of services through SMSs.

The discussion starts off with SOA key concepts and service provisioning in Section 2.2 and is followed by the evolution of SMS-based service invocation in Section 2.3. Section 2.4 presents the SMS based service invocation architecture. Load balancing as a technique is discussed in Section 2.5 which deals with the performance challenges faced by distributed systems infrastructure. The notion of classification and prioritisation of client's requests for dealing with clients who require different service quality is presented Section 2.6. Section 2.7 presents typical SMS broker functionalities and shows how load balancing applies in this context. Finally, the Chapter summary is contained in Section 2.8.

2.2 SOA and Service provisioning

The perpetual demand for IT infrastructure to support rapidly evolving business needs, has led to a rising interest in Service Oriented Architecture (SOA) (Weerawarana *et al*, 2005). SOA is a specific architectural style that is concerned with loose coupling and dynamic binding between services. There are three important components that define SOA (see Figure 2.1). These are:

(i) **Service provider** is an entity that builds and publishes the web services' interface that they offer and also reacts to requests that utilise its web services.

(ii) **Service client** is an entity that finds and utilises the web services published by the service providers for their business needs.

(iii) **Service registry** is a repository that stores the interfaces of the published web services as illustrated in Figure 2.1

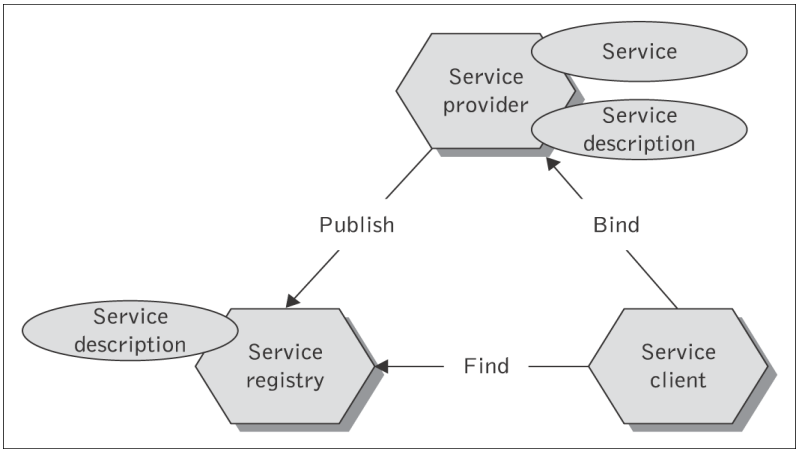


Figure 2.1 SOA architecture adopted Papazoglou (2007)

One way of implementing the SOA is to take advantage of web services where "web services are loosely coupled, reusable software components that semantically encapsulate discrete

functionality and are distributed and programmatically accessible over standard Internet protocols" (Sleeper, 2001). This may mean web services can be implemented using any technology or any level of complexity and are therefore autonomous. A web services application uses these open protocols, namely:

1. Simple Object Access Protocol (SOAP)
2. Hyper Text Transfer Protocol (HTTP)
3. Extensible Markup Language (XML)

These protocols are built to enable machine to machine interaction over a network. The SOA environments have been allowing service clients to access web services using some standard protocol through desktop PCs. However, due to the advancement of web services, service consumers can access web services using mobile devices (Rendon *et al*, 2005). This is made possible for high and low end mobile devices, using Java 2 Platform, Micro edition Web Services API and Short Messaging Service respectively. Invoking services via SMS in recent years is a growing approach to accessing web services using mobile devices as discussed in the next Section.

2.3 Evolution of SMS-based Service Invocation

The technological evolution of the Post-PC era allows us to use mobile devices for accessing web services through SMS platform. The Short Message Service (SMS) is known as the delivery of alphanumeric message to mobile phones. SMS is not inherently a wireless communication technology but it is a value-added service designed to run on long range wireless networks (B'Far, 2005). SMS is a mechanism that started off from basic use i.e. sending and receiving text messages among mobile users. This was for exchanging small messages like "See you at 8.30 tonight at xyz". SMS is particularly suited for these kinds of short messages because SMS is much cheaper than calling someone and conveying the same message.

SMS has gained popularity and its usage has led service providers or businesses to employ SMS as a mechanism not just for sending and receiving messages between mobile users which is between people, but for something more useful than unidirectional communications. In recent years, service providers use SMS to support Customer Relationship activities such as receiving free newsletters, pictures, ring tones, bonus points and coupons after joining a customer program. The other service providers such as telephone companies discovered the use of SMS for customer relationship management which was used for sending their clients information on where to get cheap pre-paid phone cards when their credits are running low. This made SMS a popular marketing and branding tool. Thus service providers have always been looking for innovative ways of using SMS technology (Wan, 2008). The other uses of SMS technology discovered were location-based services which are highly relevant for local advertising – i.e. a person might receive a text message with directions to the nearest restaurant or train station. However, experts using location-based services pointed out that as

this service is time sensitive, clients should receive the message when in front of the shop and not half an hour later.

In our day and age, service providers render services over the internet to achieve their business goal and to support one of the business processes. These service provider's services were accessible through desktop PCs, but the recent integration and convergence of various technologies have led service providers to seek solutions to enable users of mobile devices to access services through SMSs. The idea came about because service providers wanted their services to reach a larger market. Therefore, SMS-based service invocation came into existence. SMS-based service invocation in SOA allows service clients (mobile user) to invoke services from a service provider by requesting and retrieving content via SMS. These services are mostly information services such as news, weather, traffic, market rates and horoscopes. The receiver of the services would pay little or nothing for this relevant and personalised information. This is how SMS-based service invocation was born.

2.4 Architecture for SMS Invocation of Services and its Challenges

Section 2.2 has shown the evolution of SMS which led service providers to seek solutions that would enhance their businesses by using SMS as a platform for rendering services to service consumers via mobile devices. In this Section, we focus on one of the businesses to clients (people) solution based on SMS (SMS-based service invocation). SMS-based service invocation environments allow the mobile device (i.e. service clients) to send a message containing some service parameters to the SMS centre (SMSC) as depicted in figure 2.2.

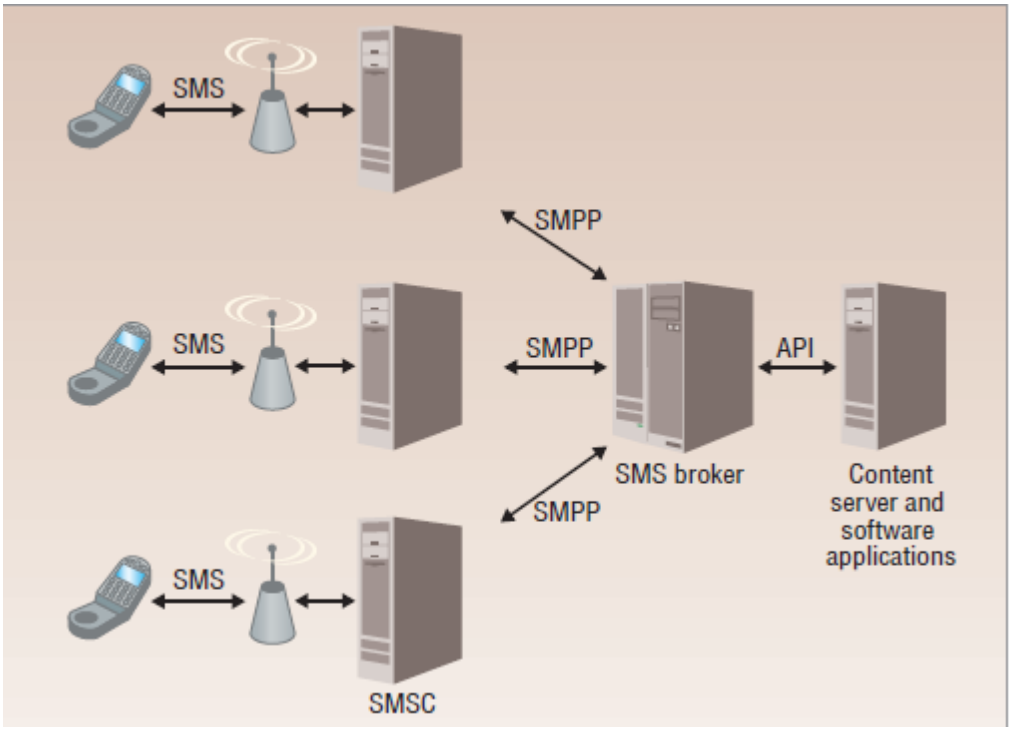


Figure 2.2 SMS based service invocation environment (Brown et al, 2008)

The SMSC uses the store and forward paradigm and relays the message to the SMS broker. An SMS broker is a business entity that negotiates an agreement with network providers. It acts as a middleman providing access to a cellular network for messaging services to third parties who have no direct relationship with the network carriers. The broker uses Short Message Peer-to-Peer (SMPP) protocol to maintain connection with carrier networks and also provides access to the content from service provider servers using custom APIs written in java, php and so on. Before a SMS broker relays the service request to the service provider, the SMS broker has to translate SMS requests that came from SMSC, to HTTP requests and then direct them to the service provider using some scheduling scheme. Content providers are entities providing value added services. This is a component that responds to a service request depending on the service parameter provided by service clients. Service parameter

refer to what is needed to invoke particular service. From the model presented in Figure 2.2, Risi *et al* (2009) developed Mobiledeck which enhanced the SMS experience by adding a Graphical User Interface (GUI) to it. The driving issue was that SMS lacks a friendly UI. The key idea behind Mobiledeck is to provide an engaging front end environment for requesting and receiving services via SMS. On the client side, it consists of a mobile application capable of displaying both textual information and graphics using predefined layouts that are accessed through instructions contained in an SMS. In other words, every time the user requests a service, the application sends an SMS that is received by a specific server and the response is redirected to the application. The limitation of this system is that there is so much SMS traffic propagated by one service request. One such example is an input screen interface which is responsible for taking a service name specified by an end user and sending it via SMS. The response will be a list of services of the same type displayed in a list view interface but coming from different service providers. The service consumer then selects a service that suits them well and the selection is sent via SMS. The results are then displayed on a result screen interface. This means there is less optimisation in the number of SMS requests per service as well as data exchange and this may lead to an SMS broker dealing with large numbers of requests hence resulting in an overloading of the SMS broker and on the part of the service providers. This poses a performance challenge in such distributed systems, such as an SMS based service invocation system. Scholars like Ramana *et al* (2005) and Cheung *et al* (2006) used load balancing as one technique for addressing the performance challenge faced by distributed environments.

2.5 Load Balancing

Load balancing is defined as the process that evenly distributes traffic among computers (i.e. servers) thereby solving overloading or congestion so that no single computer is overwhelmed. This means the goal of load balancing is to intelligently distribute requests or traffic or workload to the most appropriate server i.e. a server with a minimum workload, thus improving system performance by maximising throughput, utilising available resources and minimising execution time (response time). In conventional load balancing, there are three types of participants; clients, the load balancer and service providers servers. Clients send a number of service requests to a load balancer. The load balancer is responsible for evenly distributing the service requests to the servers based on a load balancing algorithm implemented, as depicted in Figure 2.3. The load balancer acts as a load scheduling mechanism and in so doing is one of the entities that improves distributed system performance (Kopparapu, 2002). The servers are one of the entities that receive service requests distributed by a load balancer, and then process the requests and respond accordingly.

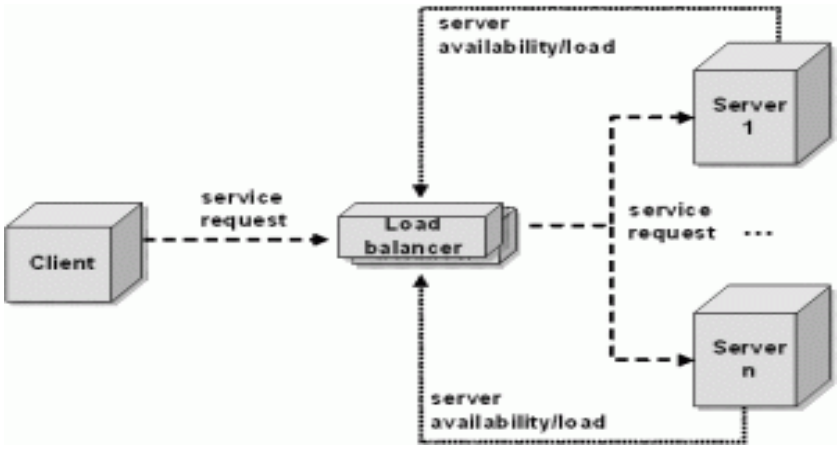


Figure 2.3 Conventional balancing (Roth et al, 2008).

Load balancing is not the only technique that can be used to address or deal with performance degradation posed by handling bursts of requests in dynamic environments such as SMS based service invocation environments. There are also techniques such as scaling up and scaling out. The scaling up technique involves adding more resources such as memory and faster CPUs to an existing computer or server. Scaling out involves adding more computers to the existing infrastructure. These techniques are feasible if the service provider has a solid plan for future growth. However, scaling up and scaling out techniques are expensive and also limited because scaling up can only occur to a certain degree. Employing scaling up and scaling out techniques may lead to resource underutilisation. Load balancing is a technique that tries to utilise existing resources by finding a way to deal with growth by distributing load around a group of servers. Among these techniques, load balancing is appropriate because it is widely used as Bourke (2001) and Kopparapu (2002) have stated.

In order for load balancing to better achieve its goal, in the past decades, a lot of research has focused on the development of effective load balancing schemes for distributed computing environments. Load balancing schemes can be classified into static and dynamic approaches. (1) Static load balancing approaches assume that a-priori information about all the characteristics of the computing resources and the communication networks are known and provided. Load balancing decisions are made deterministically or probabilistically at compile time and remain constant during runtime. The static approach is attractive because it is simple and requires minimal runtime overhead. However, it has one major disadvantage, it assumes that the computing resources and communication networks are all known in advance and remain constant. Such an assumption may not apply to a distributed environment. As a static

approach cannot respond to the dynamic runtime environment, it may lead to load imbalance on some resources and significantly increase the job (request) response time.

(2) Dynamic load balancing approaches attempt to use the runtime state information to make more informed decisions in sharing the system load. However, dynamic schemes are used often in modern load balancing due to their robustness and flexibility. Scholars like Sharma *et al* (2008) and Chhabra *et al* (2006) give classification schemes for dynamic algorithms which differ in number and type of parameters. Lists of common parameters that can be used to characterise most dynamic load balancing schemes are:

(i) Centralised vs. Decentralised. An algorithm is centralised if the parameters necessary for making the load balancing decision are collected at, and used by, a single resource i.e. only one resource acts as the central controller and all the remaining resources act as slaves. The centralised approach is more beneficial when the communication cost is less significant e.g. in the shared-memory multi-processor environment. Its limitation is a single point of failure and non-scalable. However, in decentralised approaches all the resources are involved in making the load balancing decision. Decentralised algorithms are more scalable and have better fault tolerance.

(ii) Cooperative vs. Non-cooperative. An algorithm is said to be cooperative if the distributed components that constitute the system cooperate in the decision-making process. Otherwise, it is non-cooperative.

(iii) Adaptive vs. Non-adaptive. If the parameters of the algorithm can change when the algorithm is being run, the algorithm is said to be adaptive (to the changes in the environment in which it is running). Otherwise, it is non-adaptive.

From this stance, load balancing approaches are now made to be aware of the classes of client requests they distribute because the service market deals with different types of clients' requests requiring different service quality.

2.6 Classification and Prioritisation of Service Clients

The service providers in the past have been providing services to service clients on a best effort model, which treats all service clients uniformly, without any type of prioritisation. However, because client needs may differ, it is impossible for service providers to assign the same priority to all their clients. Therefore, it is essential for service providers to render service client requests with different levels of Quality of Service, because each service consumer may require different service quality at different times to fulfil needs which may or may not be urgent.

Prioritisation is defined as a particular order, or sequence, in which service clients can be served by service providers. Prioritisation is based on a predetermined assignment of value, or importance, to different types of service consumers. Consumers with higher importance in the service market are called premium clients. Moreover, the clients with no or less importance are regular clients. The premium clients are clients served with guaranteed performance while regular clients are clients served with best effort. However, prioritisation can be done differently depending on the policy implemented by the service provider.In this work, we defined policy as a set action(s) to be performed to achieve particular task.

2.7 SMS Broker

Due to the dynamic nature of SMS-based service invocation environments, sudden load peaks can be generated by service requests. This has posed performance challenges to infrastructures for supporting SMS-based service invocation. To address this problem, scholars like Ramana *et al* (2005) adopt load balancing techniques.

Ramana *et al* (2005) proposed a load balancing approach for SMS broker that act as load dispatching mechanisms. This approach is a queuing approach which implements a store and forward mechanism. The forwarding mechanism is a load distributor that implements simple load balancing schemes such as the round-robin (RR) scheme which distributes load around service provider servers iteratively. We argue that using RR which does not consider any system state information may lead to poor load distribution decisions, thus affecting the system performance. Based on this flaw, RR is not an appropriate load balancing technique for dynamic service-oriented environments. In this work, we seek to find a load balancing approach that is well-suited to such environments and can accommodate service-oriented markets having different clients requiring differing service quality.

2.7.1 Typical SMS Broker Functionalities

An SMS broker is a component that functions between the Short Message Service Centre (SMSC) and service provider(s) where service providers mainly provide particular services as shown in Figure 2.2. SMS brokers act as a transport or scheduling mechanism. Service consumers who are mobile users can pull the services from the service provider like news, cricket scores or astrology using SMS. The general operation of the SMS broker is to bring together different parties as shown in figure 2.2

Risi and Teofilo (2009) and Lin *et al* (2009) defined SMS broker intervention as follows:
A mobile originated (MO) request from a service consumer, reaches the mobile switch centre. From the mobile switch centre, it reaches the short message service centre (SMSC) (Ramana *et al*, 2005). SMSC acts as the client to the SMS broker which serves the request. The SMS broker acts as a gateway and requests a dispatcher for service provider resources. The SMS broker interprets the request, fetches the content from the service provider and delivers it to the mobile device through the SMSC. Thus, the SMS broker performs the following tasks:

- Receives the request from the SMSCs using SMS protocols mainly Short Message Peer to Peer [SMPP].
- Translates SMS requests that come from SMSC to HTTP requests and then directs them to the service provider's side. The service provider processes the request and responds accordingly.
- Processes the response.
- Delivers the response to the requested SMSC.

2.8 Chapter Summary

In this Chapter we have introduced and discussed the basic concepts in SMS based service invocation environments. The evolution of SMS based service invocation environments, load balancing, brokerage in an SMS-based service invocation environment have also been introduced. There is a need to explore load balancing approaches and also requests prioritisation approaches. In the next Chapter, we critically analyse the existing load balancing approaches. The focus will be on how load balancing approaches can deal with the dynamic nature of SMS based service invocation environments in order to achieve better performance because current SMS based service invocation environments are facing performance challenges (Lin *et al*, 2009).

Chapter 3

Literature Review

3.1 Introduction

The nature of service oriented environments has made SMS-based service invocation feasible by enabling mobile users to access the wealth of applications which were otherwise only accessible through personal computers. However, in Chapter one we have stated SOC environments are by nature dynamic and composed of autonomous entities, which makes them unpredictable and difficult to manage. In such environments it is hard to predict how the entities involved are going to behave at a particular time (Papazoglou *et al*, 2007). In SOC environments, service consumers are known to behave unexpectedly. This implies that service providers may receive very few requests for a given service at one time, and then subsequently receive many requests at other times such that the service provider might struggle to cope with the requests. This results in overloading of service providers' servers thus posing a performance problem in service provider infrastructure that hosts the service being requested. Besides the dynamicity of SMS-based service invocation, the problem is exacerbated by the fact that SMS technology is a cheaper service technology (Jamil *et al*, 2008; Wan, 2008) and hence is able to reach a larger market of service consumers (Rendon *et al*, 2005; Risi *et al*, 2009).

To address the performance problem in dynamic environments such as SMS-based service invocation, scaling up (Microsoft, 2003; Talkington, 2002), scaling out (Microsoft, 2003; Talkington, 2002) and load balancing (Bourke, 2001; Chandra, 2002) are techniques that can be used. The scaling up technique involves adding more resources such as memory and faster CPUs to an existing computer or server. Scaling out involves adding more computers to the existing infrastructure. These techniques are feasible if the service provider has a solid plan

for future growth. However, scaling up and scaling out techniques are expensive solutions. In addition, they are also limited because scaling up can only go up to a certain point. Choosing scaling up and out techniques may lead to resource underutilisation. Load balancing is a technique that tries to live upon existing resources by finding a way to deal with growth by better handling load among a group of servers. Load balancing is defined as a process that evenly distributes traffic among computers (i.e. servers) thereby solving overloading or congestion so that no single computer is overwhelmed. Load balancing is one of the techniques mostly used to improve performance; hence it is the most suitable solution for the problem that this work seeks to address. The load balancing technique is traditionally implemented through a mechanism known as the load balancer that acts as the front end of the service providers' servers.

The load balancer in the SMS-based service invocation environment is represented by an SMS broker which is responsible for translating SMS requests to HTTP requests and directing service requests to the most appropriate server using load balancing techniques as shown in the model by Brown *et al* (2008) in Figure 2.2 in the previous Chapter. This means SMS-based service invocation environments implement traditional load balancing techniques, with the SMS broker acting as the scheduler with centralised control of the load distribution task. In order to come up with a load balancing approach that is able to handle load in SMS invocation of service environments, Section 3.2 examines the approaches that have been proposed by other researchers in addressing performance problems in environments characterised by request overload problems. The literature is discussed according to the following aspects: *load balancing techniques and algorithms*. After surveying and then getting insights about *load balancing techniques and algorithms*, prioritisation approaches are discussed in Section 3.3. These approaches explain how a load balancing approach should be designed to guarantee different client priority, if there are

clients with different importance served by the service provider such as premium and regular clients. In Section 3.4 we present the Chapter summary.

3.2 Load Balancing Techniques and Algorithms

Load balancing is important for distributed systems where it is difficult to predict the number of requests that are generated by service clients to be issued to servers of some service provider. Load balancing provides assistance in directing requests to the most appropriate server by using load balancing algorithms or techniques which are deployed according to the topology of the deployed distributed servers. Distributed systems are still advancing in terms of technology and architectures, thus requiring new load balancing techniques to deal with these new challenges. This research work aims to find a load balancing approach which is suitable for a distributed system, such as SMS based service invocation in service oriented environments, because infrastructures for such environments are faced with performance problems resulting from request overloading .

There are two common load balancing approaches being used to improve performance in relation to load handling. These are static and dynamic approaches. The static approaches are presented in Section 3.2.1 and dynamic approaches are presented Section 3.2.2. Dynamic approaches depend on monitoring for the information they use for load balancing decision making. Monitoring approaches are discussed in Section 3.2.3. The monitoring approaches usually collect system status information based on some metrics which best present the current system situation and is also used as a deciding factor by dynamic approaches. Section 3.2.4 presents these metrics monitored and collected by monitoring approaches for the purpose of aiding dynamic approaches to make load balancing decisions.

3.2.1 Static Approaches

A static approach is one of the load balancing approaches that assumes that prior information about the computing resources (i.e. servers) is known and provided. There are various types of load balancing algorithms or techniques proposed by scholars to address scalability problems under static approaches which are presented below.

3.2.1.1 Randomised Algorithm

In a randomised algorithm, the requests are assigned to any server picked randomly among a group of servers (Motwani *et al*, 1996; Sharma *et al*, 2008; Rahmawan *et al*, 2009; Li *et al*, 2009). In such a case, one of the servers may be assigned many more requests to process, while the other servers are sitting idle. However, on average, each server gets its share of the load due to the random selection. The algorithm is known to be simple to implement but it can lead to overloading of one server whilst under-utilising others. Thus, this algorithm fails to even meet the main goal of load balancing, which is avoiding load imbalances. Hence it is not a good algorithm to employ in any given environment.

3.2.1.2 Round-robin Algorithm

In a round-robin algorithm, the scheduling or central control node assigns the requests to a list of the servers on a rotating basis (Shirazi *et al*, 1995; Chhabra *et al*, 2006; Sharma *et al*, 2008; Li *et al*, 2009). The first request is allocated to a server picked randomly from the group. For the subsequent requests, the scheduling node follows the circular order to redirect the request. Once a server is assigned a request, the server is moved to the end of the list. This keeps the servers equally assigned. The round-robin algorithm is better than the randomised algorithm because the requests are equally divided among the available servers in an orderly fashion. However, the round-robin algorithm can only work best in homogeneous

environments, meaning it is best in environments that have servers with the same capability. This algorithm may lead to servers with more capacity being underutilised.

3.2.1.3 Weighted Round-robin

The weighted round-robin algorithm is an advanced version of the round-robin that eliminates the deficiencies of the plain round-robin algorithm. In weighted round-robin, one can assign a weight to each server in the group so that if one server is capable of handling twice as much load as the other, the powerful server gets a weight of two (Rotaru *et al*, 2004; Rahmawan *et al*, 2009; Li *et al*, 2009). This means that whenever there is a powerful server in the deployment; the powerful server receives more requests than the other servers. The weighted round-robin algorithm takes care of the capacity of the servers in the group, meaning it was meant for heterogeneous servers. However, it does not consider the advanced load balancing requirements such as processing times for each individual request. It assumes that the processing times for each individual request are equal, which is particularly not true in environments where there are heterogeneous requests.

3.2.1.4 Threshold Algorithm

The threshold algorithm allows a server or service endpoint to continue receiving requests until a threshold value is reached (Mouratidis *et al*, 2006; Sharma *et al*, 2008; Rahmawan *et al*, 2009). Once the threshold value is reached, subsequent requests are sent to another server or endpoint with the same service type. The next endpoint is chosen based on a round-robin strategy. This next server or service endpoint will be used until its threshold value is reached. Then the third endpoint is selected based on a round-robin strategy, etc. However, the algorithm may lead to some servers being underutilised due to load imbalances. This raises the same problem as the randomised algorithm.

3.2.1.5 Simulated Annealing

Advancements in distributed systems in recent years have demanded better load balancing solutions. This has led to the development solutions based on optimisation theory such simulated annealing. The simulated annealing algorithm's goal is to find an acceptable solution in a fixed amount of time (Chhabra *et al*, 2006; Christoduolopoulos *et al*, 2007; Boone *et al*, 2010), rather than the best possible solution to deal with certain situations faced at a particular time. However, optimisation algorithms use the heuristic approach, hence they require more computational power which renders them not suitable for a distributed system such as SMS-based service invocation that usually belongs to one service provider with limited resources. Therefore, it might slow down system responsiveness.

3.2.1.6 Machine-learning

Machine learning, a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviours based on empirical data. There are various types of machine learning algorithms that have been used for load balancing. These algorithms are Genetic algorithm (Petrovic and Fayad, 2005; Nikravan *et al* (2007); Rosas *et al* (2007); Gondhi *et al* (2009)), Bayesian networks (Gou *et al*, 2003; Romtveit, 2010) and Reinforcement learning (Parent *et al*, 2002; Wang *et al*, 2007; Jia *et al*, 2011). These machine learning algorithms have been widely used as optimisation and search tools in several problem domains. The scholars also saw it as the best way to solve a load balancing problem by giving focus to it as an optimisation problem. Most of the load balancing research that pertains to such algorithms focused on static scenarios, that is, they are fed with empirical data beforehand because of the high computing time involved. These algorithms usually obtain an optimal load distribution solution for a particular situation but intensive computation is involved. Thus it is not suitable for small or medium scale

distributed systems which may belong to a single provider such SMS-based service invocation.

3.2.1.7 Evaluation of Static Approaches

From the analysis of the static approaches we have made, it can be observed that they are not entirely appropriate for our SMS based service invocation environment. This is because the disadvantages outweigh the advantages of each algorithm. This conclusion is based on the basic requirements needed for good load balancing. For example a good load balancing solution should maximise resource utilisation and minimise execution time but none of the static algorithms achieve that as a whole. Furthermore, not only do these static approaches fail to meet the basic requirements of good load balancing, they also do not fulfil the requirements outlined as important in this dissertation, which are adaptiveness and robustness.

3.2.2 Dynamic Approaches

The dynamic approach is one of the load balancing approaches that attempts to use the system state information collected dynamically or during runtime in order to make more accurate decisions in sharing the system load or distribution. Thus, the dynamic approach is better than the static approach which assumes too much about the system status hence leading to some resources being underutilised and load imbalances as mentioned above in a number of static approaches. Due to these major drawbacks we have shifted our survey to dynamic approaches. There are various types of load balancing algorithms or techniques proposed under dynamic approaches and they are discussed below.

3.2.2.1 Load Migration or Load Sharing

More successful and accurate load balancing requires a load balancing strategy to have some notion of the server load in order to adapt the load balancing weight to the current load. This can be done by monitoring servers' behaviour. A load migration or load sharing approach balances the load between servers by checking which ones are heavily or lightly loaded using load metrics (Cortes *et al*, 1999; Raqabani *et al*, 2005; Cheung *et al*, 2006; Dinan *et al*, 2009). The requests are moved from the heavily loaded server to the lightly loaded server (offloading). This is a well-known approach but it incurs some delay when migrating load.

3.2.2.2 Round-trip Algorithm

A round-trip load balancing algorithm performs load distribution by monitoring the time taken by clients to receive the response after sending the initial request to the server (Di Stefano et al., 1999; Bryhni *et al*, 2000; Yan *et al*, 2008; Boone *et al*, 2010). The average time taken for all requests during a sliding window is calculated and the server with the lowest calculated average load is selected. This algorithm provides the best effort service.

3.2.2.3 Least Loaded Algorithm

The least loaded algorithm allows a server to continue receiving requests if the server is less loaded then the other servers in a group based on the system's current status or until a threshold value is reached (Arape *et al*; 2003; Aimrudee *et al*, 2010; Balasubramanian *et al*, 2004). However, the efficiency of the least loaded algorithm is dependent on the selected load index in order for it to make the most appropriate load balancing decision. Given that the least loaded algorithm uses a threshold value as its criteria, if all servers are over the threshold, then the least loaded algorithm will still be able to send a request to the server with

the least load even though it has reached the threshold limit. The least loaded scheme has shown its robustness and flexibility by being applicable in environments such as networking for load balancing at the routing level (Karasan *et al*, 1998; Shan *et al*, 2001). However, the least loaded algorithm can have a possible delay depending on the monitoring approach that assists it in making a load distribution decision

3.2.2.4 Evaluation of Dynamic Approaches

The literature analysis of dynamic approaches made has shown that the *least loaded algorithm* is our candidate dynamic approach to build on. This is based on the fact that the algorithm first meets the basic requirements needed for it to work in SMS-based service invocation environment as mentioned in Section 3.1.2.7 which are fulfilled as the algorithm relies on the system status to make load distribution decisions. Moreover, the decision was also based on our selection criteria or requirements as mentioned in Section 3.1.2.7. Lastly the literature about the least loaded algorithm has shown that it is a widely commercialised dynamic load balancing approach that can function in any environment in which it is implemented, as shown in Section 3.2.2.3. For the purposes of our SMS based service invocation environment, the algorithm is best suited because it is flexible.

3.2.3 Monitoring Approaches

This Section is necessitated by the adoption of the dynamic approach to load balancing for making load balancing decisions. Dynamic approaches to load balancing depend on monitoring approaches that are responsible for supplying empirical data needed for load distribution. For the purposes of selecting a suitable way of monitoring status information there is a need to survey the existing monitoring approaches which help in aiding dynamic approaches to achieve the load balancing goal. There are two ways in which monitoring is

done; the push mode and the pull mode. These modes concern how data or information is disseminated to the mechanism responsible for load distribution (Othman *et al*, 2003; Werstein *et al*, 2006; Shadrach *et al*, 2009):

- The push mode disseminates system state information at certain time intervals regardless of whether it is needed or not. Either a central node (scheduler) or all other nodes (servers) receive and store the data. It is simple to implement and incurs little overhead. However, the shortcoming of the push mode is that it is not an easy task to set the time period to disseminate system state information. Setting long time periods can make the load balancing scheme work with out-of-date system state information and short time periods introduce communication overhead.

- The pull mode disseminates system state information on demand (i.e. information is collected dynamically when there is load to be distributed otherwise no information is collected). However, the problem with this approach is synchronisation with the load balancing scheme (transfer policy) which may lead to minor delays. This approach would be best for SMS-based service invocation environments owing to the fact that it provides accurate system status information, which is not outdated, and is collected dynamically during load distribution.

3.2.4 Load Monitoring Metrics for Dynamic Approaches

Given that monitoring approaches are responsible for gathering certain information for load balancing, scholars use load metrics to complement the process and also to make good load balancing decisions. The following are metrics used for load balancing decision making: CPU utilisation, memory utilisation, IO utilisation and Bandwidth occupancy (Qin *et al*, 2003; Li *et al*, 2009; Wang et al, 2009). These load metrics can be selected or used together

based on the type of load the application or client requests exert (Wang *et al*, 2009). In our case, we select CPU utilisation as the monitored load metric because the client requests are more demanding on computational resources. Moreover, it is a well-known load index with minimum overhead (Kunz *et al*, 1991; Zhang *et al*, 2000; Xiao *et al*, 2002).

3.3 Prioritisation Approaches

In a service oriented environment such as GUISET, service guarantees are enforced because service providers deal with different types of clients that need different service time i.e. premium and regular clients. This means client requests served by service providers are not the same. The service provider needs to provide a mechanism to allow the system to know who to serve first before applying load balancing. The mechanism incorporates service prioritisation with load balancing so that the load balancing mechanism is aware of the different classes of client requests when load balancing. Requests from premium clients are given priority over regular clients. This Section discusses some request-prioritisation techniques, which are of interet to this study.

3.3.1 Over Dimensioning of Resources

The over dimensioning of resource approach provides a dedicated server for clients who want a much quicker service time (Yigitbasi *et al*, 2010; Bernado *et al*, 2003; Boone *et al*, 2010). This approach is usually prepared when service providers are setting up their infrastructure and the service provider is already aware that they will be serving different classes of client requests (i.e. premium and regular). The over dimensioning of resources takes place such that it meets the performance requirements promised by the service provider. However, the approach is expensive. Moreover, in order to utilise the over dimensioning of resources

approach there must be some mechanisms for classifying client requests before they are directed to the dedicated server which is responsible for handling client requests.

3.3.2 Admission Control

The admission control approach is a set of prioritisation criteria that maintain the quality of services for clients who need it the most (i.e. premium). This is done by denying access to some client requests that require service when the system is overloaded based on a set threshold value. While at the same time client requests which require a higher quality of service are allowed to continue to be processed (Roberts *et al*, 2001; Cherkasova *et al*, 2002; Muppala *et al*, 2011). This selective dropping of some requests which require service (i.e. regular) contributes to reducing the server load and also to meeting the SLA for premium clients. However, this approach is susceptible to sudden load change, hence suitable to dynamic environments.

3.3.3 On Demand Resource Reservation

The on-demand resource reservation approach emanated from real world scenarios. For example whenever somebody wants to use a venue for lecturing he/she needs to place a booking for a spot at a particular time which is basically known as a reservation. With the on-demand resource reservation approach, the service provider has resources reserved for particular clients such as premium clients, and as long as those clients are not present regular clients can use or scavenge the resources reserved for premium clients until premium clients arrive (Foster *et al*, 2000; Ali *et al*, 2002; Smith *et al*, 2000; Nurmi *et al*, 2008). However, if a premium client comes at a faster rate than the regulars come, the migration of regular clients occurs. This approach may cause delay hence an increase in response time for premium clients.

3.3.4 Priority Based Load Balancing Approaches

These approaches tie the two concepts together, but what we are looking for is how to classify and serve requests according to their priority if there are two classes of requests. So the focus in this approach is on how prioritization is done before getting to the load balancing process. The prioritisation criteria states that requests requiring best effort service (regular requests) are handled when there are no client requests who need guarantees (premium) otherwise if there are premium clients then regular requests wait until they are finished (Shan *et al*, 2002; Bic *et al*, 2002; Zhou *et al*, 2005; Zhang *et al*, 2009). However, this approach is more likely to result in starvation and the fulfilment of premium clients is achieved through the trade-off of regular client fulfilment.

3.3.5 Evaluation of Prioritisation Approaches

Based on the literature surveyed, we selected the over dimensioning of resources approach over others because it enables a high quality of service for premium clients. It meets the needs of premium clients by providing dedicated servers for premium requests in order to meet their SLAs (Boone *et al*, 2010). We also selected prioritisation criteria to complement the over dimensioning of resources approach which assists in categorising client requests so requests are directed to the appropriate server. This means we are combining these solutions so that better performance can be achieved.

3.4 Chapter Summary

In SMS-based service invocation environments, load balancing addresses the problem of unexpected traffic and sudden load peaks which come from service consumers. Service providers also need to guarantee a standard of performance. Apart from the fact that we need a load balancing approach that handles scalability in such environments, there is another need to be considered in such environments; how to incorporate a client categorisation mechanism in the load balancing approach that will influence load balancing and be able to serve clients according to client category. A survey of related works enabled the researcher to select approaches that can be selected to achieve the goal of this work. Based on the insight gathered from the literature it was decided to model our approach such that it will address the problem raised in Section 1.2. The following Chapter will discuss the modelling of the solution approach for this work, that is to find a load balancing approach that functions in SMS-based service invocation environments and also serves clients based on their respective categories.

Chapter 4

Design of the Client-Aware Least loaded Load Balancing Framework

4.1 Introduction

Infrastructures for SMS-based service invocation environments are still faced with performance problem with respect to a number of requests that can be handled by the system without degrading system performance. In Chapter 3, we have analysed existing approaches towards addressing the aforementioned issue. Based on the dynamic nature of SMS-based service invocation environments, we have identified a need to develop an adaptive and dynamic load balancing framework that improves performance in such environments. Therefore the introduction of abnormal behaviours such as erratic client requests can be handled while still ensuring adequate performance. We also saw a necessity for the proposed load balancing framework to be designed such that it is aware of the different clients' quality of service expectations since service oriented environments deal with different type of clients (premium and regular).

This Chapter is organised as follows: Section 4.2 proposes a load balancing solution that monitors and handles load in SMS-based service invocation environments. It also discusses the metrics that need to be monitored in order to support load balancing decision making. Section 4.3 discusses how the proposed load balancing approach was designed to guarantee different client prioritizations. Section 4.4 presents an algorithm which combines aspects discussed in Section 4.2 and 4.3 to address the performance problem in SMS-based service invocation environments. A description of our design criteria, the framework's components and how the components interact is presented in Section 4.5. Section 4.6 discusses how the

proposed load balancing framework can be used. Section 4.7 gives the Chapter's concluding remarks.

4.2 Load Balancing Model

This Section presents two important aspects that this study uses to address the performance problem faced by SMS-based service invocation environments. The discussion starts in Section 4.2.1 by proposing a load balancing approach that monitors and handles load in SMS-based service invocation environments. Section 4.2.2 discusses the metrics that need to be monitored to support load balancing decision making. The goal is to discuss how an appropriate load balancing solution can use load metrics to address performance problems in SMS-based service invocationenvironments.

4.2.1 Load Balancing Approach for SMS-based Service Invocation Environments

The first aspect that this work focused on as a means of improving performance in SMS-based service invocation in service oriented environment. The work covered come up with a load balancing approach that monitors and handles load in such environments. There was a need to find a way of dealing with the dynamic nature of such environments, which result in a lot of unpredictable behaviour from entities involved in these environments. For example, during SMS invocation of services, an abnormal behaviour from service consumers (mobile users) can occur where they all involuntarily decide to invoke a given service. This results in a burst of requests coming to a service provider. This work argues that having a queuing approach implementing RR solely as a load handling mechanism is not a good solution in such cases. This is due to that the approach is static in nature which means system status is assumed before requests are distributed. These assumptions may not apply in dynamic

environments because they may lead to load imbalance on a service provider's servers and significantly increase the requests' response time (Goyal and Patel, 2011).

In addressing the aforementioned challenge, this research work developed an adaptive and dynamic load balancing approach that deals with the nature of SMS-based service invocation environments. In doing so, this work explored the *least loaded load balancing approach* (Balasubramanian *et al*, 2004; Aimrudee *et al*, 2010; Othman *et al*, 2003; Rosas *et al*, 2007), which identifies the building blocks required for addressing performance problem in relation to scalability in dynamic environments. This approach was chosen because of its dynamicity, adaptiveness and performance in working with stress load (Balasubramanian *et al*, 2004). In the work done by Aimrudee *et al* (2010), the *least loaded approach* outperformed the Minimum, Threshold, Random, and Round-robin approach. The least loaded load balancing approach can be manoeuvred to suit any kind of load handling requirements and thereby providing flexibility (Karasam *et al*, 1998; Othman *et al*, 2003; Aimrudee *et al*, 2010). The least loaded approach has a request transfer policy, which determines whether a particular service provider's server is suitable for receiving requests. This transfer policy is supported by an information policy, which guides how information about the each service provider's servers should be disseminated. The information policy aids in the decision making when requests have to be distributed (i.e. which server should receive requests). The information policy ensures collection of service provider's servers information using a periodic approach (push mode) or on-demand driven approach (pull mode) as discussed in Section 3.2.3. The least load algorithm ((Othman *et al*, 2003); (Balasubramanian *et al* (2004) psuedo code is shown in Figure 4.1.

***Inputs**: requests*

a) Set algorithm for requests distribution
b) Get status information

While (servers available)

If (threshold is not reached in all servers)

c) Distribute request to least loaded server based on status information published using load information policy

Else

d) Retry distribution process until threshold value of at least one server has not been reached

End if

End while

Figure 4. 1 Least Loaded Algorithm

The work reported in this dissertation employs the on demand approach because it exerts less overhead (Arape *et al*, 2003; Othman *et al*, 2003). The information policy mechanism collects information about monitored load. The choice of the load metrics depend on the context of application, and in the case of this work load, metrics which are applicable to an SMS-based service invocation environment were chosen (See Section 4.2.2). The collected information is used by the transfer policy for load (requests) distribution. The following Section discusses load metrics, which were used for monitoring load on service providers' servers in this study.

4.2.2 Metrics for Load Balancing Purposes

In addressing the aspect of finding metrics needed to monitor load for load balancing in SMS-based service invocation environments, this work investigates the type of metrics used for load monitoring in general with the aim of finding metrics that will be appropriate in SMS-based service invocation environments. The common metrics used to monitor load at different levels for load balancing decisions are namely CPU utilisation, memory utilisation, IO utilisation and Bandwidth occupancy (Qin *et al*, 2003; Li et al, 2009; Wang *et al*, 2009). These load metrics can be selected or used together based on the type of load the application or client requests exert (Wang *et al*, 2009). In our case, we selected CPU utilisation as our metric to monitor load because we assumed that client requests are more demanding on computational resources. Moreover, it is a well-known load index with minimum overhead (Kunz *et al*, 1991; Zhang *et al*, 2000; Xiao *et al*, 2002)

From the aforementioned, the CPU workload index (Kunz *et al*, 1991; Zhang *et al*, 2000; Xiao *et al*, 2002) was the metric used to measure the load status of service provider servers in this study. CPU workload index is collected by the information policy and forwarded to the transfer policy for load distribution decision making. The CPU workload index is given by dividing CPU utilisation (CPUu) by CPU capacity (CPUc). Besides the use of CPU workload index this research also used the CPU idle time metric. These metrics were chosen based on the findings by Kunz *et al* (1991), Zhang *et al* (2000) and Xiao *et al* (2002), who reported CPU workload index as the most effective load index metric with a minimum overhead. The other reason is Grosu *et al* (2002) and Nandagopal *et al* (2010) reported that when monitoring load for any server machine, the calculation used to get load metric should not be computationally expensive.

4.3 Client Categorisation Model

Service oriented markets nowadays deal with different types of clients who require different service quality. The service quality requirements along with system response time that is visible to clients are used to measure satisfaction. In an environment such as GUISET that implements a pay-per-use business model (Buthelezi *et al*,2008), clients that require higher service quality are classified as premium and they are served with guarantees. The clients that do not really care about higher service quality as long as they get service delivery at the end of the day are classified as regular and they are served with best effort. It therefore follows that an important capability of our proposed approach is to guarantee different client priority if we have different client categories.

In addressing the aforementioned objective as a means of designing a load balancing approach that can classify client requests before load balancing takes place, a client categorising mechanism had to be introduced. In Chapter 3, the literature review explored various clients' prioritisation approaches and came up with the over dimensioning of resources approach as the preferred technique for dealing with premium clients which require service quality. The over dimensioning of resources approach, provides dedicated servers for premium clients' requests in order to meet their SLAs (Yigitbasi *et al*, 2010; Bernado *et al*, 1998). This approach was preferred over others because it enables a high quality of service which is needed to keep premium clients satisfied. In this work, premium requests are routed to dedicated servers only, unless regular client servers are underutilised. Although in this work, over dimension of resources approach aspect forms part of the solution but it evaluation is left for future work

We considered an alternative prioritisation approach as required by the over dimensioning of resources technique and this prioritisation approach is there to complement the over

dimensioning of resources technique. The preferred prioritisation approach determines the requests that should be served according to their class of importance (Bic et al, 2003; Teixeira *et al*, 2004; Muppala *et al*, 2011; Zhang *et al*, 2007). This means the load balancing approach proposed in this work consists of a prioritisation approach.

To introducing a priority-based approach to the proposed load balancing model, the client's request arrival rate in the priority-based approach is given by R_i, which is understood by the SMS broker. This SMS broker is responsible for load scheduling of these requests to service provider servers. The requests' arrival rate becomes the sum of two classes of requests (i.e. premium and regular), which is represented by $R_i=R_i^p+R_i^r$, where R_i^p denotes arrival rate of premium requests and R_i^r denotes arrival rate of regular requests. As stated in the literature, premium requests have higher priority over regular requests (Teixeira *et al*, 2004). To differentiate between these two classes we used numeric values to denote priority and it is derived from each request (premium or regular) by a specific function $P=Priority(R_i)$ where P denotes numeric value that indicates the level of priority of a certain client request. In this work, we described a request with a higher numeric value P as of the type high priority and with lower numeric value P as of the type lower priority which then maps premium and regular clients respectively. What follows is an example that demonstrates how the prioritisation mechanism functions; suppose we have the following request sets, $R^p = \{ R_1^p , R_2^p , R_3^p \}$ and $R^r = \{ R_1^r , R_2^r \}$ with a request arrival rate of R_i. R^p is a set of premium requests and R^r is a set of regular requests. The premium requests R^p are handled first until they are finished whether regular requests are queued or not (Teixeira *et al*, 2004). This is done by sending premium requests to the servers dedicated for them with the assistance of the over dimensioning of resources approach. In the event that the regular client dedicated servers are

being underutilised, then premium requests R^p can be sent to the regular requests' dedicated servers. Regular requests R^r are only handled when there are no premium requests queued.

4.4 Realisation of Client-aware Least Loaded Load Balancing Approach

The goal of this research is to design an effective load balancing approach with prioritisation for an SMS based service invocation environment. This is being done with the aim of mitigating performance problem faced by SMS-based service invocation environments in relation to scalability. The approach consists of aspects discussed in Sections 4.2 and 4.3 these aspects are load balancing and categorisation of clients' requests in SMS-based service invocation environments. These approaches are combined to form one approach to suit the goal of this research. Section 4.4.1 presents design criteria for the client-aware least loaded load balancing approach and from these design criteria we derived the client-aware least loaded load balancing algorithm which is discussed in Section 4.4.2.

4.4.1 Design Criteria for Client-aware Least Loaded Load Balancing Approach

In order to make the proposed load balancing solution suit our goal and take care of the aspects discussed in Section 4.2 and Section 4.3, this Section presents the design criteria that influenced it. These include: Adaptive Load Handling Scheme, Provision of Load Monitoring Assistance, and Client Classification.

4.4.1.1 Adaptive Load Handling Scheme

The environment we are dealing with is dynamic which implies that such an environment is capable of manifesting unpredictable behaviour. During SMS invocation of services in service oriented environments, an abnormal behaviour from mobile users can lead to congestion of traffic on service providers' servers. In order to address this issue, the

environment needs an adaptive load handling mechanism which will adapt to change in the environment and make decisions based on the system's current status. This load handling mechanism should be able to make load balancing decisions and also adjust load distribution accordingly i.e. make decisions based on particular situations occurring and then determine where to distribute the load.

4.4.1.2 Provision of Load Monitoring Assistance

The adaptive load handling mechanism must be assisted in making fair load distribution decisions by monitoring the current state of the service providers' server resources (Grosu *et al*, 2002). This gives the added advantage to the adaptive load handling mechanism in that status information of providers' server resources is provided, and thus the load handling mechanism does not have to guess or make assumptions when distributing load.

4.4.1.3 Prioritisation Based Client Classification

In order to ensure client satisfaction, a mechanism that will be responsible for differentiating client requests according to their category is needed. Clients are either premium clients who are served with guarantees or regular clients who are served with best effort service. Against this background, it is imperative to classify client requests before employing load balancing processes because the premium requests require different service quality. Furthermore, in order to satisfy premium clients, there must be prioritisation mechanism that will give preferences to premium clients.

4.4.2 Proposed Client-aware Least Loaded Algorithm

In order to meet the aforementioned design criteria, the algorithm should be made up of three aspects which are discussed below: Load Monitoring Mechanism, Adaptive Load Handling Mechanism, , and Client Classification Mechanism.

4.4.2.1 Load Monitoring Mechanism

The load monitoring mechanism, which functions as the load information policy, should reflect the current CPU load index of the service provider servers. Traditionally, the load of a service provider server at a given time is described simply by CPU load which is CPU utilisation divided by CPU capacity. CPU load refers to the number of requests or processes which are either being executed or waiting to be executed. In the proposed algorithm, CPU load and CPU idle time are used as load monitoring metrics. The system statistics, such as CPU load and CPU idle time of a node, change during the life of processes. For example, the CPU utilisation may be high in one second but low in the next second. Therefore, it is reasonable to get these statistics on demand so that out-of-date server host state information is avoided. CPU load, CPU utilisation (CPUu), CPU capacity (CPUc), and CPU idle time are considered as load information parameters to measure the load of a service provider server node. The following equations are used to calculate the metrics ((Bic and Shaw, 2002); (Zhang et al, 2000) and (Wang *et al*, 2009)):

$$CPUload = CPUu / CPUc \tag{4.1}$$

$$CPU\ idle\ time = CPUc - CPUload \tag{4.2}$$

In order to illustrate how the above formulae works and if more detail are needed then refer to Bic and Shaw(2002), Zhang et al(2000) and Wang *et al*(2009).

Lets take an example, Given two servers S1 and S2 where S1 have *CPUc* =100 and *CPUu* =40 and also S2 have have *CPUc* =100 and *CPUu* =90.Calculating which server is suitable to execute following task.

S1 is *CPUload= 40/100=0.4 since CPU idle time is in % so CPUload *100 give 40 and CPU idle time=100-40=60%*

S2 is *CPUload= 90/100=0.9 since CPU idle time is in % so CPUload *100 give 90 and CPU idle time=100-90=10%*

From the above the calculation S1 is preffered over S2 based CPU idle time.

4.4.2.2 Adaptive Load Handling Mechanism

The adaptive load handling mechanism acts as the transfer policy that implements the least loaded load balancing approach. The decision for distribution is made as follows; the requests are going to come from the client's prioritisation policy. These requests are served based on their priority and distributed among available service providers' servers. The distribution of requests to service providers' servers is allowed until a threshold value is reached. This decision is made based on the status information of service providers' servers published by the load information.

4.4.2.3 Client Classification

The client classification mechanism complements the prioritisation policy. The concept of priority requires that we decide which requests should be served first as discussed in Section 4.3. Basicaly premium and regular requests arrive concurrently and classifying mechanism will distiguish these requests.The requests placed in separate queues and the prioritization function determines which requests should go through first.In this case,the premium requests are served first based on priority determined by the prioritization function. We used numeric values to denote the priority level of a request. Higher values denote higher priority and lower values denote lower priority. To aid the above explanation of the classification mechanism see figure below.

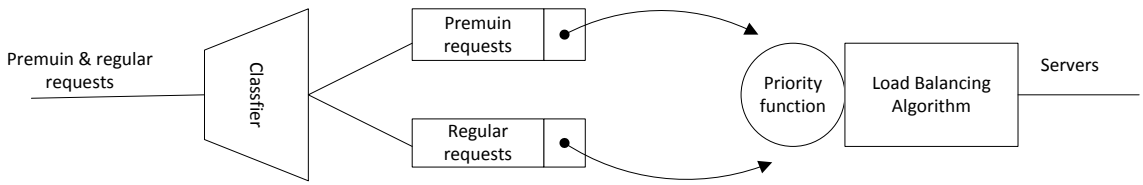


Figure 4. 2 illustrate client classification mechanism

From the specification of the algorithm discussed above, Figure 4. 3 shows the Client-Aware Least Loaded Load Balancing Algorithm:

***Inputs**: premium or regular clients' requests*

1. *Prioritisation policy receives requests from external entities*

//premium or regular clients' requests

a) *Classifies requests according to their priority using priority function*
b) *Class 1 will have premium status, Class 2 will have regular status*

//Ready to be sent to transfer policy

While (premium not empty)

c) *Send Class1 requests to transfer policy*

End while

d) *Otherwise send class2 requests to transfer policy*

2. Transfer policy receives requests from prioritisation policy

//premium or regular clients' requests

e) *Set algorithm for requests distribution*
f) *Get status information*

While (servers available)

If (threshold is not reached in all servers)

g) *Distribute request to least loaded server based on status information published using load information policy*

Else

h) *Retry distribution process until threshold value of at least one server has not been reached*

End if

End while

Figure 4. 3 The Client-Aware Least Loaded Load Balancing Algorithm

4.5 Client-aware Least Loaded Load Balancing Framework (CaLLF)

This Section presents a framework aimed at providing a platform for the algorithm discussed in Section 4.4. We coined this platform the Client-aware Least Loaded Load Balancing Framework (CaLLF). To ensure maintainability, a modular design approach was followed. Each component of the algorithm is a separate and independent software module. Section 4.5.1 presents the components of CaLLF and how they are integrated to form the CaLLF framework.

4.5.1 Framework Components

The framework's main components are identified as the client classifier, load balancing decision maker and load monitor. These components are derived from the algorithm presented in Section 4.4. The components interact together through event based communication. This communication style allows the communicating components to be loosely coupled and independent from each other. It also promotes scalability (Mühl *et al*, 2006) through allowing the system to grow without any effect on the other components. Given that the components communicate through an event-based communication style, it follows that some CaLLF components should be producers of events and some should be consumers of the events. To this end the load monitor component and the client classier act as producers of events that will be consumed by the load balancing decision maker component, which acts as a consumer.

CaLLF is meant to be incorporated into a load dispatching mechanism, such as an SMS broker, to serve as a load distribution mechanism.

4.5.2 CaLLF Components Interaction

Each of the CaLLF components performs some operations to support the execution of the load balancing process. The service consumer requests first go through the client classifier which categorises client requests. After classification the client classifier relays the requests to the load balancing decision maker component. The load balancing decision maker component decides which server should process the request based on the information on the resources available for each server coming from the load monitor component. Figure 4.4 shows the CaLLF framework which was derived from Brown *et al* (2008) where the SMS broker component contains our load balancing algorithm.

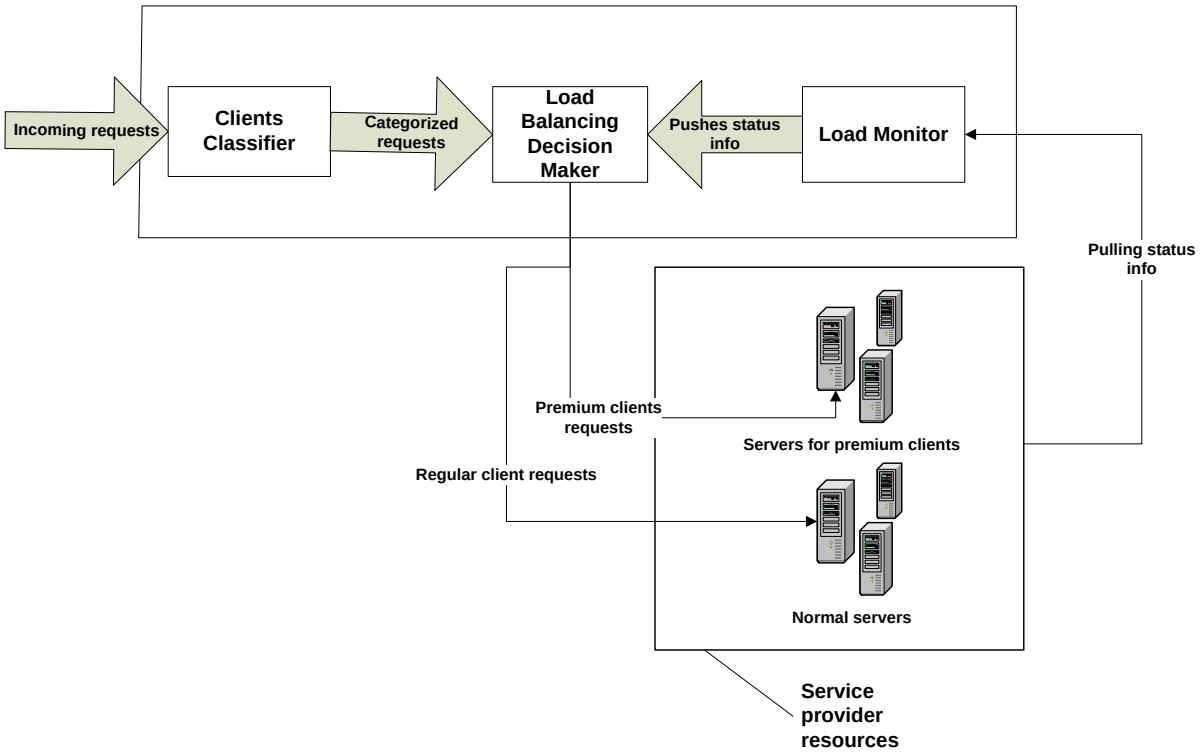


Figure 4.4 Conceptual Client-aware Least Loaded Load Balancing Framework (CaLLF)

Communication between the components is purely event-based. The fact that most communication brokers/middleware is event driven (Muhl *et al*, 2006) made it paramount for CaLLF to follow this pattern. The following sub-Section describes the load balancing framework and operations of each component in greater detail.

- ***Client Classifier***

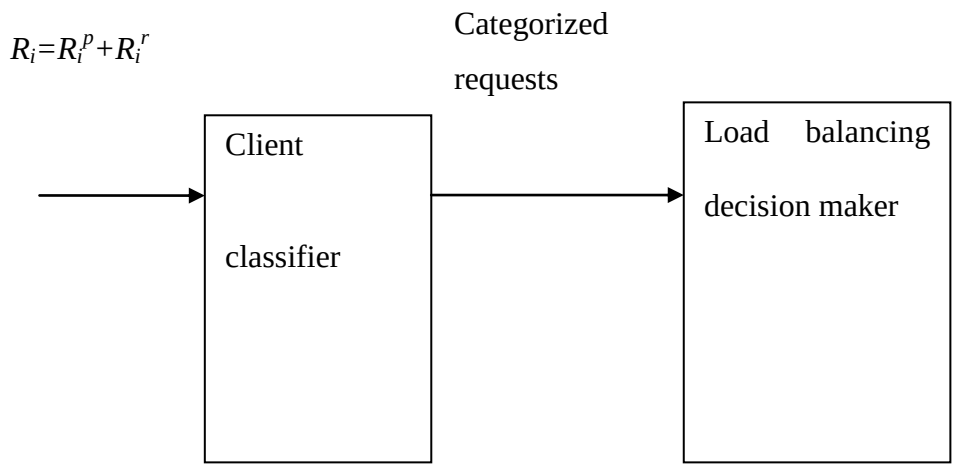


Figure 4.5 Components involved in Client Classifier.

The client classifier component is responsible for receiving requests from clients. These requests include both premium (R_i^p) and regular requests (R_i^r). Premium requests are of type high priority, so they need guarantees while regular requests are of type lower priority and they are served with best effort service. The client classifier takes these requests and categorises them using the priority function which is coordinated by a prioritisation mechanism. After classification, the client classifiers send or relay the requests according to their class of importance to the client prioritisation mechanism in the client classifier component. The client prioritisation mechanism determines which class of requests should be

served first by the load balancing decision maker component so that distribution of these requests should take place. In load distribution preference is given to premium requests.

- ***Load Balancing Decision Maker***

The load balancing decision maker is the main component responsible for the distribution of requests evenly among servers with the aim of optimising resource utilisation. The actual destination of a request is decided using the *least loaded load balancing algorithm* which distributes requests to the server with the lowest load, based on service provider server load status. The load balancing decision maker subscribes to the load monitor component to get updates on each server's load information, and uses the information obtained to check which server has the least amount of load. The load information is made available to the load balancing decision maker on demand by the load monitoring component.

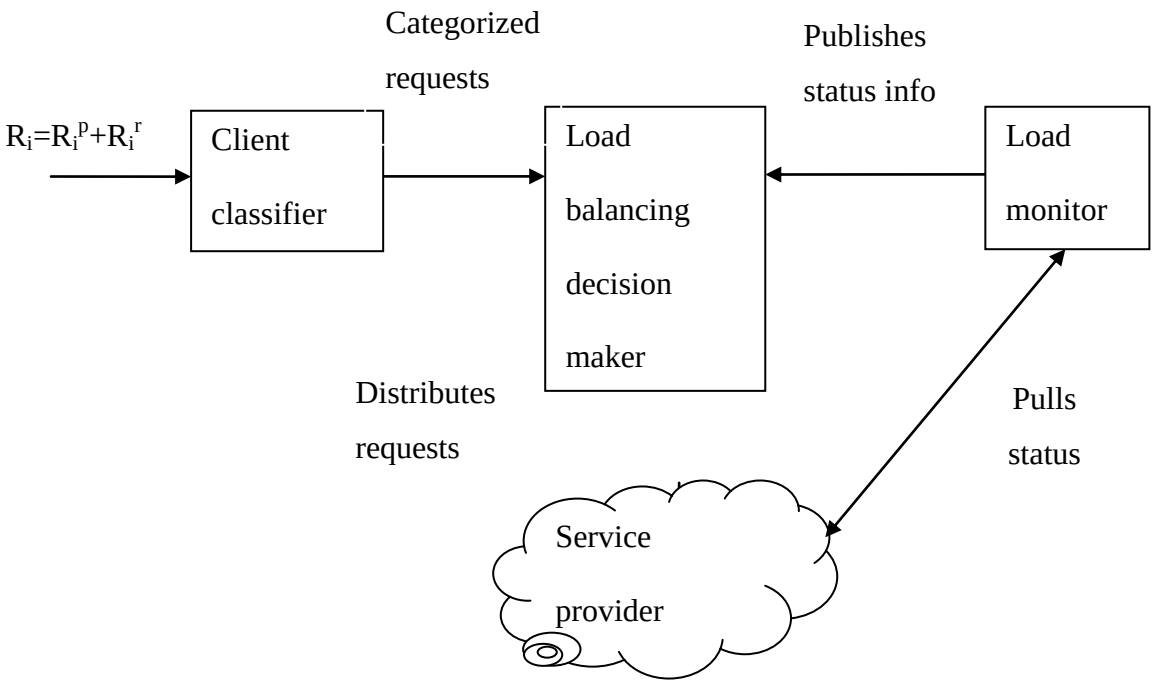


Figure 4.6 The Load Balancing Decision Maker Component

The load balancing decision maker receives requests, which include both premium and regular requests, from the client classifier. These requests are taken and distributed among available service provider's servers based on their priority and the ones with high priority are served until they are finished before regular requests can be processed.

- ***Load Monitor***

The load monitor acts as an event producer for the load balancing decision maker component by publishing monitored status information about the service provider's servers. The status information is used for making load balancing decisions based on load metrics being monitored. In the case of this work, we are dealing with a CPU bound application which led to us monitoring CPU load.

The load monitor is responsible for connecting to service provider's servers and obtains load information. The provision of the load monitoring information is going to be on-demand. Status information is fetched and sent to the load balancing decision maker component, only when it is needed.

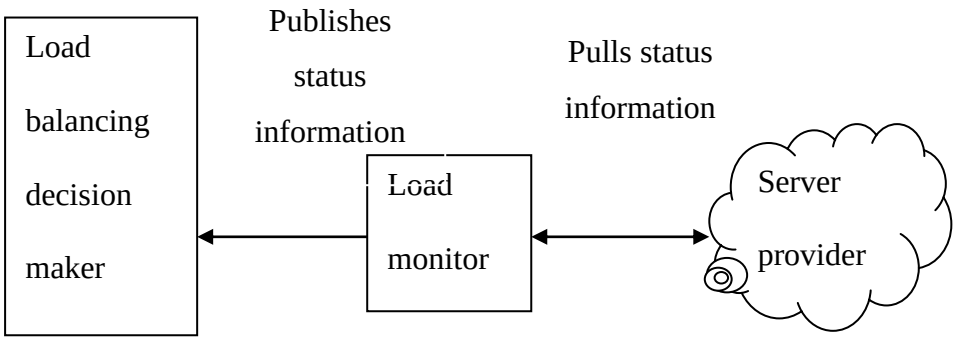


Figure 4.7: The Load Monitor Component.

4.6 Use Case

An SMS broker is a component that functions between the Short Message Service Centre (SMSC) and service providers. The SMS broker acts as a transport or scheduling mechanism. Service clients who are mobile users can invoke services such as news from the service providers. The general operation of the SMS broker is to bring together different parties as shown in Figure 2.2 in Chapter 2. In this Section, we present a usage scenario of the proposed CaLLF:

Consider a service provider who exposes a low-cost service of which service clients are well aware. This results in more regular clients trying to invoke the service at any given time. The SMSCs act as clients to the SMS broker who receives and relays the requests. The SMS broker receives these requests from multiple SMSCs leading to higher chances of the whole system being congested. If there is no efficient mechanism for load balancing some servers are going to be overloaded whilst some are almost idle as the number of requesst being received increases. There must be some special intervention to handle unexpected request increases that drive servers into overloading leading to poor performance.

CaLLF is incorporated into the SMS broker in order to make sure load is evenly distributed among service providers' resources. This avoids imbalances or overloading in the service provider servers which may lead to performance challenges. Given that the SMS broker is a component that propagates or dispatches the requests to service provider servers for processing, it is essential to have a load balancing mechanism in order to avoid overloading on some service provider servers while others are idle. In this case CaLLF serves as a load balancing mechanism which is able to deal with client different service quality or importance.

4.7 Chapter Summary

This Chapter presented the design of CaLLF for an SMS-based service invocation environment. CaLLF is an adaptive and dynamic load balancing solution to the performance problem in SMS-based service invocation environments. It was also proposed that CaLLF should include a priority-based model to enable it to serve client requests with different priority. As stated in Chapter 1, the goal of this work was to come up with a load balancing framework that will be able to function in SMS-based service invocation environments and also serve clients based on their categories. This has been partly achieved in this Chapter with the detailed description of the model design and algorithm. In order to validate that CaLLF achieves what we have claimed, we decided to evaluate its performance and compare CaLLF with the current existing approach. This is presented in Chapter 5.

Chapter 5

Evaluation of the Client-aware Least Loaded Framework

5.1 Introduction

The current SMS based service invocation environments are dogged by performance challenges. These performance challenges emanate from the dynamic nature of the environment coupled with the fact that for people not having phones with access to internet ,SMS-based service invocation environments are preferably a cheaper means for accessing services and this drives service provider servers to be overloaded. In Chapter 4, we presented the Client-aware Least Loaded Framework (CaLLF) as a solution towards dealing with the performance challenges faced by the service provider's infrastructure in SMS-based service invocation environments. This Chapter presents the evaluation of the CaLLF in two ways. First, the scalability of CaLLF is evaluated based on the queuing approach with round-robin (RR) as the load distribution mechanism. Second, the utility of CaLLF is evaluated based on the utility of a non-client aware least loaded algorithm (LLA). Section 5.2 presents the specification of the testbed environment used for the evaluations. Section 5.3 presents how each experiment to evaluate the CaLLF was set up, and the results that were obtained from the experiments. Section 5.4 discusses the findings of this study. Section 5.5, concludes the Chapter.

5.2 Testbed Assumptions

In developing the testbed, the following assumptions were considered due to the duration of this project and considerations of the environment where CaLLF would be used.

1. We assume that the web services exposed by the service provider servers are purely computational services. As a consequence, their execution time is directly proportional to the amount of service requests sent by service clients.
2. We assume that network delay is constant throughout the experimentation.

5.3 Evaluation of Client-aware Least Loaded Framework

For purposes of efficacy and utility we have evaluated the CaLLF to prove that it does achieve what we have claimed in Chapter 4. The evaluation of the CaLLF had two aspects: The first aspects (presented in Section 5.3.1) evaluate the scalability (i.e. load testing) of the least loaded scheme which is employed by CaLLF against RR on the testbed using some simulated client requests. The second aspect (discussed in Section 5.3.2) evaluates the utility of CaLLF against a non-client aware LLA.

5.3.1 Scalability Evaluation of the CaLLF

Scalability is the ability of a system to handle growing amounts of work in a graceful manner or its ability to be enlarged to accommodate that growth. This evaluation is focused on the scalability of CaLLF with increases in the number of requests being processed per unit time. This was investigated in four dimensions; (i) scalability of memory usage, (ii) CPU usage, (iii) throughput of requests and (iv) and response time with increases in the number of requests being processed. The scalability of the CaLLF was benchmarked on the queuing approach with the RR for load distribution. The reason for comparing CaLLF with RR is based on the fact that RR is the most widely used load distribution approach in SMS-based

service invocation environments and is also taken as the de facto standard for load distribution in such environments (Ates, 2012; Aimrudee *et al*, 2010; Balasubramanian *et al*, 2004; Othman *et al*, 2003). The experimental testbed setup, environment and design of the scalability evaluation is discussed in the following subsections.

5.3.1.1 Experimental Setup for Scalability Evaluation of CaLLF

A testbed for the CaLLF was developed. This included implementation and incorporation of CaLLF into the synapse engine (Azeez, 2008; Godage, 2008; Boone *et al*, 2010). Synapse acts as a load scheduler. We chose the synapse engine because it comes with a set of transports, mediators and standard brokering capabilities, such as round-robin, weighted round-robin load balancing algorithms and fail-over. Moreover, the synapse engine can emulate service-oriented environments and can act as an SMS broker for a service oriented environment. The least loaded algorithm (LLA) implemented by CaLLF, distributes clients' requests to appropriate servers based on run time system state information which contributes to load balancing decision making for choosing a server that is least loaded, based on the load metrics monitored. In this work the CPU load index is used.

CaLLF was benchmarked with the queuing approach with RR for load for distribution. The RR scheme distributes load on a rotational basis, based on the assumption that system state information is known in advance and remains constant. Synapse comes with the RR algorithm already implemented, so our experimentation is going to take advantage of this fact. Table 5.1 presents characterisation of CaLLF and RR.

Table 5.1: Characterisation of CaLLF and RR

	CaLLF	RR
Nature	Dynamic	Static
Adaptability	More adaptive	Less adaptive
Centralised	Yes	Yes
Load balancing policy	Least loaded scheme	Round-Robin scheme
Overhead	More	Less
Simplicity	Not simple to implement	Simple to implement

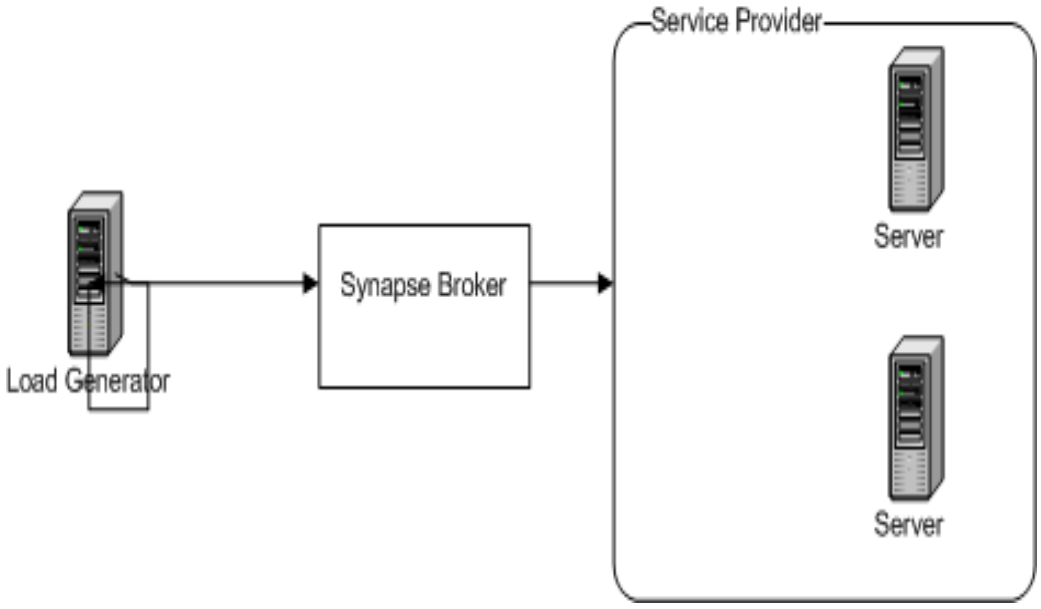


Figure 5.1 Testbed Entities for Experimentation

To mimic the SMS-based service invocation environment, three (3) machines running the Windows 7 Operating System were used, each machine serving its own purpose. The first machine served as a load generator responsible for simulating clients' HTTP requests. The load generator was configured using the Apache HTTP request generator, which creates a number of virtual clients that invoke common web services. For the purpose of load generation, drawing inspiration from Balasubramanian et al, (2004), a fast machine was used

to generate heavy loads imposed on the other two machines which acted as servers. To further avoid potential resource constraints, the synapse engine that acted as a load dispatcher was deployed on the same machine where the load generator was running. The synapse engine was responsible for distributing the requests among the servers using either CaLLF or RR. The machine hosting the load generator and the synapse engine was running an Intel Core i5 3.20 GHz PC, with a RAM 3GB that has Hyper-Threading Technology. The second and third machines were homogeneous servers running Apache Axis for hosting the services to be invoked. The same computationally intensive service was hosted on both machines.

The servers were registered and grouped by the synapse engine for fault tolerant purposes because during experimentation one of them could crash due to unforeseen constraints, so we needed notification when that failover happens. This clustering technique helps the synapse engine to monitor what is happening on available servers ensuring that they respond to request traffic. Both server machines that have Apache Axis were running on Intel Core2Duo 2.94GHz PC with RAM 2GB. The web service that the servers were hosting was a pure computational web service. The web service finds permutations of 5 elements (i.e. given "abcde", the service gives the possible ways that these characters can be ordered). In the experimentation, this permutation web service was invoked by the simulated client requests from the load generator using required parameters. The servers running the permutations web service will compute requests distributed by the synapse engine and then respond accordingly. All nodes were connected via a wired departmental LAN network, which is hardly used. Throughout the whole experimentation process it was made sure that our experiments were the only processes running in the network. This was meant to eliminate traffic and latency that might emanate from the activity in the network.

5.3.1.2 Experimental Design for Scalability Evaluation of CaLLF

The experiment for scalability testing was performed under two scenarios which considered two overloading variants. The first overloading variant involved overloading servers interchangeably where one server was kept overloaded for a certain period until it became free and the other server became overloaded with the process of requests distribution still taking place. The second overloading variant involved overloading one server and keeping the other free throughout the process of sending requests. The inspiration was acquired from scholars like Balasubramanian *et al* (2004). The simulation of overloaded servers was achieved by running multiple computationally intensive computations on the servers, apart from the service that was being invoked through SMSs. The computations used for this purpose were for finding permutations of nine elements. These computations were ran as threads that start and stop interchangeably between the servers.

From the above-mentioned scenarios, response time and throughput were metrics used to measure how each load balancing technique performs. These load balancing techniques involved the least loaded algorithm implemented in CaLLF and round-robin as the benchmarking scheme. Response time is defined as the time taken to send a request and to receive a response. Throughput is defined as the maximum number of requests a load balancing algorithm combined with system can process in a certain period of time. For further analysis, resource oriented metrics were chosen such CPU and memory utilisation. This was done through observing how each load balancing algorithm will utilise and incur cost in such resources when workload increases. CPU utilization is the amount of CPU time taken to send a request and to receive a response. Memory utilisation is the amount of memory consumed between the process of request and response. These metrics are recorded

during the same time for consistency and correlation purposes. The metrics on the client side were recorded as shown in Figure 5.2, Figure 5.3, Figure 5.4 and Figure 5.5.

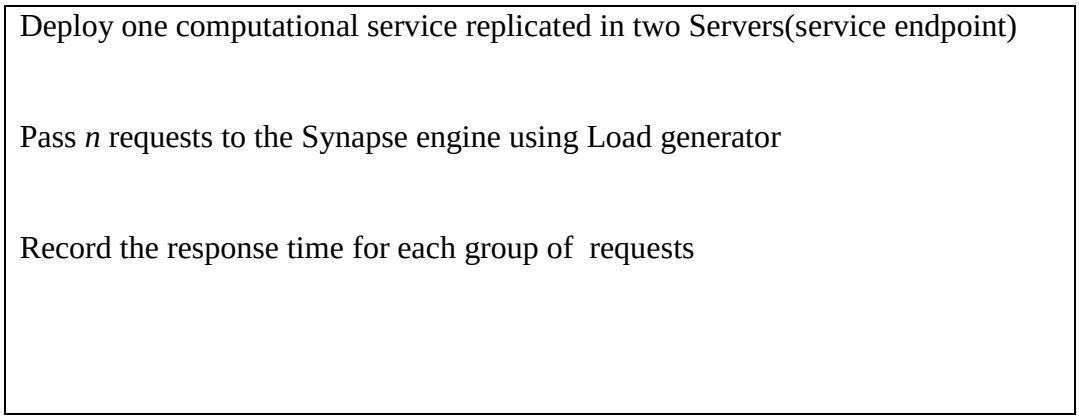


Figure 5.2 Capturing of Response Time of Requests for our CaLLF and RR

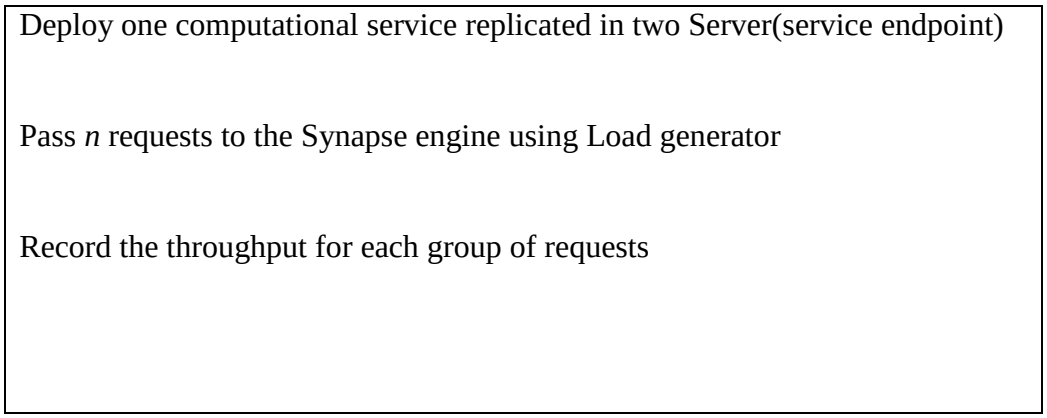


Figure 5.3 Capturing of Throughput for CaLLF and RR.

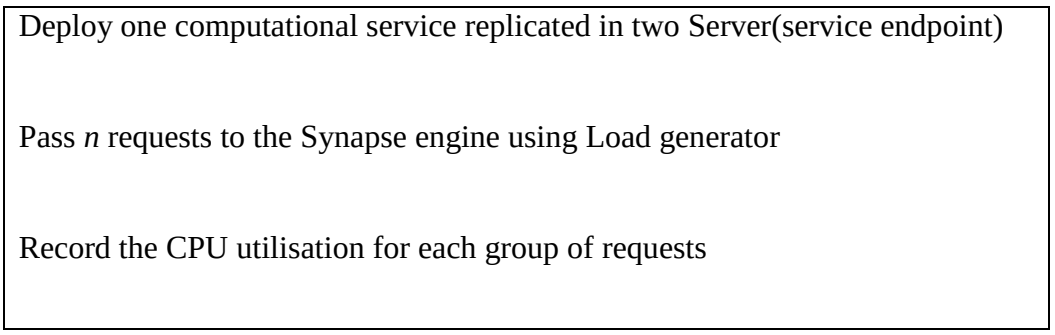


Figure 5.4 Capturing of CPU utilisation forCaLLF and RR

Deploy one computational service replicated in two Server(service endpoint)

Pass *n* requests to the Synapse engine using Load generator

Record the Memory utilisation for each group of request

Figure 5.5 Capturing of Memory Utilisation for CaLLF and RR.

In the process of preparing for the experiments, it was decided to show how the LLA implemented by CaLLF and the RR scheme work. This was achieved through observing how each load scheduling technique distributes requests under situations where servers are overloaded interchangeably. Figure 5.6 shows how each load balancing approach distributes requests given servers statuses (overloaded or underloaded). The figure shows that CaLLF is adaptive because when server 1 was overloaded and server 2 was under loaded, CaLLF decided to distribute requests to the under-loaded server 2. Figure 5.7 shows that RR is not adaptive because when server 1 was overloaded and server 2 was underloaded, the RR distributed requests almost equally to both servers.

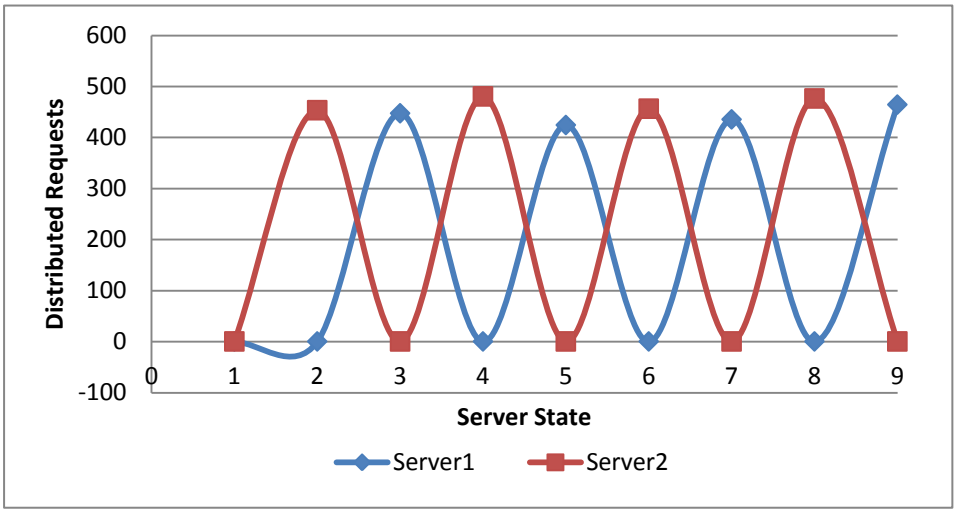


Figure 5.6 Request Distribution of CaLLF with the changing server's status

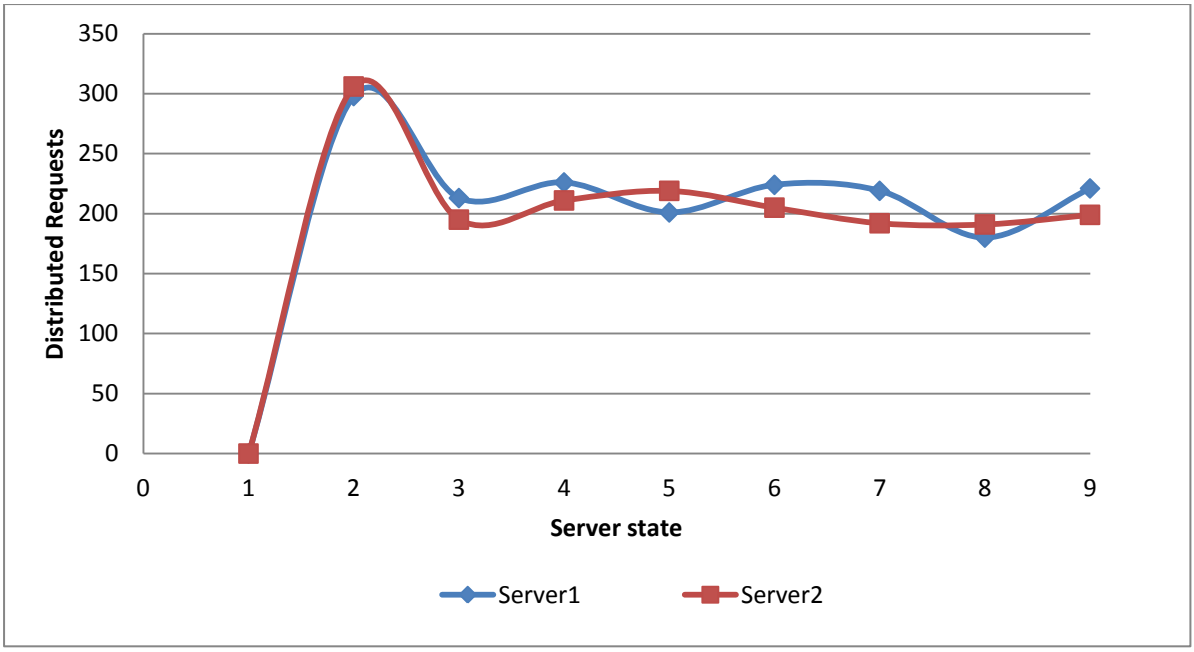


Figure 5.7 Request Distribution of RR with the changing server's status

5.3.1.3 Scalability Evaluation of CaLLF

The purpose of investigating scalability was motivated by the fact that our CaLLF should be able to adapt and scale with increasing numbers of requests, while providing adequate performance on the service provider's infrastructure. CaLLF has more capability and hence promises fair load balancing and better performance over RR which distributes requests on a rotational basis with little or no information about system status. Therefore, it is necessary to investigate whether these capabilities render the CaLLF scalable or not and whether it provides better performance. Weinstock and Goodenough (2006) defined scalability as the ability of a system or product to continue to function well when it is changed in size or volume in order to meet the user need.

This Section describes in detail the two experiments that were conducted under different overloading scenarios, discussed in Section 5.3.1.2, to evaluate the scalability of CaLLF benchmarked on the queuing approach with Round-Robin for load distribution.

Experiment One (Scalability of CaLLF and RR)

This experiment was performed under the first overloading variant, which overloads the servers interchangeably. The design was set to test the Response time, Throughput, CPU and Memory Utilisation when CaLLF (incorporated Synapse engine) is in use for a given number of service requests sent by a load generator or client simulator, invoking web services hosted and replicated in two Apache Axis servers. We started by deploying a permutation web service which is placed in the two servers. We sent 1000 clients' requests to proposed CaLLF hosted by the synapse engine acting as a load dispatcher. Response time, throughput, CPU and memory utilisation were recorded. The same procedure was followed on the RR scheme.

The number of client service requests was increased by 500, starting from the 1000, up until it reached 3000. For each of the two load balancing schemes which were being compared and with each number of request, 10 runs were done and average response time, average throughput, average CPU and memory utilisation were recorded.

Further analysis and discussion of the experimental results is presented in Section 5.4 but in this Section we presented the general view of the results in Figures 5.8, 5.9, 5.10, 5.11 and 5.12. Figure 5.8 shows how the two load balancing techniques (least loaded algorithm implemented in CaLLF and RR) scale with increases in the number of requests with respect to the response time. Figure 5.8 shows that the response time increases linearly as the number of client requests increase for both CaLLF and RR hence they are both scalable but CaLLF provides better response time. Figure 5.9 presents the results of the throughput versus the

number of requests sent. The results in Figure 5.9 show that the throughput of both CaLLF and RR decreases with increases in the number of client requests. In Figure 5.9 we observed that the number of requests sent is inversely proportional to the throughput but CaLLF provides better throughput. In order to do asymptotic analysis on the scalability of the two frameworks being compared, Figure 5.10 presents the graph of the number of requests against the inverse of throughput. Figure 5.10 shows that the inverse of throughput increases as the number of requests increase for both CaLLF and RR. The figure shows that both CaLLF and RR are scalable with increases in the number of requests with respect to throughput.

Figure 5.11 shows that as the number of client requests increases, the CPU utilisation also increases for both the CaLLF and RR approach. The graph in Figure 5.12 shows that as the number of client requests increases, memory utilisation also increases for both the CaLLF and RR approach. Therefore, based on resource utilisation, CaLLF was found to have a larger CPU utilisation compared to the RR scheme as was expected, given the fact that CaLLF supports more capabilities. This is a result of the fact that CaLLF implements adaptive and dynamic load balancing algorithms (Goyal and Patel, 2011); Sharma *et al*, 2008) which is not included in the RR algorithm. Memory utilisation was the same for both the CaLLF and RR approaches as was expected. This is due to the fact that both techniques are distributing requests of the same type.

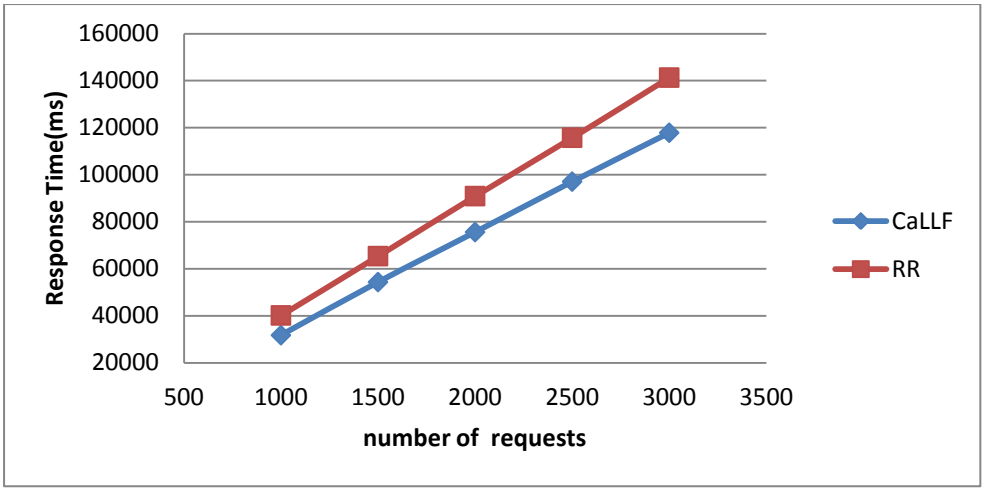


Figure 5.8 Response Time: Scalability of CaLLF and RR with increases in the number of requests

Figure 5.9 Throughput: CaLLF and RR with increases in the number of requests

Figure 5.10 Inverse Throughput: CaLLF and RR with increases in the number of requests

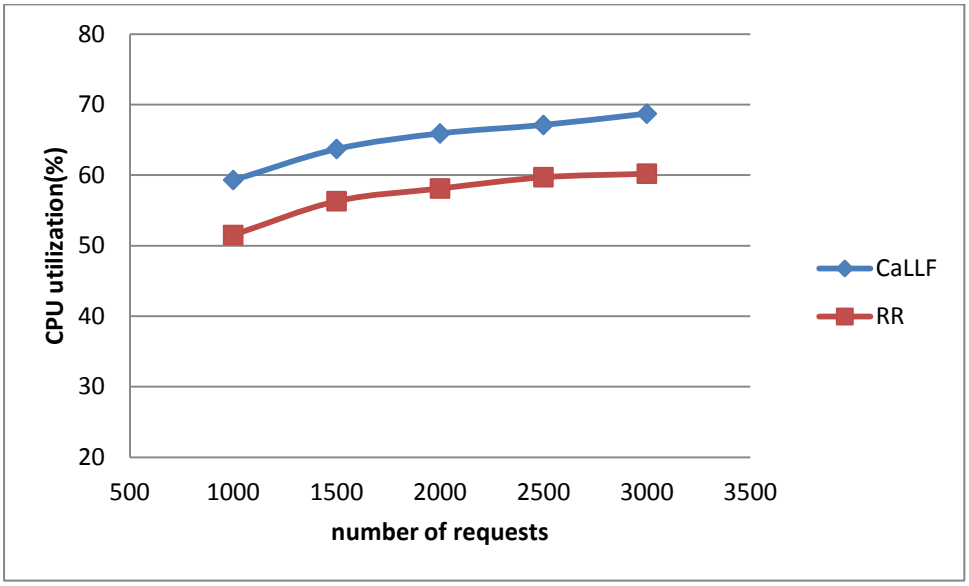


Figure 5.11 CPU Utilisation: Scalability of CaLLF and RR with increases in the number of requests

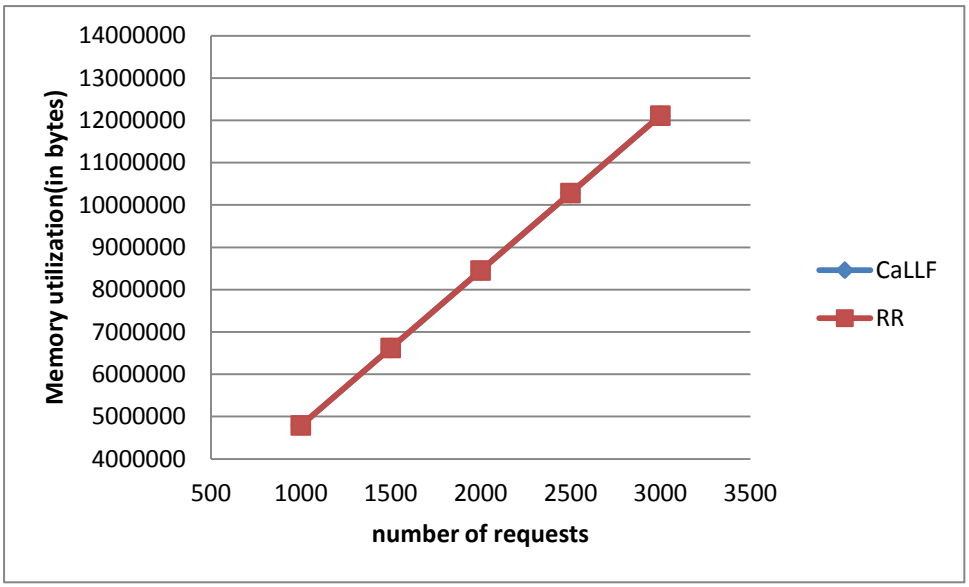


Figure 5.12 Memory Utilisation: Scalability of CaLLF and RR with increases in the number of requests

Experiment Two (Scalability of CaLLF and RR)

In this experiment, the experimental setup is the same as the one presented for Experiment One. The only difference was that the scenario under scalability was evaluated. In this experiment, the second overloading variant which overloads a single server was used. The results from this experiment serve as further proof of the fact that CaLLF is more scalable than RR. The results of this experiment were generally just the same as those from Experiment One, except for differences in the numerical values of the metrics being considered.

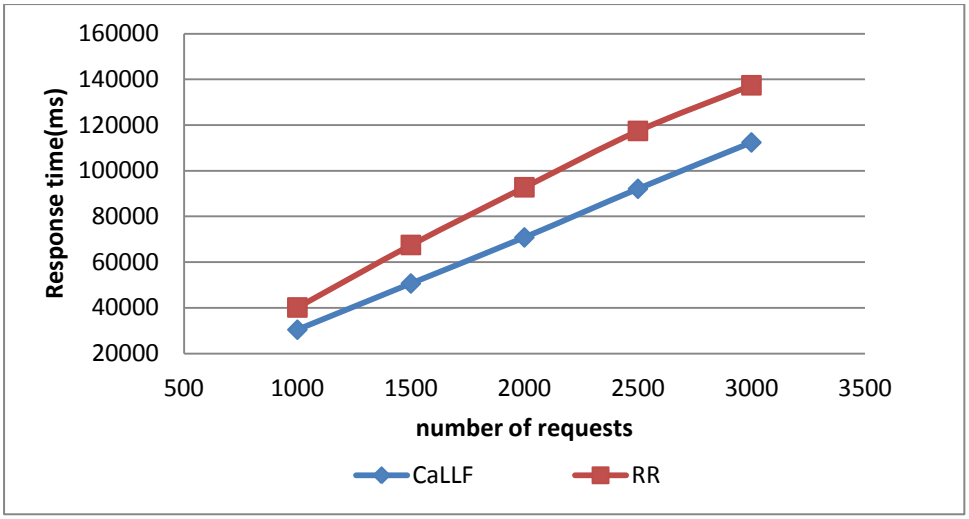


Figure 5.13 Response Time: Scalability of CaLLF and RR with increases in the number of requests

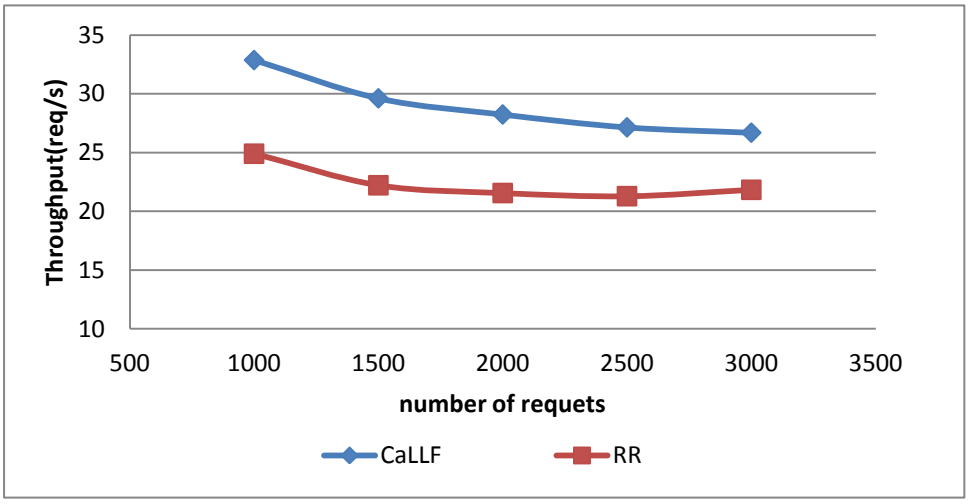


Figure 5.14 Throughput: Scalability of CaLLF and RR scheme with increases in the number of requests

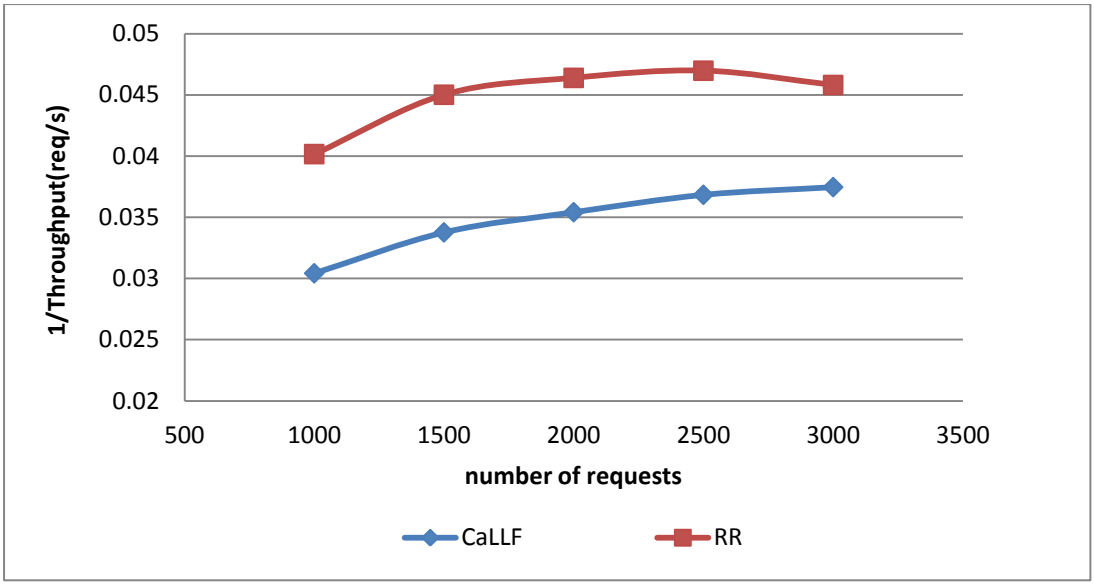


Figure 5.15 Inverse Throughput: Scalability of CaLLF and RR scheme with increases in the number of requests

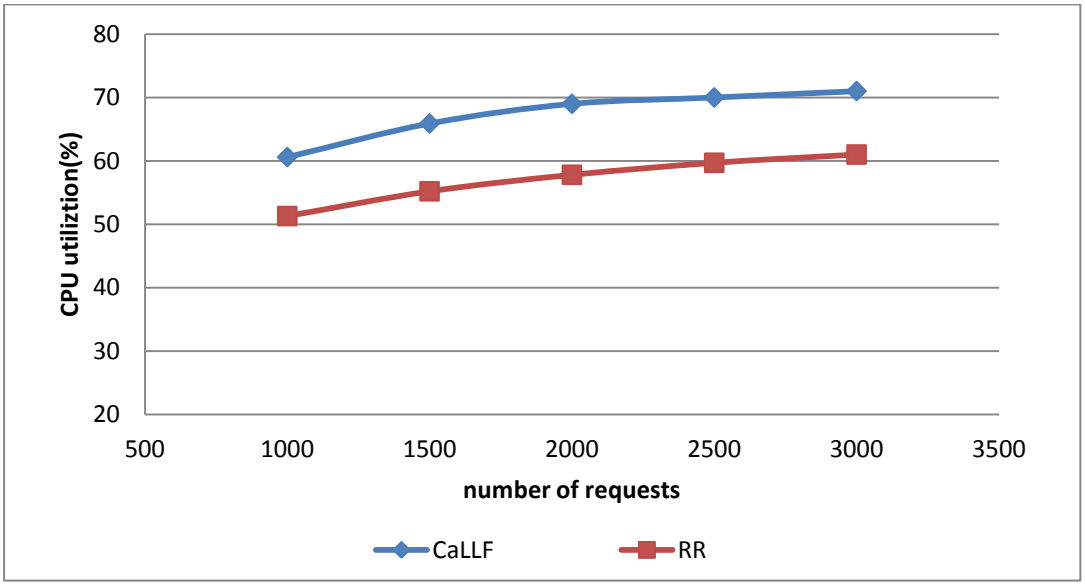


Figure 5.16 CPU Utilisation: Scalability of CaLLF and RR with increases in the number of requests

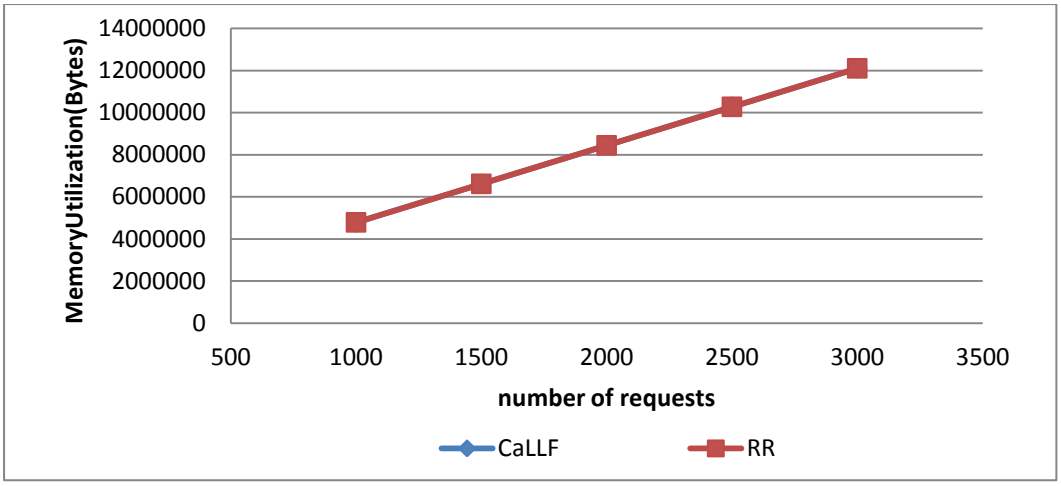


Figure 5.17 Memory Utilisation: Scalability of CaLLF and RR with increases in the number of requests

5.3.2 Evaluation of the Utility of the CaLLF

It is well known that service oriented environments market deals with various types of clients who need different service quality. In our case, these clients are classified as premium clients who need higher service quality and regular clients who just need best effort service. The CaLLF proposed in Chapter 4 combines the client prioritisation mechanisms; over dimensioning of resources and a load balancing algorithm (i.e. the least loaded algorithm). This makes the load balancing approach aware of the different client categories when distributing requests to the servers. In this evaluation, we investigated whether the client prioritisation mechanism introduced in CaLLF improves clients' utility. Utility is achieved when the premium and regular client requests are processed and delivered below or equal to the *Satisfaction threshold*. The following sub-section presents experimental set-up for the experiments being presented in this Section.

5.3.2.1 Experimental Setup for Utility Evaluation of CaLLF

In order to investigate whether client prioritisation improves client utility, CaLLF was compared with the non-client-aware least loaded algorithm (LLA). Both CaLLF and the non-client-aware LLA were incorporated into the synapse engine.

In order to evaluate utility, we used two (2) machines running the Windows 7 Operating System. The first machine was used to simulate clients HTTP requests containing a header called priority that specifies the class of a request. This helps in determining whether a request came from a regular or premium client. In this work, we assumed that clients are already subscribers. The load generator which acts as a client simulator invokes stock quote web service hosted by service provider servers. The load generator was running on a machine that is fast so that load can be imposed on the second machine that serves as a server. The load generator or client's simulator was running on Intel Core i5 3.20 GHz PC with RAM 3GB. The second machine was a server running Apache Axis hosting a simple Qoute informational service of different companies such as IBM and MSFT which are made available through the Apache Synapse engine. The load generator produces HTTP requests with different request priority headers. The headers for premium and regular requests were set to 7 and 4 respectively. This was done to enable the prioritisation process to decide which requests have preference over others based on their priority values. The server machine was running on Intel Core2Dou 2.94GHz PC with RAM 2GB. The server was used to handle different types of web service requests requiring different service quality coming from the load generator. The load generator machine was responsible for generating requests carrying service parameters of the stock quote web service. The request is send through the synapse engine hosting the load distribution algorithms. For the purposes of benchmarking, CaLLF was compared with the non-client aware LLA.

5.3.2.2 Experimental Design for Utility Evaluation of CaLLF

For the purpose of evaluating utility, a service client specifies a *Satisfaction threshold* (T_i), related to some quality of service, T_i. In our case response time was used. The response time threshold required by each class of clients was determined experimentally. Once the service client sends a request to the selected service provider, the response time, X_i, the service provider took to respond to the client is measured. In other words, X_i is a measure ofthe client's quality of experience (QoE) associated with the time it took the service provider to respond to the client's request.

The service client is said to be satisfied if $X_i$is less or equal to (T_i). If this condition is not met, then the service client is said to be dissatisfied.

Pass *n* premium requests to the Synapse engine

Pass *m* regular requests to the Synapse engine

Synapse engine in turn relays Axis server hosting a service

Observe X_i

If $X_i \leq$ *Satisfaction Threshold* (T_i)

then the service client is satisfied

else the service client is dissatisfied

Observe all the X_i that satisfied the service client

Record percentage of satisfied service clients

Figure 5.18: Algorithm for calculating the percentage of satisfied clients

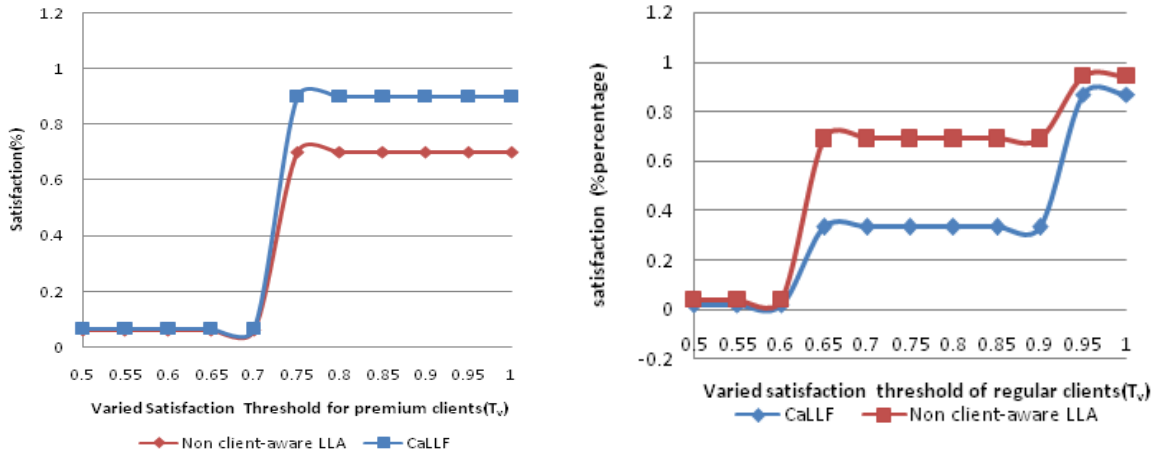
In order to investigate whether client prioritisation improves client utility there was a need to find an optimal satisfaction threshold level for premium and regular clients. In order to find a

suitable satisfaction threshold we experimentally determined some initial satisfaction thresholds. These initial satisfaction thresholds were derived from performing experiments, giving premium and regular clients equal chances of being processed by any server. The initial satisfaction thresholds were found to be 42.70 ms for premium clients ($T_{premium}$) and 50.79 ms for regular clients ($T_{regular}$). These initial satisfaction thresholds were only used as initial values for getting the thresholds to be used in evaluating utility of CaLLF. The goal was to find the satisfaction threshold that maximised the percentage of satisfied clients in each category (regular or premium) of clients.

The satisfaction thresholds were varied by multiplying $T_{premium}$ and $T_{regular}$ with weights (Ws) in the range of 0.5 to 1. The weights were increased by 0.05 intervals from 0.5 up until they got to 1. The varied satisfaction threshold (T_v) was given by $W^* T_{premium}$ for premium clients and $W^*T_{regular}$ for regular clients. Clients were deemed satisfied if the observed response time (X_i) was less than or equal to the threshold response time (W^*T_i). Ten (10) runs were performed for each satisfaction threshold and the average percentage of satisfied clients was recorded for each class of clients. These experiments were run with each run having 100 requests. Four (4) sets of experiments were run with different proportions of regular or premium clients. The proportions were (i) 75 premium clients, (ii) 50 premium clients, (iii) 25 premium clients and (iv) 75% regular clients. The results are shown in Figure 5.19 and Figure 5.20.

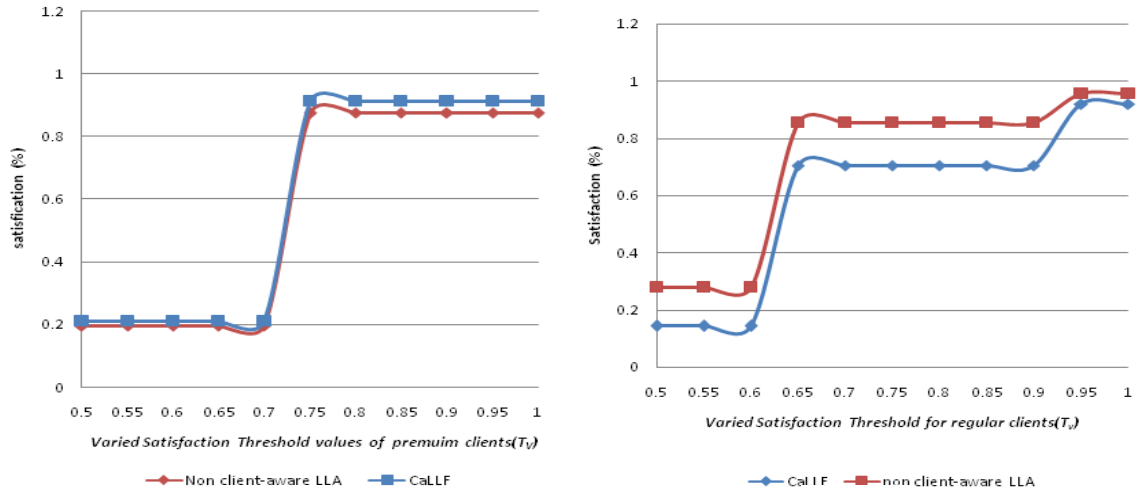
Figure 5.19a and Figure 5.20a shows that optimal satisfaction for premium clients is achieved between $0.7^*T_{premium}$ and $0.75^*T_{premium}$ as shown by drastic increases in the proportion of satisfied premium clients within this range. The drastic increase stops close to $0.75T_{premium}$. This trend was observed for both CaLLF and non–client aware LLA. Figure 5.19b and Figure

5.20b shows that the satisfaction threshold for regular clients is between 0.9 $T_{regular}$ and 0.95 $T_{regular}$.



a b

Figure 5.19: Determining optimal satisfaction threshold for premium users (a) 75% premium clients, (b) 75% regular clients



a b

Figure 5.20: Determining optimal satisfaction threshold for premium users (a) 50% premium clients, (b) 50% regular clients

Drastic increases in the number of satisfied regular clients is observed in two (2) ranges, 0.6 $T_{regular}$ to 0.65$T_{regular}$ and 0.9 $T_{regular}$ to 0.95$T_{regular}$, for both CaLLF and non-client aware LLA. The range with higher weights, 0.9 $T_{regular}$ and 0.95 $T_{regular}$, was used to determine the optimal

satisfaction threshold for regular clients. As a result the satisfaction thresholds 32.025($0.75T_{premium}$) and 48.2505 ($0.95T_{regular}$) were used as satisfaction thresholds for premium and regular clients respectively.

The satisfaction thresholds found in Section 5.3.3.2 were used in determining whether a client is satisfied or not. The experiments were carried out with instances of different proportions of premium and regular clients. Each instance had 100 clients. The 100 client requests were split among premium and regular clients such that the first instance had 90 premium clients and 10 regular clients, followed by the second instance with 80 premium clients and 20 regular clients, etc., up to an instance with 10 premium clients and 90 regular clients. For each of the two (CaLLF and non-client aware LLA) load balancing solutions, 10 runs were run for each instance and the average percentage of satisfied clients was recorded.

5.3.2.3 Experimental Results for Utility Evaluation of CaLLF

This Section presents the results of the comparison of the utility of CaLLF with a non-client aware LLA. Figure 5.23 shows that for CaLLF as the proportion of premium requests increases, the proportion of satisfied clients decreases, while for non-client-aware LLA requests, the proportion of satisfied clients increases, as the proportion of premium requests increases from 10 to 50 per cent premium clients and then decreases as the proportion of premium requests increases from 50 to 90 per cent. It was observed that when the proportion of premium requests is less than 40% CaLLF had better utility compared to non-client aware LLA, but when the proportion of premium users is more that 40% the non-client aware LLA had better utility than CaLLF. This is more likely to be a result of the fact that as the number of premium requests increases CaLLF will end up only serving premium clients at the

expense of regular clients, resulting in the regular clients not being satisfied with the response time they get. We concluded that CaLLF provides better utility for premium clients at the expense of regular clients. In order to further investigate this, Figure 5.22 and Figure 5.23 show the satisfaction level for premium and regular requests in separate graphs. Figure 5.22 shows that CaLLF has better utility than non-client aware LLA for premium clients and Figure 5.23 depicts that non-client aware LLA has better utility than CaLLF for regular clients.

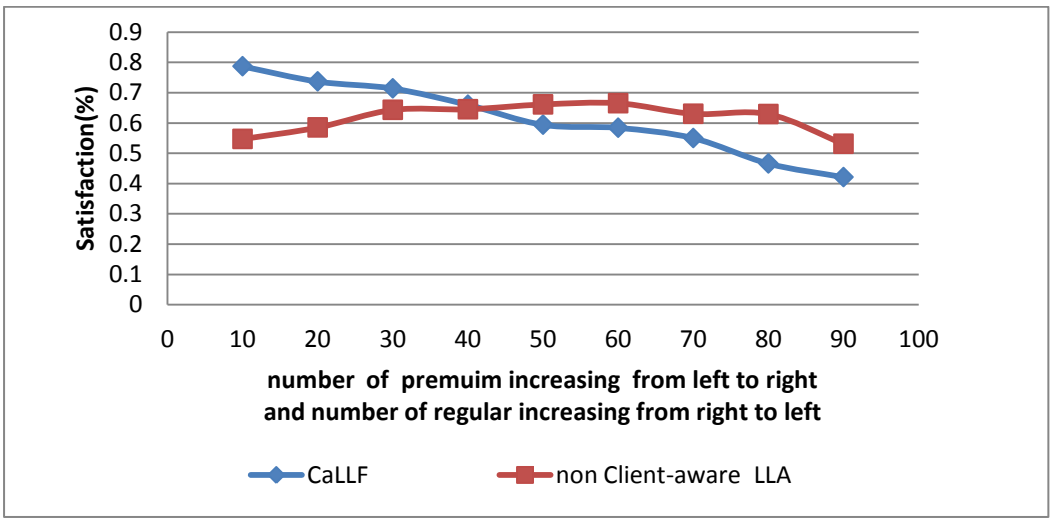


Figure 5.21 Comparison of CaLLF's satisfaction levels for all clients against that of non-client aware LLA

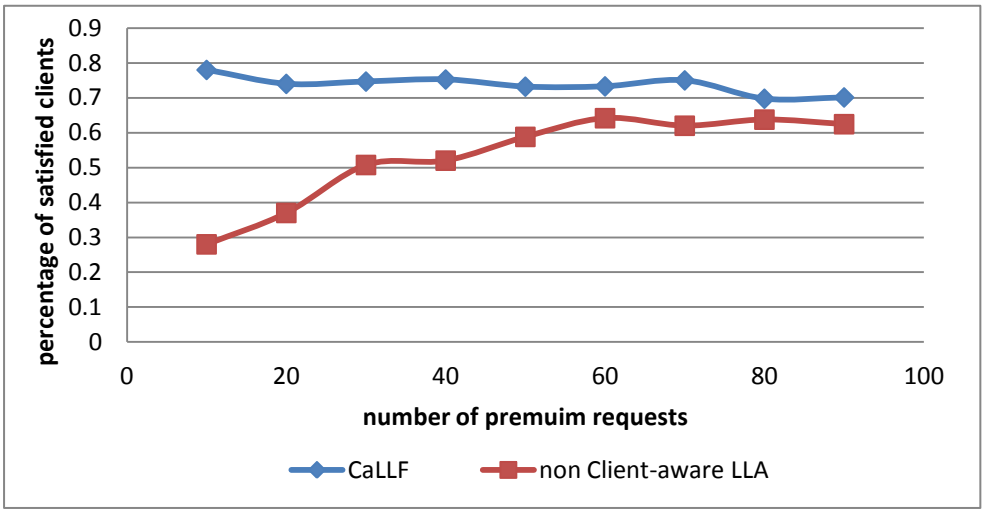


Figure 5.22: Comparison of CaLLF's satisfaction levels for premium clients against that of non-client aware LLA

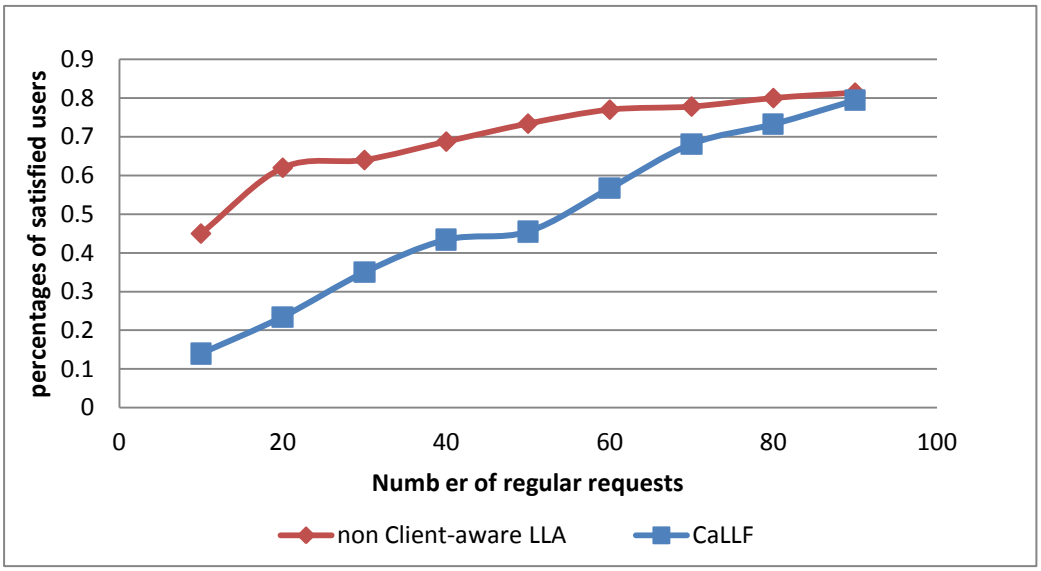


Figure 5.23 Comparison of CaLLF's satisfaction levels for regular clients against that of non-client aware LLA

5.4 Discussion of Results

The experimental results on scalability in Section 5.3.2 proved one of the expected impacts of CaLLF which is providing a load balancing approach that can cope with unexpected traffic in environments such as SMS-based service invocation. These scalability experimental results showed that both the CaLLF and RR scheme are scalable which is needed as Chandra (2002) and Arape *et al* (2003) stated that a successful load balancing approach should improve scalability. However, even though CaLLF and RR are scalable, CaLLF had better response time implying that CaLLF has better system performance as Goyal and Patel (2011) stated that desirable properties of the load balancing approach should be to improve or optimise overall system performance which usually is in relation to minimising response time. Moreover, a load balancing approach should perform well even under strenuous conditions. The scalability experiments carried out revealed that CaLLF performs relatively well even under strenuous conditions. The CaLLF also showed better throughput against RR and according to Chhabbra *et al* (2006), the goal of the load balancing is to maximise throughput.

However, even though CaLLF performed better than RR in terms of response time and throughput, it was found that CaLLF utilises more CPU and memory resources. This is due to the fact that CaLLF implements an adaptive and dynamic load balancing approach. Chhabbra *et al* (2006) and Goyal and Patel (2011) stated that adaptive and dynamic load balancing approaches incur overheads which emanate from their use of runtime system status to make decisions on the distribution of load.

The experimental results on utility in Section 5.3.2 proved that the CaLLF is able to serve clients according to their category i.e. premium and regular clients, while providing good utility to clients requiring higher service quality. This was the second expected impact of CaLLF. CaLLF was shown to be client aware because it provided better utility for premium

clients and less utility for regular clients, which required best effort service. CaLLF was compared to non-client-aware LLA and CaLLF was found to provide better utility to premium clients. As Texeira *et al* (2004) stated, if a service provider deals with different types of client with different importance it is essential to bring in differentiation or prioritisation. From the above however, non-client aware LLA provided more satisfaction for regular clients compared to CaLLF. This was due to the fact that CaLLF implements strict prioritisations which means that regular clients are served only when there are no premium clients (Boone *et al*, 2010). Thus CaLLF keeps premium clients satisfied at the expense of regular users.

5.5 Chapter Summary

This Chapter presented results of the evaluation of CaLLF. Experiments for evaluating the scalability and utility of CaLLF were conducted. The goal was to evaluate whether CaLLF was able to adapt to the dynamic nature of SMS based service invocation environments and also achieve increased client utility/satisfaction. With respect to scalability, CaLLF was found to be scalable and performed better than the RR scheme in terms of response time and throughput. With respect to utility CaLLF was found to have better utility than non-client aware LLA.

Chapter 6

Conclusion and Future Work

6.1 Introduction

This work addressed the performance challenge of infrastructure in an SMS based service invocation environment. The dynamic nature of the environment and the high capacity demand occasioned by the high usage preference for SMS as a cheaper way of utilising services, creates a high demand on the infrastructure. This performance challenge occurs when many service consumers (mobile users) take advantage of the low cost of SMS based service invocation. This, coupled with the dynamics of the service oriented environment and autonomy of the entities involved, results in lots of uncertain behaviour. Therefore, one of these entities, such as service consumers, is bound to generate burst loads of requests for the service provider's servers. The servers can be overloaded hence leading to a performance degradation in the service provider's infrastructure if there is no special precaution taken for the issue mentioned above. Thus a system that can handle unexpected traffic or sudden load peaks in a scalable manner while maintaining performance requirements was proposed.

The overall goal of this research was to come up with a load balancing framework for SMS based service invocation environments and to serve clients based on their categories. This framework is referred to as a Client-aware Least Loaded Framework (CaLLF). The proposed CaLLF, modelled, implemented and evaluated, enabled load balancing decisions and distribution based on system runtime status information. This was to ensure that CaLLF is able to handle large loads in a scalable manner while maintaining adequate performance in dynamic environments as shown in Chapter 5. Moreover, based on the current status of the service oriented market which may have different clients requiring different service quality, CaLLF was also designed to serve clients by their class of importance. This was achieved by

combining a load balancing algorithm with a prioritisation mechanism which formed the modular components of CaLLF. A testbed of CaLLF was then carried out for scalability purposes against the de facto standard which is RR (Ates, 2012; Jongtaveesataporn *et al*, 2010; Balasubramanian *et al*, 2004 and Othman *et al*, 2003) as discussed in Section 5.3.2. Moreover, CaLLF was also evaluated for utility in order to observe the satisfaction level that it can provide for premium client requests requiring high service quality as presented in Section 5.3.4.

The remainder of this chapter is organised as follows: Section 6.2 highlights the achievements of this research. Section 6.3 discusses the limitations of this work and gives some suggestions on how the model can be extended in future.

6.2 Conclusion

Services are created and made available to be consumed via a service interface chosen by a service provider to render their services. The popularity of SMS has led service providers to use it as a service interface because it allows service provider services to reach a massive market. SMS based service invocation is enabled to reach larger audiences (service consumers). It is also enabled in dynamic environments such as a service-oriented environment which is unpredictable. These scenarios may cause an unexpected traffic load from service consumers. This makes the system have to deal with large number of requests which causes performance challenges to system infrastructure. To have a solution for such environments there was a need for an adaptive and dynamic load balancing approach that tries to address performance problems. The least loaded approach (Balasubramanian *et al*, 2004, Aimrudee *et al*, 2010 and Othman *et al*, 2003) was chosen to arrive at a solution. The least loaded approach identifies the building blocks required for handling load in a scalable manner. The approach is dynamic; it sends/distributes requests to servers in a cluster, based

on which server currently has the lowest load. The approach is made up of a transfer policy which makes load balancing decisions and distribution based on information collected about system status. The collection of information about the system status is done by a load information policy which is one building block of the least loaded approach. These building blocks formed a part of CaLLF which fell under the load balancing decision maker and load monitor components. It was perceived necessary to make the load balancing approach aware of different types of clients that participate in the service oriented market for fairness purposes since service orientated environment deal with various types of clients. Against this background of the over dimensioning of resources, classification and prioritising mechanisms were introduced to seal-up CaLLF. The proposed CaLLF consists of a load balancing decision maker, load monitor and client classifier components. These components are proved to be important in an SMS based service invocation environment in terms of handling load in scalability while providing adequate performance. Moreover, adding and combining these components to CaLLF makes it unique from other load balancing approaches.

After implementing CaLLF, there was a need to validate its performance expectation. The proposed CaLLF and RR were first evaluated in experimentation testbed configurations. The proposed CaLLF was compared to RR which acts as a benchmark. This testbed was used to conduct performance evaluation experiments between the two approaches. As mentioned in section 1.4, this work is driven by first finding a load balancing approach that addresses the issue of unexpected traffic and sudden load peaks. While this is the main theme, it was necessary to include a client aware mechanism. The second evaluation was meant to test client awareness of CaLLF compared with the non-client aware least loaded algorithm (LLA) which is the same load balancing approach used in CaLLF but without prioritisation. This was done in order to observe the satisfaction level of CaLLF over non client-aware LLA. In the testbed experimentation, the following evaluation parameters were used:

1. Scalability

2. Utility

Analysis of the results showed that both CaLLF and RR schemes are scalable but CaLLF had better performance in terms of response time and throughput over the RR scheme. Goyal and Patel (2011) stated that the dynamic load balancing algorithms, one of which was implemented by CaLLF, are robust and flexible for unpredictable environments such as distributed systems. Therefore, the experiments showed that CaLLF has improved performance i.e. handling an increasing amount of load in the system without degrading performance. The results obtained from the experimental testbed also revealed that CaLLF brings more satisfaction to a consumer who requires high service quality (i.e. premium clients) as compared to the non-client aware least loaded algorithm (LLA).

In essence, the evaluations conducted show that CaLLF as proposed in this research is able to handle increasing load in a scalable manner while providing adequate performance. It is also able to serve clients according to their category, either as regular or premium clients.

6.3 Limitations and Future Work

Although the proposed CaLLF has been proved to be a more effective approach in dynamic environments such as SMS based service invocation in a service oriented environment, it has some limitations calling for future enhancements. The developed CaLLF has a higher CPU utilisation in comparison with the RR scheme. This may be due to the number of capabilities that the CaLLF has. Even though this high CPU utilisation was expected, in future, CaLLF will be fine-tuned to achieve results same as the one observed in this study (in terms of reponse time and throughput) with minimal CPU utilisation. The other limitation is that experiments with CaLLF only considered homogeneous servers. Future enhancements of

CaLLF would be tested against the RR scheme in heterogeneous environments to compare performance. Moreover, testing of the over dimension of resources approach aspect of CaLLF was left for future work. This needs to be tested for the purpose of seeing what value it will add to utility. Lastly, an approach for attaining satisfaction thresholds for premium and regular clients should be addressed, probably by training or some other approach, in future work.

References

Adigun, M.O, Emuoyibofarhe O. J., and Migiro, S.O.(2006). "Challenges to Access and Opportunity to use SMME enabling Technologies in Africa", *a presentation at 1st All Africa Technology Diffusion Conference*, June 12-14, Johannesburg South Africa.

Aimrudee Jongtaveesataporn, Shingo Takada. (2010). "Enhancing enterprise service bus capability for load balancing". *Journal WSEAS Transactions on Computers archive*, Volume 9 Issue 3, March 2010.

Al-ali, R. J., Rana, O.F., Walker, D.W., Jha, S., and Sohail, S. (2002)."G-QoSM: Grid service discovery using QoS properties," *Computing and Informatics Journal*, Special Issue on Grid Computing, vol. 21.

Al-Raqabani, R.A., Barada, H., & Benlamri, R. (2005). "Performance of probing and coordinated load sharing", in:*Proceedings of 17th IASTED Int. Conf. on Parallel and Distributed Computing and Systems*, pg 66--71. Phoenix, Arizona, USA.

Arapé, N., Colmenares, J. A., and Queipo, N. V. (2003). On the Development of an Enhanced Least Loaded Strategy for the CORBA Load Balancing and Monitoring Service, *in Seong-Moo Yoo & Hee Yong Youn, ed., 'ISCA PDCS'*, pp. 205–211.

Azeez, A. (2008). "Auto-scaling Web services on Amazon EC2". Available at http://people.apache.org/_azeez/autoscaling-web-services-azeez.pdf [Accessed:23-April-2012].

B'Far, R. (2005). "Mobile Computing Principles: Designing And Developing Mobile Applications with UML And XML". Cambridge University Press.

Balasubramanian, J., Schmidt, D. C., Dowdy, L. W., Othman, O. (2004). "Evaluating the Performance of Middleware Load Balancing Strategies", in:*Proceedings of 8th Intl.. Conf. on Enterprise Distributed Object Computing*, pp. 135-146.

Bernardo, L., Pinto, P. (1998). "Scalable Service Deployment on Highly Populated Networks". *In: Intelligent Agents to Telecommunication Applications - Proceedings Second International Workshop IATA'98*, Paris, Springer-Verlag LNCS Vol. 1437.

Bic, L. F., and Shaw, A. C. (2002). "Operating Systems Principles", 1st ed. Prentice Hall.

Bonald, T., and Roberts, J. (2001). "Performance modeling of elastic traffic in overload," *SIGMETRICS Perform. Eval. Rev*., vol. 29, no. 1, pp. 342–343, Jun. 2001

Boone, B., Van Hoecke, S. and Van SeghBroek, G. (2010). "SALSA: QoS-aware load balancing for service brokering". *Journal of Systems and Software*. 83, pp. 446–456

Bourke, T. (2001). *Server Load Balancing* (1st ed.). O'Reilly Media.

Brown, J., Shipman, B., and Vetter, R. (2008). SMS: The Short Message Service. Computer 40, 12 (Dec. 2007), 106-110.

Bryhni, H., Klovning, E., Kure, O. (2000). "A comparison of load balancing techniques for scalable web servers*". IEEE Netw2000; July/August*: pp 58–64.

Buthelezi, M., Adigun, M.O., Ekabua, O., Iyilade, J. "Accounting, Pricing and Charging Service Models for a GUISET Grid-Based Service Provisioning Environment". *CSREA EEE 2008*: 350-355

Cardellini , V., Casalicchio, E., Colojanni, M., AND Yu, P. (2002). "The state of the art in locally distributed web-server systems". *ACM Comput. Surv*. 34, 2 June, 263–311.

Cherkasova, L., and Phaal, P. (2002). "Session Based Admission Control: a Mechanism for Peak Load Management of Commercial Web Sites". *IEEE J. Transactions on Computers*, Vol. 51, No. 6.

Cheung, A. K and Jacobsen, H. (2006). "Dynamic Load balancing in Distributed Content-Based Publish/Subscribe". in: *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, Middleware '06*. Springer-Verlag New York, Inc., New York, NY, USA, pp. 141–161

Chhabra A. Chhabra, G. Singh. (2006). "Qualitative Parametric Comparison of Load Balancing Algorithms in Distributed Computing Environment", *14th International Conference on Advanced Computing and Communication*, July 2006 IEEE, pp 58 – 61.

Christodoulopoulos, K., Varvarigos, M., Develder, C., De Leenheer, M., Dhoedt, B. (2007). "Job demand models for optical grid research". *In: Proceedings of the 11th Conference on Optical Network Design and Modelling (ONDM),* Athens, Greece.

Cortes A. Cortes, Ripoll, A., Senar, M., and E. Luque. (1999). "Performance Comparison of Dynamic Load-Balancing Strategies for Distributed Computing," in:*Proceedings of 32nd Hawaii Conf. System Sciences*, vol. 8, pp. 8041.

Doulkeridis, C., and Vazirgiannis, M. (2004). "Querying and Updating a Context-Aware Service Directory in Mobile Environments". *Web Intelligence* : 562-565.

Doulkeridis, C., Loutas N. and Vazirgiannis M. (2006). "A System Architecture for Context-Aware Service Discovery". *in: Electronic Notes in Theoretical Computer Science 146*, pp. 101-106.

Foster, I., Roy, A., Sander, V., and Gmbh, F.J. (2000). "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation," *in: 2000 Eighth International Workshop on Quality of Service, 2000. IWQOS. Presented at the 2000 Eighth International Workshop on Quality of Service, 2000. IWQOS*, pp. 181–188.

Gondhi, N.K., and Pant, D. (2009). "An Evolutionary approach for scalable Load Balancing in Cluster Computing", *IEEE International Advance Computing Conference*.

Goyal S. K.G., Patel. R.B. (2011). "Adaptive and Dynamic Load Balancing Methodologies for Distributed Environment", *International Journal of Engineering Science and Technology* (IJEST), 3(3), 1835 - 1840.

Grosu, D., Chronopoulos, A. T., and Leung, M. Y. (2002). "Load balancing in distributed systems: An approach using cooperative games," in:*Proceedings of IPDPS*, pp. 52–61.

Guo, H. (2003). "A Bayesian approach for autonomic algorithm selection IJCAI workshop on AI and autonomic computing: developing a research agenda for self-managing computer systems, Acapulco, Mexico, August; 10.

Hewlett Packard (HP), "DreamWorks Animation Gets 'Extreme' with HP Scale-out Storage." [Online]. Available: http://www.hp.com/hpinfo/newsroom/press/2009/090615xa.html [Accessed: 22-Jul-2012].

Jamil Md. S. and Mousumi, F.A. (2008). "Short Messaging Service (SMS) based m-Banking System in Context of Bangladesh," *11th International Conference on Computer and Information Technology*, KUET, Bangladesh, pp 599-604, December 25-27, 2008.

Karasum E. Karasam, E. Ayanoglu. (1998). Effects of wavelength routing and selection algorithms on wavelength conversion gain in WDM optical networks, *IEEE/ACM Transactions on Networking 6 (2)* ,pp.186-196..

Kopparapu, C. (2002). "Load balancing servers, firewalls, and caches". *Wiley Computer Publishing*, John Wiley & Sons.

Kunz, T. (1991). "The influence of different workload descriptions on a heuristic load balancing scheme," *Software Engineering*, *IEEE Transactions on*, vol. 17, no. 7, pp. 725 – 730, Jul. 1991.

Li, W., and Shi, H. (2009). "Dynamic Load Balancing Algorithm Based on FCFS," in 2009 *Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pp. 1528 –1531.

Lin, M.T .N.G., Silva, T.P.C., dos Santos, R.O., and da Silva Neto, A.F. (2009). "SMBots - An architecture to manage dynamic services based on SMS", *in:Proceedings of 2010 11th International Conference on Mobile Data Management (MDM).* pp. 311–313..

Microsoft "Load Balancing - Project Server 2003 IT Documentation - Office.com." [Online]. Available: http://office.microsoft.com/en-us/project-server-it/load-balancing-HA001165275.aspx [Accessed: 22-Jul-2012].

Motwani, J., and Raghavan, P. (1995). "Randomized Algorithm". Cambridge University Press.

Mouratidis, K., Yiu, M. L., Papadias, D., & Mamoulis, N. (2006). Continuous Nearest Neighbor Monitoring in Road Networks. In:*Proceedings of the Very Large Data Bases Conference (VLDB)*, (pp. 43-54). Seoul, Korea.

Mühl, G., Fiege, L., and Pietzuch P. (2006). "Distributed event-based systems". Springer-Verlag. Berlin

Muppala, S., and Zhou, X. (2011). "Coordinated session-based admission control with statistical learning for multi-tier internet applications," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 20–29.

Nandagopal, M., and Uthariaraj, R. (2010) "Sender Initiated Decentralized Dynamic Load Balancing for Multi Cluster Computational Grid Environment", in: *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, Tamilnadu, India, pp. 149-160.

Nikravan, M., and Kashani, M.H. (2007)."A Genetic algorithm for process scheduling in distributed operating systems considering load balancing", in:*Proceedings 21st European Conference on Modelling and Simulation Ivan Zelinka, Zuzana Oplatková*, ISBN 978-0-9553018-2-7

Nurmi, D., Wolski, R., Brevik, J., 2009. "Probabilistic Reservation Services for Large-Scale Batch-Scheduled Systems". *IEEE Syst. J. 3*, 6–24

Othman, O., Balasubramanian, J., and Schmidt, D C. (2003). "The Design of an Adaptive Middleware Load Balancing and Monitoring Service". *In LNCS/LNAI: Proceedings of the Third International Workshop on Self-Adaptive Software*, Heidelberg, Springer-Verlag.

Papazoglou, M.P. and W.J. van den Heuvel (2005): Service-oriented architectures,*VLDB Journal 16*, July 2005.

Papazoglou, M.P., Traverso, P., Dustdar, S. and Leymann, F. (2007). "Service-Oriented Computing: State of the Art and Research Challenges", *Computer Volume 40, Issue 11*, Nov. 2007 Page(s): 38 – 45.

Papazoglou M.P.(2007).Web Services: Principles and Technology, Prentice Hall.

Parent, J., Verbeeck, K., and Lemeire, J. (2002). Adaptive load balancing of parallel applications with reinforcement learning on heterogeneous networks. In: *Proceedings of international symposium DCABES* 12, Amsterdam, The Netherlands, pp.71–79.

Petrovic, S. and Fayad, C. (2005). "A Genetic Algorithm for Job shop Scheduling with Load Balancing", in AI 2005, *Lecture Notes in Artificial Intelligence 3809 edited by S.Zhang and R.Jarvis*, pp. 339-348. (Springer)

Qin, X.Q., Jiang, H., Zhu, Y., and Swanson, D.R.(2003). "Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters," *in the Proceedings of the 10th International Conference on High Performance Computing (HiPC 2003)*, December 17-20, Hyderabad, India.

Rahmawan, H., and Gondokaryono,Y.S. (2009). The Simulation of Static Load Balancing Algorithms, Electrical Engineering .2009

Ramana, K. K., and Jataayu, M.V.G. (2005). "Load balancing of services with server initiated connections", in: *Proceedings of the 2005 IEEE International Conference on Personal Wireless Communications (ICPWC)*, Delhi, pp. 254 – 257.

Rendón, O.M.C., Pabón, F.O.M., Vargas, M.J.G., and Guaca, J.A.H. (2005). "Architectures for Web Services Access from Mobile Devices*",.in: Proceedings of the Web Congress, 2005. LA-WEB 2005*, Buenos Aires, Argentina.

Risi, D., and Teófilo, M. (2009). MobileDeck: turning SMS into a rich user experience. *In: Proceedings of the 6th international Conference on Mobile Technology, Application & Systems (Nice, France,September 02 - 04, 2009)*. Mobility '09. ACM, New York, NY, pp. 1-4.

Romtveit, T. (2010). "Load-balancing by applying a bayesian learning automata (BLA) scheme in a non-stationary web-crawler network," pp. 130.

Rosas, F. J. L., and Romo, J. C. M. (2007). "Improving Dynamic Load Balancing Under CORBA with a Genetic Strategy in a Neural System of Off-line Signature Verification," *presented at the PDPTA*, 2007, pp. 510–516.

Rotaru, T., and Nägeli, H.H.(2004). "Dynamic Load Balancing in Heterogeneous Systems by Diffusion", *Journal of Parallel and Distributed Computing* 64(4): 481-497 .

Roth, G. "Server load balancing architectures, Part 2: Application-level load balancing," JavaWorld, 21-Oct-2008. [Online]. Available: http://www.javaworld.com/javaworld/jw-10-2008/jw-10-load-balancing-2.html [Accessed: 22-Jul-2012].

Shadrach, D. C., Balagani, K. S., and Phoha V. V. (2009) "A Weighted Metric Based Adaptive Algorithm for Web Server Load Balancing," *in: Proceedings of the 3rd International Symposium on Intelligent Information Technology Application(IITA), vol. 1*, pp. 449 –452.

Shan G. Shen, S. K. Bose, T. H. Cheng, C. Lu, and T. K. Chai. (2000). "Efficient wavelength assignments for light paths in WDM optical networks with/without wavelength conversion," *Photon. Netw.Commun. 2*, pp.349–360.

Shan, Z., Lin, C., Marinescu, D.C., and Yang, Y. (2002)."Modeling and performance analysis of QoS-aware load balancing of web-server clusters". *Computer Networks* 40 2, pp. 235–256

Sharma, S., Singh, S., and Sharma, M. (2008). "Performance Analysis of Load Balancing Algorithms", World Academy of Science, *Engineering and Technology*.

Shirazi, B.A., Hurson , A.R. and Kavi, K.M. (1995). Editors, "Scheduling and Load-Balancing in Parallel and Distributed Systems", *IEE CS Press* .

Sleeper, B., and Robins, Bill. (2001). "Defining Web Services". The Stencil Group.

Smith, W., Foster, I., and Taylor, V. (2000). Scheduling with Advanced Reservations. *In: Proceedings of the IPDPS Conference, Cancun, Mexico*. IEEE Computer Society Press: Los Alamitos, CA, 127

.

Teixeira, M., M. Santana and R. Santana, (2004). Using Adaptive Priority Scheduling for Service Differentiation in QoS-aware Web Servers. In: *Proceedings of the 23rd IEEE International Conference on Performance Computing and Communications*, Arizona, April 15-17, pp: 279-285.

Toufik Taibi, Abdelouahab Abid and Engku Fariez Engku Azahan.(2007) .“A Comparison of Dynamic Load Balancing Algorithms”, *Jordan Journal of Applied Sciences*, Vol. 9, No 2.

Vanrompay, Y., Kirsch-Pinheiro, M. and Berbers, Y. (2009). Context-Aware Service Selection with Uncertain Context Information. In: *Proceedings of the 2nd International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services* (CAMPUS 2009).

Wan, B. (2008). “Business-Based SMS Mobile Search”. In: *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications* – Workshops, pp. 692-695.

Wang, J. Chen, J., Wang, Y., and Zheng, D. (2009). “Intelligent Load Balancing Strategies for Complex Distributed Simulation Applications,”.in: *Proceedings of the International Conference on Computational Intelligence and Security, 2009. CIS ’09*., pp. 182–186..

Weerawarana, S., Curbera, F., Leymann, F., Storey, T., and Ferguson, D.F. (2005). Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, 1st ed. Prentice Hall.

Weinstock, C. B and Goodenough, J.B. (2006).“*On System Scalability*,” Technical Note,

CMU/SEI-2006-TN-012.

Werstein, P., Situ, H., and Huang, Z. (2006). “Load Balancing in a Cluster Computer”, in: *Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Taiwan, pp. 569-577.

Xiao, L., Chen, S., and Zhang, X. (2002). “Dynamic cluster resource allocations for jobs with known and unknown memory demands,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 223 –240.

Yan, C., Zhu, M., and Shi, Y. (2008). "A Response Time based Load Balancing Algorithm for Service Composition," *in Proceedings of the Pervasive Computing and Applications. ICPCA 2008*., vol. 1, pp. 13 –16.

Yigitbasi, N., and Epema, D. (2010). "Overdimensioning for Consistent Performance in Grids". *In Proceedings of the 10th IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGRID10)*, IEEE Computer Society, Melbourne, Australia, pp. 526-529.

Zhang, X., Qu, Y., and Xiao, L. (2000). "Improving Distributed Workload Performance by Sharing Both CPU and Memory Resources".*in: Proceedings of the 20th International Conference on Distributed Computing Systems, 2000.*, pp. 233–241

Zhang, Y., Harrison, P.O. (2007). "Performance of a priority-weighted round-robin mechanism for differentiated service networks". *in: Proceedings of the 16th International Conference on Computer Communications and Networks, 2007. ICCCN 2007*, pp. 1198–1203.