AUTOMATED DYNAMIC SERVICE PLACEMENT AND REPLICATION FRAMEWORK USING TEAM FORMATION APPROACH TO ENHANCE SERVICE AVAILABILITY

OOI BOON YAIK

UNIVERSITI SAINS MALAYSIA

2012

AUTOMATED DYNAMIC SERVICE PLACEMENT AND REPLICATION FRAMEWORK USING TEAM FORMATION APPROACH TO ENHANCE SERVICE AVAILABILITY

by

OOI BOON YAIK

Thesis submitted in fulfillment of the requirements
for the degree of
Doctor of Philosophy

May 2012

ACKNOWLEDGEMENTS

First and foremost I offer my deepest gratitude to my supervisor, Dr. Chan Huah Yong, who has guided me throughout my postgraduate studies with patience. I deeply appreciate all the teachings, training, guidance and advice given by him.

I am also very grateful to have Dr. Cheah Yu-N as my co-supervisor. I want to thank him for all the encouragement, guidance and advice given to me.

In my daily work, I want to thank everyone at the School of Computer Sciences, USM who have helped and supported me to accomplish my postgraduate studies and I want to extend my gratitude to my colleagues from the Faculty of Information, Communication and Technology, UTAR for their support and understanding.

I thank the Institute of Postgraduate Studies (IPS), USM for their support, cooperation and understanding.

Last but not least, special thanks go to my parents, my two brothers and my wife for their unconditional love and unlimited moral support. Without their support and understanding, I may not have been able to complete my postgraduate studies. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACI	Autonomic Computing Initiative
ANFIS	Adaptive Network-based Fuzzy Inference System
CSP	Constraint Satisfaction Problem
DSPR	Dynamic Service Placement and Replication
FL	Fuzzy Logic
FIS	Fuzzy Inference System
MAUT	Multiple Attribute Utility Theory
MCDM	Multiple Criteria Decision Making
MSE	Mean Square Error
MTBF	Mean Time Between Failure
MTBM	Mean Time Between Maintenance
MTTR	Mean Time To Repair
NN	Neural Network
RMSE	Root Mean Square Error
RR	Round Robin
RSerPool	Reliable Server Pooling
SGE	Sun Grid Engine
SLA	Service Level Agreement
SOA	Service Oriented Architecture
TCO	Total Cost of Ownership
VM	Virtual Machine
WAN	Wide Area Network

RANGKA KERJA PENEMPATAN DAN REPLIKASI SERVIS YANG DINAMIK SECARA AUTOMATIK DENGAN PENDEKATAN FORMASI PASUKAN UNTUK MENINGKATKAN KEBOLEHSEDIAAN SERVIS

ABSTRAK

Pengurusan dan pentadbiran servis dalam persekitaran sistem teragih menjadi semakin rumit disebabkan oleh saiz persekitaran sistem teragih yang semakin meluas dan dinamik. Tesis ini mencadangkan satu rangka kerja berautomatik untuk menguruskan servis dalam persekitaran sistem teragih yang dinamik dalam mana sumber yang tersedia untuk servis tersebut akan berubah dari semasa ke semasa. Tujuan penyelidikan ini ialah untuk mereka bentuk satu rangka kerja automatik yang boleh mencari sumber pengkomputeran yang mempunyai prestasi yang lebih tinggi secara berterusan serta menabung sumber pengkomputeran untuk mencapai tahap kebolehsediaan yang baik dengan menggunakan pendekatan formasi pasukan. Pentadbir masih diperlukan tetapi tidak perlu membuat keputusan aras rendah seperti keputusan tentang penempatan servis yang sebenar serta tatacara "failover" untuk setiap servis. Tambahan pula, penyelidikan ini juga mencadangkan satu teknik penilaian sumber yang menggabungkan logik kabur dan "Adaptive Network-based Fuzzy Inference System" (ANFIS). Ia menggunakan FL untuk mengumpulkan keperluan servis daripada pentadbir dan ANFIS membenarkan pentadbir membuat penyelerasan halus terhadap proses penilaian sumber semasa persekitaran sistem teragih berubah. Simulasi telah dibangunkan untuk menilai keupayaan rangka kerja yang dicadangkan ini dari segi meningkatkan kebolehsediaan servis dengan menggunakan cara suntikan kegagalan berturut-turut yang berbilang. Keputusan dari

penilaian tersebut menunjukan cadangan ini dapat meningkatkan kebolehsediaan service walaupun mengalami kegagalan berbilang sumber. Selain itu, penyelidikan ini juga menerangkan batasan dan penambahbaikan terhadap cadangan kerja ini pada masa depan yang berpotensi.

AUTOMATED DYNAMIC SERVICE PLACEMENT AND REPLICATION FRAMEWORK USING TEAM FORMATION APPROACH TO ENHANCE SERVICE AVAILABILITY

ABSTRACT

Managing and administering services in the distributed environment are getting more complicated as the size of the distributed computing environment grows larger and becomes more dynamic. This thesis proposes an automated framework to manage services in a dynamic distributed environment where the resources available for these services would change from time to time. The aim of this research is to design an automated framework that would continuously search for resources with better performance and pool resources together to achieve better availability using a team formation approach. Human administrators are still required but are freed from making low-level decisions such as to decide the actual placement of the services and to design the failover procedures for each of the services. In addition to that, this research also introduces a resource evaluation method that fused Fuzzy Logic (FL) and Adaptive Network-based Fuzzy Inference System (ANFIS). It uses FL to capture services' requirements from administrators and ANFIS to allow administrators to fine-tune the resources evaluation process when environment resources change. Simulations were developed to evaluate the ability of the proposed framework in improving service availability using multiple consecutive random fault injection method. The experimental results showed that the framework can improve service availability even during multiple consecutive resource failure. Besides that, this research also highlights the limitations and potential future enhancement of the proposed work.

CHAPTER 1

INTRODUCTION

1.1 Overview and Motivation

The introduction of various well-distributed computing paradigms such as grid computing, cloud computing, and ubiquitous computing along with the advancement of distributed computing technologies namely server virtualization, service-oriented architecture (SOA), and distributed agents, have greatly increased the flexibility and scalability of distributed systems. However, these flexibility and scalability are not achieved without problems of their own, notably the difficulty to manage and administer resources in a distributed environment [1-3], in view of the tediousness of maintaining availability, cost, security and performance of a large number of services running on a huge number of heterogeneous machines across different networks.

Conventional resource management methods that use human administrators to manage dedicated and specialized infrastructures such as cluster computing to host a fixed set of services are no longer suitable in view of the rapidly growing size of networks on the Internet [4]. In general, administrators are required to constantly monitor utilization of the services and physical resources, define high-level utilization policies, and perform low-level implementation in order to ensure performance of all the services are within their respective acceptable range. As the network size increases, intricacy of the distributed system will affect the productivity of administrators [5] and often results in high operating cost and non-optimal use of resources.

The need for a self-management framework has begun to emerge [6-14] and it would be essential when administrators could no longer handle the scale and heterogeneity of the ever growing distributed computing environment. One of the most notable movements of self-management automation was the Autonomic Computing Initiative (ACI) [15] by IBM in 2001. The research direction of ACI was to design a distributed computing system which can autonomously configure, heal, optimize and protect according to changes in the environment with minimal intervention from administrators. This self-management system was inspired by the human body's autonomic nervous system, which controls functions such as heart beat, blood sugar and respiration without requiring conscious human action.

Although the goal of autonomic computing has been set, the goal has yet to be realized completely [16]. This is due to the fact that the scope of autonomic computing goes beyond the traditional boundaries of automation. It requires the components in a distributed environment to work together as one "super organism" that exhibits the capabilities of self-healing, self-configuring, self-protecting and self-optimizing [17]. However, a common acceptable definition of an autonomic system's characteristics has yet been standardized [4]. For instance, it is difficult to distinguish between self-protecting, self-healing and self-optimizing mechanisms, as all of these mechanisms serve in an interwoven manner to improve the state of a system.

Long before ACI, there were many attempts made to automate administrative tasks and reduce the difficulty of distributed resource management. Organizations have been sharing the best practices of ICT [18] and there have been continuous

development of distributed system management tools for network monitoring, performance monitoring, load balancing and fault tolerance. Unfortunately, all of these tools still require a lot of human intervention to the finest detail.

The advancement of virtual machines (VMs) has enabled administrators to decouple the dependency between software and hardware at the expense of a slight degradation of the hardware performance [19]. However, as the technologies of machine virtualization and hardware improve, any difference between VMs and physical machines would soon be insignificant [20]. The ability of VMs in reducing setup time, migrating services without downtime, and consolidating multiple underutilized servers into a smaller number of physical machines have made VM technology seem a viable way to reduce the intricacy of resource management [21, 22]. Unfortunately, this has yet to become a reality. This is because the administrators are still required to plan and map the VMs to physical hosts manually. Even though automated VM migration tools exist, these still require the administrators to provide low-level detailed migration rules in advance [23].

The emergence of cloud computing seems to be a viable solution in reducing the total cost of ownership (TCO) of many distributed systems. This is achieved by sharing hardware infrastructure hosted in a third party data centre [23, 24]. However, reducing the TCO does not reduce the administrative intricacy, and due to the ease of deploying and scaling new cloud instances, cloud computing will continue to attract larger scale of distribute systems to be developed and deployed [25]. Cloud computing does not solve the problem of managing a large number of services [26]. Moreover, it is possible that services are deployed across multiple networks

indiscriminately. Hence, services running on cloud computing environment will still require the administrators to plan the architecture of the distributed system carefully to leverage on the scalability offered by cloud computing [27, 28]. Making a service available in a cloud computing environment does not mean that the service is automatically converted into high availability mode [29].

Thus, a self-management framework for a distributed computing environment is required to act as a middleware between the services and the physical resources. This would allow human administrators to cope with the dynamic and fast-growing distributed computing environment.

1.2 Problem Statements and Research Questions

As a result of the ever-growing distributed computing environment, many service placement algorithms [30-37] have been introduced to automatically manage the services. Besides being different in terms of their centralized and decentralized architectures, each of them was built on different resource evaluation schemes with different objectives. The decisions made are also based on different criteria. Most service placement algorithms are application specific and aim to improve service performance or to reduce operation cost. A majority of the algorithms did not consider the availability of resources which will directly affect service availability.

Besides the differences in architecture and objectives, most of the solutions proposed to improve service availability [38-40] are based on having redundant servers to mask failures. The effectiveness of these solutions is often very dependent on the

experience and knowledge of the system administrators. Although the additional servers increase service availability, the difficulty in managing the distributed system increases as well.

In addressing these problems this research focuses the following questions:-

- How to free administrators from making low-level decisions such as to decide the actual placement of services and design the failover capabilities for each service?
- How a self-management framework could make service placement and replication decisions according to the requirements given by the system administrators?
- What are the factors that contribute directly to the availability of a service and how to enhance the availability of services even during the event of multiple consecutives resources failures?
- How effective is the proposed solution in term of enhancing services availability?

1.3 Research Objectives

The main objective of this research is to design a self-management framework that manages services in a dynamic distributed environment according to the administrators' service requirements. Depending on the environment, this self-management framework will continuously manage the placement and replication of its services to maintain the performance and availability of the services.

The proposed self-management framework is designed based on the following sub-objectives:-

1. To perform low-level decisions on the placement and replication of the services on behalf of an administrator. Continuously manage services on behalf of the administrators in terms of availability, performance and cost.
2. To autonomously evaluate and select resources according to the preference of administrators. The preference of the administrators must be configurable into the framework.
3. To automatically increase the availability of managed services even during the event of multiple consecutives resource failures by taking into account of factors that affect service availability.
4. To simulate a distributed environment in order to measure the effectiveness of the proposed self-management framework in increasing the availability of the managed services in different scenarios.

1.4 Research Contributions

The proposed self-management framework is called the **D**ynamic **S**ervice **P**lacement and **R**eplication framework (DSPR). At first glance, this work might seem similar to some existing work such as server failover. In general, failover techniques can be categorized into two types namely static and dynamic. Static failover requires the administrator to define the hardware involved and carefully plan the failover procedure [41, 42]. Dynamic failover uses a pool of servers that are identified by the administrator to perform failover instead of specifying the exact servers to be used [43, 44]. Although failover techniques can mask hardware failure and increase service availability, they cannot be used to improve service performance.

There may also be some semblance of this work to resource allocation techniques. Resource allocation in distributed computing is a process of mapping computational tasks to processing units [45]. There are two types of computational tasks namely batch jobs and services. Resource allocation for batch jobs is commonly known as jobs scheduling while resource allocation for services is known as service placement. From our literature review, we found that most service placement algorithms focus on performance and cost effectiveness rather than service availability. Figure 1.1 distinguishes our framework from the existing work such as failover and resource allocation.

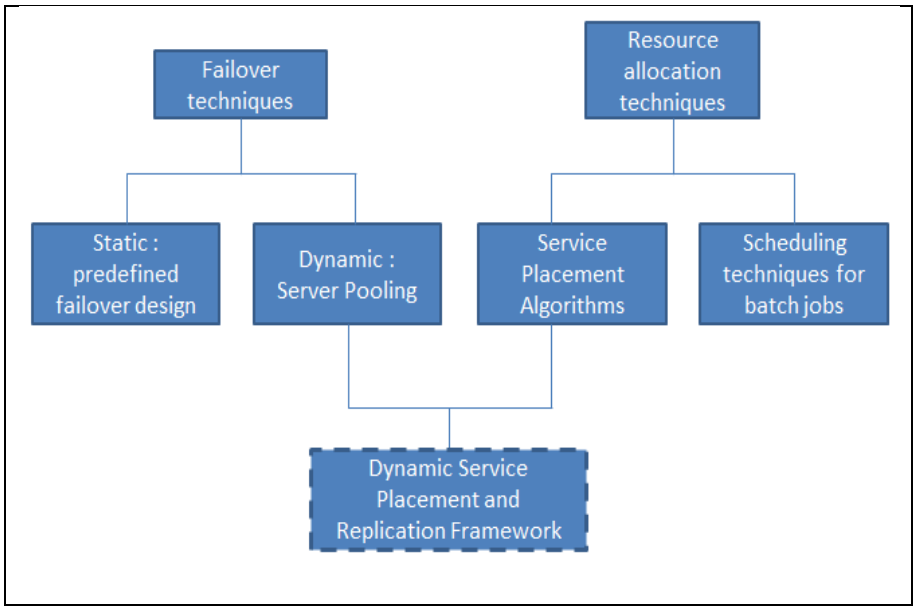

Figure 1.1 Research Contribution

The following is the contributions of this research:-

1. The main contribution of this research is the design of the self-management framework. Depending on the environment, the framework leverages on the service placement and replication techniques automatically and continuously maintain the

availability, cost, and performance of all the managed services are within the range specified by the administrator.

2. Besides that, this work also identified an appropriate resource evaluation technique that uses FL and ANFIS together to be employed in the proposed framework. The resource evaluation technique has the ability to represent the administrator's resource management policies using rules with approximate values which are more intuitive for the administrator to set compared to assigning specific values for the rules (please refer section 2.4.1 for details). In addition, the proposed resource evaluation technique has the ability to learn from the feedbacks given by the administrators to make better decisions preferred by the administrators in the future.

3. In the event of resource failures, the framework will search for opportunities to enhance service availability by migrating services to other available resources. Even in low-availability environments, DSPR will resort to replication to improve service availability. Thus, the design of this framework enables physical resources to be added and removed from the distributed environment without having to be concerned about the services running on the framework. From a user's perspective, services managed by DSPR that are running in a dynamic distributed environment would be perceived as services running on high-availability infrastructure.

4. A simulator was developed to evaluate the effectiveness of the proposed solution in term of service availability. The experimental results also highlights the limitations and potential future enhancement of this research.

1.5 Methodology

The design and development of the proposed framework is divided into 3 stages. In the first stage, a team formation algorithm that makes decisions on service placement and service replication is designed. The team formation algorithm is inspired by the way humans team up autonomously to solve problems that cannot be achievable individually. Bruce Tuckman proposed a model of group development that for a team to grow, to overcome challenges, to solve problems, and to deliver solutions, the team has to go through stages such as forming, storming, norming and performing [46]. Team formation algorithm using stages approach but does not use similar stages.

The team formation process is designed to be a closed control loop and an adaptive process where the team will continuously recruit new members (other available machines in the resource pool) and remove existing underperforming members (existing service-host pairs that are not performing) until the best working group is attained to satisfy the service level specified by the administrator. Whether a member is performing or underperforming depends on the resource evaluation technique that is used and how it is configured. Please refer to Section 3.4 on the resource evaluation technique proposed in this work.

In the second stage, we identified potential resource evaluation techniques for the team formation process to perform resource evaluation. Besides reviewing existing resource evaluation techniques, we also explored fuzzy inference system (FIS) and adaptive-network-based fuzzy inference system (ANFIS). In the same stage, we implemented and performed a preliminary evaluation on the team formation

algorithm with the proposed resource evaluation technique using simulations. We continued improvising the framework based on problems that arose from the preliminary evaluation results. This led us to explore other possible methods to reduce the search space and speed up the team formation algorithm.

In the final stage, we implemented the improvised framework and evaluated the ability of the proposed framework in managing services via a simulation. The simulation simulates a dynamic distributed environment with services to be managed by the proposed framework. The simulated environment includes an environment where resources are randomly turned on and off, and an environment where resources are randomly turned off without being turned on again. We compared the proposed framework with existing failover techniques with the same simulated environments.

1.5.1 Research Scope

The focus of this research is the design of a self-management framework and an algorithm for service placement, and replication. For its complete implementation, the framework would require other supporting technologies such as:-

- Service migration solutions such as virtual machines to enable services to be migrated and deployed on heterogeneous types of machines.
- Network protocols such as heartbeat protocol and network algorithms such as election algorithm to detect and identify faulty nodes.

- Pure peer-to-peer (P2P) networking architecture that does not require any super nodes to remove the dependency of the framework on any centralized components.

However, these technologies are not the focus of this research. There are some existing work done in [11, 13] that identified the key components which are required in an autonomic framework.

Instead, the scope of this work focuses on the DSPR framework, particularly on the service placement and replication decision making component, to enhance service availability with performance and cost constraints. The framework is designed to allow violations on certain constraints when the primary criterion is threatened depending on the administrator's preference. For instance, the performance and cost constraints can be violated when the availability of the service is threatened. Many existing work have explored automated service placement but not many of them focus on more than two criteria, or allow constraints to be violated during computational resource crisis.

However, measuring and benchmarking a self-management framework is not without problem of its own. Notably, different self-management frameworks exhibit different levels of automation and they could not be quantitatively measured; e.g. different amount of human intervention required [47] and different objectives such as self-configuration [48], self-healing [49], self-optimization [50] and self-protection [51]. Self-management frameworks are geared more towards dependability benchmarking [52, 53]. Unlike the well established performance benchmarking, dependability

benchmarking is a relatively new research field and it is still lacks of commonly accepted benchmarking method [54].

Therefore, for the purpose of this research, the proposed self-management framework had to be evaluated using simulation and the evaluation focuses on how the framework design is capable of enhancing service availability using fault injection method proposed in [53]. Thus, the experimental results shows the ability of the framework to perform self-management but it would not be able to measure the effectiveness of the proposed solution. In order to observe the proposed framework from an end-user perspective, we also implemented the simulation across a wide area network (WAN).

1.6 Thesis Layout

This thesis is organized as follows. In Chapter 2, we provide a review of existing service placement algorithms, server failover solutions, and the fundamental details of Fuzzy Logic and the Adaptive Network-based Fuzzy Inference System (ANFIS). Chapter 3 presents the details of the proposed self-management framework with extensive justifications. Details of the proposed framework and simulation implementation are presented in Chapter 4, and the experimental results are presented in Chapter 5. Finally we conclude this thesis in Chapter 6 by revisiting the contributions, and by suggesting some future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter begins with the definition of availability, and exploration of existing techniques that are commonly employed to improve the availability of services in distributed computing systems. This is followed by a review of existing service placement algorithms. This exploration includes a discussion on the advantages and disadvantages of existing algorithms, and the resource evaluation functions employed.

2.2 Definition of Availability

Availability is often confused with reliability. Reliability is the probability of a system performing its intended function over a given period of time without interruption, while availability measures the ability of a system to be up and ready to be used at a random point of time [55]. Reliability is commonly measured using Mean Time Between Failure (MTBF), and Failure Rate (FR) [56] where MTBF is the average time between consecutive failures, and FR is defined as the inverse of MTBF. Equation 2.1 illustrates the inverse relationship between MTBF and FR, while availability is mathematically represented in equation 2.2.

$$FailureRate = \frac{1}{MTBF} \tag{2.1}$$

$$Availability = \frac{MTBF}{MTBF + MTTR} \quad (2.2)$$

where *MTTR* is the mean time to repair. MTTR is a value reflecting the maintainability of a system. Maintainability is the probability of a maintenance action completing within a given duration [56].

From the two equations 2.1 and 2.2, the relationship between reliability and availability can be distinguished more clearly. From equation 2.1, it shows that high failure rate will result in low MTBF and reliability is improved when the duration between failures is extended. From equation 2.2, it shows that reliability is only one of the factors that affect availability and there is another factor that affects the availability of a system which is the maintainability of a system. Thus, poor reliability does not necessarily imply low availability. From the availability point of view, poor reliability can be compensated by having shorter maintenance time, availability of a system can be increased by having a longer MTBF and shorter MTTR. For systems that cannot be repaired, the systems' availability is equivalent to the systems' reliability [57].

Besides the general definition of availability mentioned above, there are several other more specific definitions for availability [57-59] namely *inherent availability*, *achieved availability* and *operational availability*. *Inherent availability*, the availability function only considers the downtime of corrective maintenance and it assumes that spare parts and manpower are always available without delays. It is used to determine availability of the design of the equipment. *Achieved availability* of a system, the availability function will consider both preventive and corrective

maintenance without including the delay of spare parts and manpower arrival. It is often used to determine the availability of the design of the equipment and facility. Finally, for *operational availability*, instead of computing the mean time between failures, it divides the total system uptime by the total time that the system is expected to operate. Operational availability is a measure of availability over a of duration time, and includes the actual time to perform maintenance and the delay of spare parts, manpower arrival and any administrative waiting time. Therefore, the operational availability is the actual availability that the user experiences.

The difference between the definitions of inherent, achieved, and operational availabilities stems from whether or not the duration of preventive maintenance, corrective maintenance, logistic delay of spare parts, and administrative waiting time were included or excluded in the general availability equation, i.e. Equation 2.2. Table 2.1 illustrates the differences among these availability definitions.

Table 2.1 Various definition of availability functions

Type of Availability	**Equation**	**Definition**
Inherent Availability	$Availability = \frac{MTBF}{MTBF + MTTR}$	Inherent availability considers the downtime of corrective maintenance only. It assumes that spare parts and manpower are always available without delays. It is used to determine availability of the design of the equipment.
Achieved Availability	$Availability = \frac{MTBM}{MTBM + MTTR}$	Achieved availability considers both preventive and corrective maintenance excluding the delay of spare parts and manpower arrival. It is used to determine the availability of the design of the equipment and facility

Operational Availability	$Availability = \frac{Uptime}{Operating\ Cycle}$	Operation availability includes the actual time to perform maintenance and the delay of spare parts, manpower arrival and any administrative waiting time. The operation availability is the actual availability that the user experiences.
Note: MTBF is Mean Time Between Failure, MTTR is Mean Time Between Repair and MTBM is Mean Time Between Maintenance.		

Another commonly used index to indirectly reflect the availability of a system is the downtime of a system [60]. Downtime is often expressed using the duration of downtime per year. On the other hand, availability is often specified in percentage. The relationship between downtime (measured in minutes/year) and availability can be expressed with the following equation:-

$$Downtime = (1 - Availability) \times total\_minutes\_in\_a\_year \quad (2.3)$$

Where *total_minutes_in_a_year* = 525,600 by assuming that one year has 365 days. Downtime provides a more intuitive value for understanding the difference between availability values of two systems [61]. For instance, comparing the availabilities of two systems, 99.9% and 99.999% might not seem to have much difference but in terms of downtime, 99.9% availability has 8.76 hours/year of downtime where as 99.999% has 5.256 minutes/year of downtime.

The availability of a system is often dependent on the aggregation of its components' respective availabilities. The aggregation process is done by computing the interconnection of components of the system using the following two rules [62]:-

Rule 1: If failure of a component will cause the system to fail, then the availability between the components are considered to be operating in series. Figure 2.1 illustrates availability in series and the aggregated availability is shown in equation 2.4.

Figure 2.1. Availability in series

$$Series\ Availability = A_{ComponentX} \times A_{ComponentY} \quad (2.4)$$

where $A_{ComponentX}$ is the availability for component X and $A_{ComponentY}$ is the availability for component Y respectively.

Rule 2: If there is another component to take over a failed component, then the availability between the components are considered to be operating in parallel. Figure 2.2 illustrates availability in parallel and the aggregated availability is computed using equation 2.5.

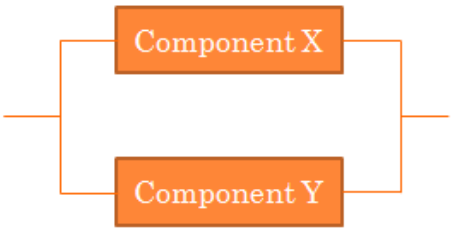

Figure 2.2 Availability in parallel

$$Parallel\ Availability = 1 - (1 - A_{ComponentX}) \times (1 - A_{ComponentY}) \quad (2.5)$$

where $A_{ComponentX}$ is the availability for component X and $A_{ComponentY}$ is the availability for component Y respectively.

2.3 Existing Approaches to Enhance Service Availability

In most high-availability distributed systems, redundancy is used to increase availability and mask failure [38, 39, 63]. In case of failure, the redundant server will take over the responsibility of the actual server. This switching process is known as failover. Although the failed server is not repaired, the redundant server makes the system appear as available and operating as usual to the users. Once the failed server is repaired, a failback procedure is initiated to restore the configuration back to original before another failover occurred. This failback procedure usually requires human intervention. Although redundancy is able to increase the availability of a system, it is not without problems of its own, notably additional cost and underutilized resources. For instance database mirroring and server replication techniques [64] require additional hardware that do not contribute to the performance of a system. The cost would be even higher if the system is required to withstand multiple consecutive resource failure.

There are also techniques that are available to reduce the number of underutilized resources by using the additional resources to improve the service performance. For example, server content caching [65] and load balancing techniques [66] were introduced to use the additional resources to improve service performance. However, the design and management of these existing methods to improve service availability

in a distributed system itself are by themselves complicated for the system administrators [67]. This is because the administrators need to decide on the architecture, roles, and relationships of the servers, while matching applications to servers. These tasks would be even more complicated in a large distributed environment.

In order to reduce the intricacy, technologies such as clustering [40, 68, 69] and reliable server pooling (RSerPool) [70, 71] were devised. For example, the clustering failover technique used in Sun Grid Engine (SGE), SGE's master node has all its child compute nodes arranged in a serial manner to form a series of redundant nodes. In the event of the master node failure, the first compute node in the serial arrangement will take over and continue its operation. This process can be repeated until all the nodes in the SGE fails [40].

At a glance, RSerPool appears similar to clustering. However, RSerPool is different as it dynamically selects the redundant node, i.e. the selection of the redundant node is not predefined and arranged in a serial manner. Administrators do need to manage RSerPool by defining the redundant server selection policies. There are two types of server selection policies: static or dynamic [71]. Static policies use predefined schemes such as round robin (RR) where servers are selected sequentially in a cyclic manner. Besides RR, there are weighted RR where weights are used to indicate the server's capacity. On the other hand, policies make decisions based on the current state of the system. For example, the least used selection policy selects the server with the lowest load. Besides using the current state of the system, Dreibholz [72] proposed the distances-aware least-used policy which uses the distance between

servers to make server selection decisions. The purpose of having such server selection policies is to allow servers to be distributed over a large geographical area to ensure survivability in the event of disasters such as earthquake, volcano eruptions or tsunami. Table 2.2 highlights the advantages and limitations of existing methods.

Table 2.2 Overview of the advantages and limitations of existing methods

	Improves Availability	Handles Multiple Consecutive Failover	Selects Failover Server Automatically	Manages Size of Server Pool for Failover Automatically	Considers Performance While Improving Availability
Server Mirroring Technique	✓	✗	✗	✗	✗
Clustering Failover Technique	✓	✓	✗	✗	✗
RSerPool Technique (static and dyanmic)	✓	✓	✓	✗	✗

To conclude, existing techniques are capable of improving service availability. However, most of the techniques did not consider the difficulty in managing the distributed system as the system grow larger. Although the dynamic server selection policies of RSerPool appears to be very similar to the proposed work and capable of handling multiple consecutive failovers, RSerPool does not have the ability to manage the size of the server pool yet. An administrator is still required to decide the size of the cluster in order to achieve the desired level of availability. Besides that, existing failover techniques only focus on availability and not performance. For instance, a failover would not be initiated when a web server which is heavily loaded is still running. However, from the users' perspective the web server would be

perceived as not available. Thus, resource management automation is necessary to help human administrators cope not only with server failures but also to ensure that the performance of the server is within an acceptable range.

2.4 Service Placement Algorithms and Resource Evaluation Functions.

A survey of existing swarm intelligence techniques for self-organization and service placement was carried out by Andrzejak in year 2002 [30]. They opined that a good service placement solution should be decentralized while not overloading the communication channels. Besides comparing ant colony optimization, broadcast of local eligibility, and intelligent agents, they also compared simple and stateless techniques such as round robin and simple greedy algorithms. However, they concluded that different approaches have different levels of tradeoff between speed and solution accuracy. Each offers better performance in some circumstances and they proposed that a combination of techniques is necessary to solve the self-organization and service placement problem.

Service placement algorithms are required to discover and select appropriate resources for all the services based on the preferences defined by the administrator. Different service placement algorithms employ different resource discovery methods and different resource evaluation functions. Resource discovery methods can be classified into centralized or decentralized and either complete or heuristic. In order to distinguish preferable resources from the non-preferable resources, these service placement algorithms require a resource evaluation function. Resources are usually

distinguished using criteria such as performance, dependability, security and cost [73].

Therefore, these service placement algorithms, besides being different in terms of their centralized and decentralized architectures, each of them is built on different dynamic allocation schemes that have different objectives, and the decisions made are based on different criteria. Table 2.3 summarizes the service placement objectives and resource evaluation techniques.

Table 2.3 List of existing service placement algorithms with respective service placement objectives and resource evaluation techniques

No	Related Work	Service Placement Objectives	Resource Evaluation Technique
1	Hien et al. [74]	Optimize global utilization which consists of SLA fulfillment and operating cost.	Utility function-based
2	Marjan et al. [71]	Enhance service availability by automatically select backup candidate using predefined scheme.	Rule-based and Procedure-based
3	Karve et al. [33]	Maximize service performance with minimal number of placement changes.	Procedure-based
4	Ardagnaa et al. [36]	Maximize service revenue while balancing the cost of using the resources. The cost includes energy, software and hardware required.	Utility function-based
5	Adam et al. [34, 35]	Improve service performance by performing automated service placement by mapping CPU demands to CPU supplies.	Utility function-based
6	Famaey et al. [37]	Improve service performance by not only mapping CPU demands to CPU supplies but taking network latencies into consideration as well.	Constraint-based
7	Verma et al. in [75],	Reduce power consumption via service consolidation using virtual machines.	Procedure-based
8	Nogueira et al. [31]	Ensure service QoS within acceptable level. It	Utility function-based

		automatically tradeoffs between performance and QoS.	
9	Oikonomou et al. [76, 77]	Determine the optimal location of services without using global information.	Utility function-based
10	Menasce et al. [78, 79]	Improves QoS of a system by automatically selecting appropriate group of services	Utility function-based
11	Herrmann [80]	Designed an adaptive service placement algorithm to find a stable and low-cost replica placement.	Rule-based with Utility function-based

We found that utility function-based, rule-based, constraint-based, and procedure-based methods are the commonly used techniques to perform resource evaluation. Hence, the review of existing service placement algorithms is classified into subsections according to the resource evaluation method they employed. In this work, we explore the potential of using fuzzy logic (FL) to solve the resource evaluation problem in view of the ability of FL in solving multi-criteria decision making (MCDM) problems [81]. The resource evaluation process that involves more than one criterion is very similar to the MCDM problem. Figure 2.3 illustrates the classification of resource evaluation functions.

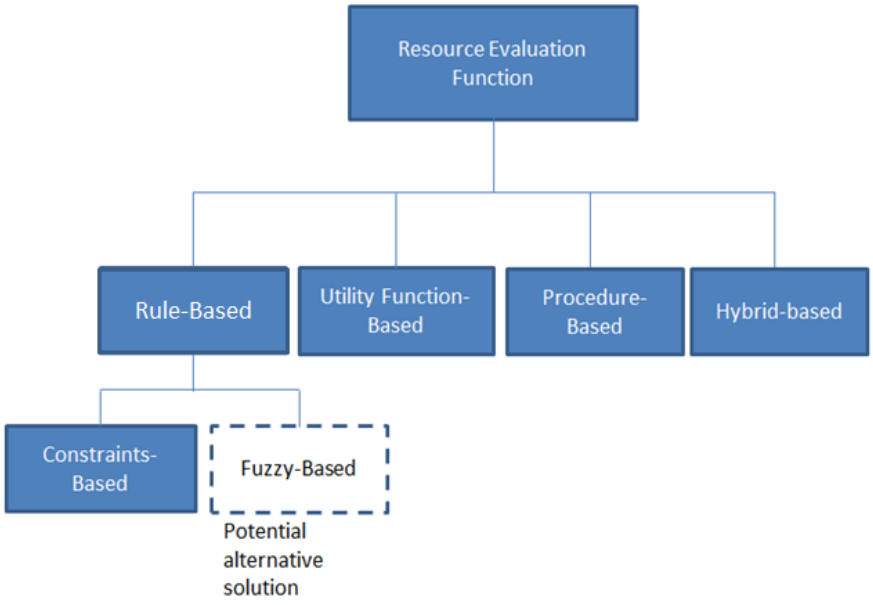

Figure 2.3 Classification of resource evaluation functions in this work.

2.4.1 Rule-based techniques

A rule consists of two parts namely, antecedent and consequent [82]. In resource evaluation, rules are used to represent the knowledge and preferences of administrators. The antecedent of a rule can be used to represent the state of a resource and the conditions of the environment that need to be fulfilled while the consequent of a rule is used to represent the suitability of the resource being selected. The execution of these rules is managed by an inference engine. In general, there are two principle ways to execute rules, namely forward chaining and backward chaining.

For instance, in grid computing, resource requirements are usually represented in the form of rules and constraints. It is a set of resource requirements of an application that must be fulfilled in order for the application to be executed [83, 84]. In addition, many existing resource management tools require the administrator to provide low-level instructions such as defining the maximum CPU load and ideal CPU load [85].

Unfortunately, the rule-based technique cannot tolerate situations in which the resources only fulfill some of the requirements. No rules can be fired unless all conditions in the antecedent of a rule are met. For instance, if an administrator uses conventional production rules to represent a preference for a 3GHz CPU at the cost of 50 cents per hour, such a rule will turn down a 2.98GHz CPU even if the price is 25 cents per hour. Although a huge number of rules can allow rule-based techniques to deal with more conditions, it is difficult for administrators to ensure that all possible conditions have been considered, especially those involving tradeoffs between the criteria. For example, higher CPU with lower availability can be