IMPROVING RELIABILITY OF SERVICE ORIENTED SYSTEMS WITH CONSIDERATION OF COST AND TIME CONSTRAINTS IN CLOUDS

WEI GUO

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy

School of Computing and Engineering
The University of Huddersfield

March 2016

Abstract

Web service technology is more and more popular for the implementation of service oriented systems. Additionally, cloud computing platforms, as an efficient and available environment, can provide the computing, networking and storage resources in order to decrease the budget of companies to deploy and manage their systems. Therefore, more service oriented systems are migrated and deployed in clouds. However, these applications need to be improved in terms of reliability, for certain components have low reliability. Fault tolerance approaches can improve software reliability. However, more redundant units are required, which increases the cost and the execution time of the entire system. Therefore, a migration and deployment framework with fault tolerance approaches with the consideration of global constraints in terms of cost and execution time may be needed.

This work proposes a migration and deployment framework to guide the designers of service oriented systems in order to improve the reliability under global constraints in clouds. A multilevel redundancy allocation model is adopted for the framework to assign redundant units to the structure of systems with fault tolerance approaches. An improved genetic algorithm is utilised for the generation of the migration plan that takes the execution time of systems and the cost constraints into consideration. Fault tolerant approaches (such as NVP, RB and Parallel) can be integrated into the framework so as to improve the reliability of the components at the bottom level. Additionally, a new encoding mechanism based on linked lists is proposed to improve the performance of the genetic algorithm in order to reduce the movement of redundant units in the model.

The experiments compare the performance of encoding mechanisms and the model integrated with different fault tolerance approaches. The empirical studies show that the proposed framework, with a multilevel redundancy allocation model integrated with the fault tolerance approaches, can generate migration plans for

service oriented systems in clouds with the consideration of cost and execution time.

Acknowledgements

I would like to show my deepest gratitude to my supervisor, Prof. Joan Lu for her supervision and financial support for my tuition fee and some living cost. Thanks to her trust. I also thank my second supervisor, Dr. Qiang Xu, who gives me the suggestions of my thesis.

I shall extend my thanks to Dr. Hugh Osborne for his valuable advice to my progress reports. I also want to thank the School of Computing and Engineering for offering the great opportunity and facilities of study.

I want to thank my parents, Zhenjie Guo and Li Zhang who support my study no matter how big difficulties arise. More than anyone else, I would like to thank my wife, Jinyan Li, for her patience and love.

Declaration

This dissertation is submitted for the degree of Doctor of Philosophy at the University of Huddersfield. I declare that the work in this dissertation was carried out in accordance with the Regulations of the University of Huddersfield.

This work is original except where acknowledgment and references are made to the previous work. Neither this nor any substantially similar dissertation has been or is being submitted for a degree, diploma or other qualification at any other universities.

Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Introduction

Web services are becoming more and more popular for the design and development of systems and applications in distributed network environments. Service oriented systems provide interoperable machine-to-machine communication and interaction in distributed networks. Organisations have a large number of software systems, which may need to communicate with each other at service and system levels. Web service technology is common and available to support communication and data exchange between two systems or components. It can provide unique and standard interfaces (Ahmed, Wu, & Zheng, 2015) for transferring messages. Additionally, systems may be designed and implemented in various programming languages, which may cause difficulties in integrating their services or functions.

Web services can be described by universal description, discovery and integration (UDDI), which defines the accessed components of systems and the data types of their parameters. An XML-based interface definition language, Web services description language (WSDL) is utilised to depict functionalities of a service (Qiu, Zheng, Wang, Yang, & Lyu, 2014). There are two roles in communication between two systems, which are the service provider and the service requester, respectively. Service requesters send requests and the service provider is responsible for processing requests and sending responses back to the service requester. UDDI is called the service broker. Specifically, if one system requires a specific function provided by a service of another application, it can be found out via UDDI. Thereafter, the service requester sends data via a protocol called the simple object access protocol (SOAP). Once the service provider receives the request, it is processed and its response is sent back. Representational state transfer (REST) or RESTful Web services are another way of implementing communication between computers. Computer systems can send requests and manipulate Web resources with a uniform and stateless operations (Maciel & Hirata, 2013). The usage of a stateless protocol is for fast performance.

In the software reliability field, four methods are commonly utilised to enhance software reliability, which are separately fault prevention, fault tolerance, fault removal and fault forecasting (Qiu, Zheng, Wang, Yang, & Lyu, 2014). Current systems and applications are commonly complex, including a lot of functionalities and services which are implemented with various technologies. A function of an application may consist of several subcomponents. Fault prevention and fault removal strategies are insufficient to prevent the occurrence of faults and remove software faults (Qiu, Zheng, Wang, Yang, & Lyu, 2014). Fault forecasting may detect possible software faults. Certain faulty components, however, cause serious economic losses if these faults cannot be identified and occur. The only possible alternative to solve faults in distributed systems is fault tolerance strategies, which adds redundant equivalent components into systems to tolerate faults so as to improve their reliability (Zheng & Lyu, 2015). In addition, distributed systems are also influenced by the deployment platforms (Qiu, Zheng, Wang, Yang, & Lyu, 2014).

Cloud computing is an approach of availing network, computational and storage resources to end-users through the Internet from companies, organisations and individuals. Cloud computing provides shared network, computational and storage resources on demand. End users of clouds can access these resources and return them after utilisation. Providing resources on demand enables users to access cloud computing platforms according to their actual requirements, which is like the electricity grid (Altmann & Kashef, 2014; Armbrust, Fox, Griffith, Joseph, Katz, Konwinski, Lee, Patterson, Rabkin, Stoica, & Zaharia, 2009). Clouds are universally being utilised by a large number of companies, and more and more legacy applications are being migrated to cloud computing platforms (Qiu, Zheng, Wang, Yang, & Lyu, 2014). Several companies have already provided cloud services to end users, such as Gmail and OneDrive (Rajaraman, 2014). Figure 1.1 depicts an example of the deployment of a service oriented in clouds. Software as a service (SaaS) is a delivery model in which applications are deployed in distributed systems and accessed by users through Web browsers (Kureshi, Pulley, Brennan, Holmes, Bonner, & James, 2013). The circles at the SaaS layer are services distributed in virtual machines supported by physical machines at the physical layer.

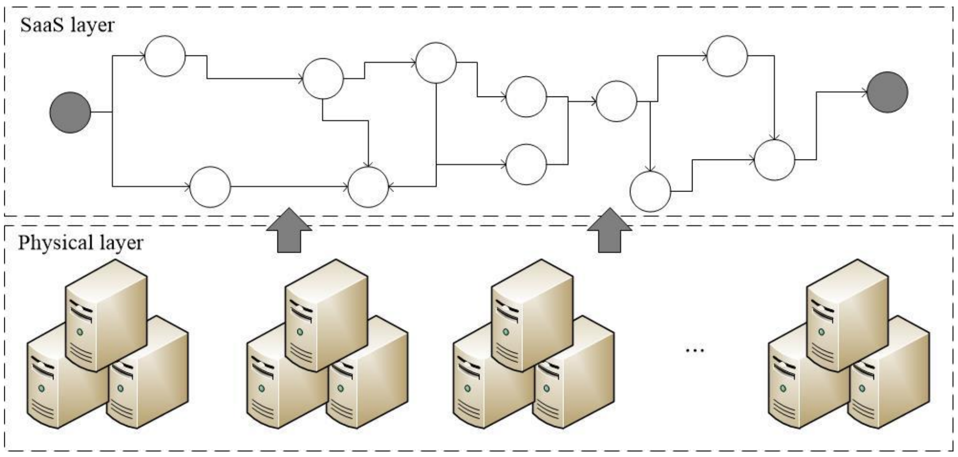


Figure 1.1: An example of the deployment of service oriented systems in clouds

Service oriented systems are deployed in cloud computing platforms as shown in Figure 1.1. To improve the reliability, fault tolerance strategies can be employed (Qiu, Zheng, Wang, Yang, & Lyu, 2014). However, more redundant units add additional costs to the configuration of fault tolerance strategies and redundant components, which forces companies deploying services to pay more expenses. Therefore, a trade-off between the reliability of systems and cost needs to be considered. The problem is generally called redundancy allocation problems (RAPs) (Fyffe, Hines, & Lee, 1968), which are used to depict systems consisting of several subsystems. The aim of RAPs is to maximise the reliability of systems by allocating redundancy to these subsystems under the constraints, such as cost, etc. RAPs are general nonlinear integer programming problems and NP-hard (He, Wu, Xu, Wen, & Jiang, 2013). Therefore, RAPs attempt to identify the optimal redundancy allocation of systems in order to maximise their reliability under several constraints.

Multilevel redundancy design is commonly employed in computing systems, control systems, communication systems and critical power systems(Wang, Loman, & Vassiliou, 2004). Figure 1.2 illustrates an example of a four-level system. In multi-level systems, some assumptions (He, Wu, Xu, Wen, & Jiang, 2013; Kumar, Izui, Yoshimura, & Nishiwaki, 2009; Wang, Tang, & Yao, 2010) are commonly made: (1) units at system and subsystem levels are serial; (2) for a unit at the component level, the relation of duplicated units is parallel; (3) units at all levels can have redundancy allocation.

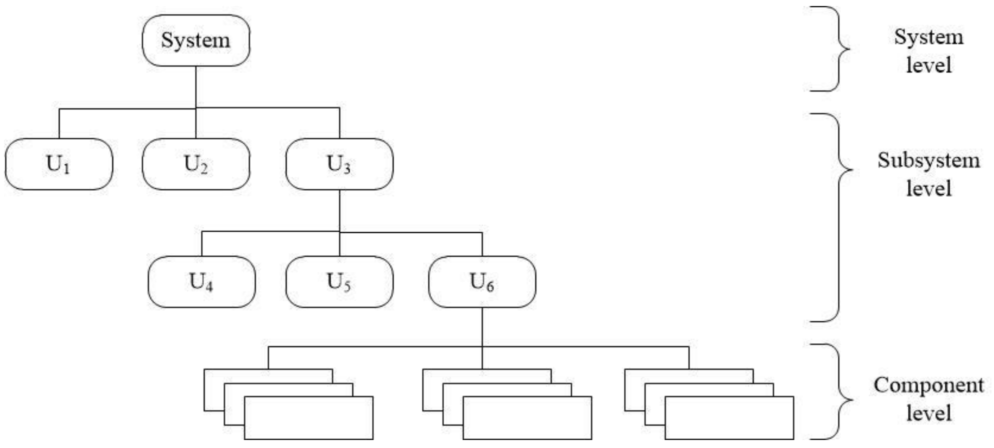


Figure 1.2: A four-level multi-level system (Kumar, Izui, Yoshimura, & Nishiwaki, 2009)

1.2 Importance and significance

Redundancy allocation to one unit can improve reliability of a system. However, the additional units of equivalent functions need more extra computational, network and storage resources, which enable companies migrating and deploying service oriented systems in clouds to consider additional expenses. Therefore, it may be necessary to design an algorithm to improve the reliability of systems and take cost into consideration. In addition, the performance of service oriented systems may be represented by the execution time, which can be computed via all their units. Therefore, the proposed algorithm in this work not only enhances the reliability of systems, but also meets the cost and execution time constraints. Importance and significance are depicted as follows.

1) The reliability of service oriented systems needs to be improved, for software bugs are difficult to detect in distributed platforms (Qiu, Zheng, Wang, Yang, & Lyu, 2014).

2) The cost and execution time constraints may be considered for the migration and deployment of service oriented systems to clouds so as to reduce companies' expenses.

1.3 Aims and tasks

When a Web service based system is being migrated into a cloud, redundant services can be assigned in order to improve the reliability. However, currently there is less research to treat it as an optimization problem. In practice, computing, storage and networking resources limit and higher reliability should be balanced. Therefore, this work proposes the hypothesis that migration of Web service based systems into clouds is an optimization problem.

The aims and tasks of this work are described as follows.

1) A migration framework for service oriented systems to clouds needs to be proposed, which utilises a multilevel redundancy allocation model to describe service oriented system structure with the consideration of cost and execution time constraints. The generated migration plan via the framework can guide developers to perform the migration and deployment of service oriented systems to clouds.
2) A model needs to be created in order to describe the redundancy allocation problem for service oriented systems in clouds.

3) An algorithm needs to be created to generate the optimal solution for the migration of service oriented systems under the cost and execution time constraints.

In summary, the aims and tasks in this work are to propose a migration framework with a model describing the structure of service oriented systems in clouds. It can generate an optimal migration plan under the cost and execution time constraints.

1.4 Contributions

The main contributions of this work are summarised as follows,

1) A migration framework of service oriented systems to clouds is proposed, which considers cost and execution time features and is different from previous research (Qiu, Zheng, Wang, Yang, & Lyu, 2014; Zheng & Lyu, 2015). The framework can automatically generate an optimal migration and deployment plan for service oriented systems in clouds.
2) The multilevel redundancy allocation model is utilised to describe the structure of service oriented systems.
3) One genetic algorithm in the multilevel redundancy allocation field is employed to explore the optimal migration and deployment plan under the cost and execution time constraints. Additionally, it can be integrated with some fault tolerance strategies, such as Recovery Block (RB), N-version programming (NVP) and Parallel.
4) The cost and execution time constraints are taken into consideration for the exploration of the optimal migration and deployment plans.
5) A new encoding mechanism is proposed to improve the performance of the genetic algorithm.

In summary, the contributions in this work propose a migration framework with a genetic algorithm expanding TDA-HGA (He, Wu, Xu, Wen, & Jiang, 2013), which is a multilevel redundancy allocation method with two dimensional arrays encoding mechanism and hybrid genetic algorithm. The framework in this work can generate the optimal migration and deployment plan for a service oriented system under cost and time constraints. Additionally, this work proposes a new encoding mechanism to improve TDA-HGA performance.

1.5 Structure of the thesis

Chapter 1 illustrates the importance and significance, aims and tasks and contributions. This thesis proposes a migration framework for service oriented systems in clouds to guide developers in order to improve their reliability under the cost and execution time constraints.

Chapter 2 describes the related work in reliability, service and cloud computing field. The fault tolerance strategies, RB, NVP, Parallel are described.

Chapter 3 depicts the migration framework. The log files and predefined constraints are input parameters and the optimal solution can be generated automatically for the developers.

Chapter 4 describes the multilevel redundancy allocation model, which is improved to take reliability, cost and execution time constraints into consideration. Furthermore, a new encoding mechanism is proposed to improve performance.

Chapter 5 depicts the design of the experiments in this thesis.

Chapter 6 illustrates the empirical studies. The performance comparison between traditional and proposed encoding mechanisms has been performed. The results from the model integrated with RB, NVP and Parallel are shown and compared.

Chapter 7 depicts the conclusion and the future work of the thesis.

Chapter 2 Background

2.1 Introduction

In this chapter, relevant algorithms, approaches, schemes and techniques are listed and discussed. Previous research is listed in terms of the migration of service oriented systems, their reliability, fault tolerance strategies and the multilevel redundancy allocation problem.

2.1.1 Migration of service oriented systems into clouds

Cloud computing platforms enable companies and end users to use computing, networking and storage resources (Zhu, Zheng, & Lyu, 2013). This type of resource organisation decreases hardware and human costs and budgets. Meanwhile, the companies providing the resources of the clouds can efficiently organise computer resources and offer stable and extensible platforms to users. Clouds are ideal computing platforms for small scale companies to deploy their distributed systems and applications (Zheng, Zhang, & Lyu, 2010). The distributed computing environment, however, cannot confirm the stability of the deployed systems. Therefore, the reliability of service oriented systems needs fault tolerance strategies to guarantee stable and reliable services for users (Qiu, Zheng, Wang, Yang, & Lyu, 2014).

2.1.2 The reliability of service oriented systems

Currently, the algorithms and approaches to solve the reliability problem of service oriented systems mainly focus on the selection of some key components of systems, which are frequently invoked by end users or other components. If these components crash or fail to work, the whole system perhaps cannot successfully complete corresponding tasks. If they are related to finance or human safety, financial losses or accidents may occur. Current algorithms and approaches select the key components and provide these components with fault tolerance approaches (Zheng & Lyu, 2015), which enhance the reliability of service oriented systems in clouds. They, however, bring extra costs and execution time, since additional components utilised in fault tolerance strategies require more design, development and configuration costs and execution time.

2.1.3 Fault tolerance strategies

Failure definition: A failure is that the delivered service no longer complies with the specifications (Zheng & Lyu, 2015).

Fault definition: A fault is an abnormal condition or defect at the component, equipment, or sub-system level which may lead to a failure (Zheng & Lyu, 2015)**.**

Fault tolerance approaches assign functionally equivalent components in order to improve reliability (Wang, Tang, & Yao, 2010). Commonly used fault tolerance strategies include RB, NVP and Parallel (Duan, Peisert, & Levitt, 2014; Kotla, Alvisi, Dahlin, Clement, & Wong, 2009 ; Qiu, Zheng, Wang, Yang, & Lyu, 2014). The detailed description of these strategies is as follows,

1) Recovery Block (RB) (Bonvin, Papaioannou, & Aberer, 2010): If a service fails to work, one functionally equivalent service continues to perform the request. If the selected service fails to work either, another sequential service continues to finish the request.

2) N-Version Programming (NVP) (Brin & Page, 1998): All the services in NVP are invoked in parallel and the final response is voted on and determined by the majority of responses from all the services.

3) Parallel (Wang & Li, 2012): A request is assigned to all the services in Parallel. The final result is determined by the first arrival response from the services.

2.1.4 Redundancy allocation problems (RAPs)

RAPs assign additional identical units to a component in a parallel manner to enhance its reliability. As a special problem of RAPs, multilevel redundancy allocation problem (MLRAP) adopts a layer approach to describe the system structure. Theoretically, a system or application can be depicted as a tree structure. Each layer of the tree structure can be allocated with redundant units. MLRAP not only provides redundancy allocation to each component at the bottom layer, but the ones at the abstract layers as well (He, Wu, Xu, Wen, & Jiang, 2013; Onishi, Kimura, James, & Nakagawa, 2007; Wang, Tang, & Yao, 2010; Yalaoui, Chatelet, & Chu, 2005).

2.2 Related work

2.2.1 Migration of service oriented systems

A component ranking framework has been proposed for services in cloud computing platforms (Zheng, Zhang, & Lyu, 2010). A network can be utilized to describe software structure. However, a software system in reliability may be a tree. The location of the services of the applications has been utilised for reliability prediction (Lo, Yin, Deng, Li, & Wu, 2012). The location can affect execution and message transferring time. Two personalised prediction approaches for Web services have been proposed (Zheng & Lyu, 2013). Software reliability can be predicted early (Cheung, Roshandel, Medvidovic, & Golubchik, 2008). An empirical analysis of predictive algorithms has been performed in the reliability research (Breese, Heckerman, & Kadie, 1998). These work describes a system as a graph. Total cost and qualities can be calculated via collecting each service information.

The latency and time-correlated faults of the applications in distributed platforms have been considered in a lot of research. The time correlation of the failures of applications has been investigated in distributed environments (Yigitbasi, Gallet, Kondo, Iosup, & Epema, 2010). The work takes the fundamental network mechanism, failure group window, into account. Dynamic request transferring has been performed in clouds in order to tolerate latency variability (Zhu, Zheng, & Lyu, 2013). Hybrid architecture in clouds has been proposed to decrease the latency of on-demand games (Choy, Wong, Simon, & Rosenberg, 2014). Time cost in distributed environments can affect software quality. The factors impacting the response time of the applications in clouds have been investigated (Wang, Kanemasa, Li, Jayasinghe, Kawaba, & Pu, 2012). The latency of services has been regarded as a QoS indicator in cloud computing platforms (Pedersen, Tahir Riaz, Dubalski, Ledzinski, Júnior, & Patel, 2013). Therefore, time cost of service execution and message transferring should be considered to improve software availability.

The representational state transfer (REST) and SOAP interfaces have been researched in terms of latencies of service invocation and processing (Aihkisalo & Paaso, 2012). The work embeds the latency mechanism into communication protocols. However, if applications are not based on REST or SOAP, the latency mechanism may be not utilized. The latency of Web services in health care systems has been investigated via GSM networks (Meiappane, Murugan, Murugan, Arun, & Ramachandran, 2010). Methods to reduce Web latency have been proposed (Flach, Dukkipati, Terzis, Raghavan, Cardwell, Cheng, Jain, Hao, Katz-Bassett, & Govindan, 2013). A Web service positioning framework has been proposed for response time prediction (Zhu, Kang, Zheng, & Lyu, 2012). The location has been regarded as an element of the decision in a QoS based service recommendation (Yu & Huang, 2014). In summary, latency can influence the performance of service oriented systems and needs to be considered for the migration and deployment of service oriented systems into clouds.

A lot of research has been undertaken in cloud computing (Armbrust, Fox, Griffith, Joseph, Konwinski, Lee, Patterson, Rabkin, Stoica, & Zaharia, 2010; Assuncao, Calheiros, Bianchi, Netto, & Buyya, 2013; Baroncelli, Martini, & Castoldi, 2010; Borsato, 2015; Jung & Sim, 2011; Kumar & Vidhyalakshmi, 2012; Liu, 2013; Lombardi & Di Pietro, 2011; Phyu & Thein, 2011; Qian, Hardjawana, Shi, & Vucetic, 2015; Ro, 2015; Shen, Chen, Shen, Mei, & Pu, 2014; Tian & Meng, 2010; Wei, Pearson, Matsuura, Lee, & Naik, 2014). Cloud computing platforms are the foundation of SaaS providers (Armbrust, Fox, Griffith, Joseph, Katz, Konwinski, Lee, Patterson, Rabkin, Stoica, & Zaharia, 2009). Grid scheduling based on a service level agreement (SLA) has been proposed (Hasanzadeh Mofrad, Jalilian, Rezvanian, & Meybodi, 2016). The resources of clouds have been provided for application deployment (Yangui, Ben Nasrallah, & Tata, 2013). A distributed storage management system has been proposed with reliable, efficient and transparent features (Kosar, Akturk, Balman, & Wang, 2011). A mechanism should be utilized to improve software reliability transparently in clouds. Some studies have been performed based on Citrix (Dunn, 2007; Feng, Luo, & Jin, 2012; Jung, Bae, & Lee, 2011; Schlosser, Staehle, Binzenhöfer, & Boder, 2010; Shirey, Charng, & Nguyen, 2013; Staalinprasannah & Suriya, 2013).

Some cloud computing platforms are being widely utilized. A lot of research is based on OpenStack (Campos, Fernández-del-Castillo, Heinemeyer, Lopez-Garcia, Pahlen, & Borges, 2013; Corradi, Fanelli, & Foschini, 2014; Kim & Mook, 2013; Kostantos, Kapsalis, Kyriazis, Themistocleous, & da Cunha, 2013; Kureshi, Pulley, Brennan, Holmes, Bonner, & James, 2013). Windows Azure is a popular cloud computing platform for software deployment (Cała & Watson, 2010; Dave, Lu, Jackson, & Barga, 2011; Zhao, Ren, Li, & Sakurai, 2012).

Some studies have been performed on the integration of application on cloud computing platforms and big data platforms (Assunção, Calheiros, Bianchi, Netto, & Buyya, 2015; Baek, Vu, Liu, Huang, & Xiang, 2015; Chang, Tsai, Chen, Huang, & Hsu, 2015; Hashem, Yaqoob, Anuar, Mokhtar, Gani, & Ullah Khan, 2015; Hu, 2015; Yeo & Crawford, 2015). Apache Hadoop as a tool of big data has been investigated (Saraladevi, Pazhaniraja, Paul, Basha, & Dhavachelvan, 2015; Yao, Tian, Li, Tian, Qian, & Li, 2015; Yu, Zhao, Wang, Wang, & Zhang, 2015). Apache HBase has been applied for data storage in big data platforms (Agrawal, Chakravorty, Rong, & Wlodarczyk, 2014; Chang, Tsai, Guo, & Chen, 2015; Huang, Wang, Zhu, Wang, & Yu, 2014; Lee & Zheng, 2015; Serrano, Han, & Stroulia, 2015; Wang, Cheng, Wu, Wu, & Teng, 2015). Apache Hive has been investigated for big data platforms (Chao, Li, Liang, Lu, & Xu, 2015; Luo, Liu, & Watfa, 2014; Wang, Bian, Chen, Wang, & Xu, 2014). Based on the research above, cloud computing platforms is a suitable environment for the migration and deployment of service oriented systems.

In terms of the reliability of service oriented systems in clouds, there has been a lot of research. The method of component ranking for applications in cloud computing platforms has been proposed, which can rank the services of applications according to their structures and the invocation frequencies (Zheng, Zhou, Lyu, & King, 2012). A framework for service oriented applications in clouds has been illustrated, which can identify the important components and improve their reliability with fault tolerance strategies (Zheng, Zhou, Lyu, & King, 2010). Software systems are described as graphs. After these algorithms calculate component ranking, the most significant services can be selected. However, sometimes a group of neighbouring components are important. These algorithms may consider the whole application instead of some small groups.

Some work based on Bayesian work has be investigated. A knowledge engineering framework based on a Bayesian network for the management of the services in distributed environments has been illustrated (Wang, Wang, Yang, Liu, Liu, & Zeng, 2013). Bayesian networks have been exploited for software defect prediction (Okutan & Yıldız, 2014). Research has been performed on Byzantine fault tolerance for applications (Zhao, 2007). A method based on Byzantine fault tolerance has been investigated for services with commutative operations (Chai & Zhao, 2014). A novel approach has been proposed to integrate Byzantine fault tolerance and Shamir’s method to detect failures in stored data in clouds (AlZain, Soh, & Pardede, 2013). An approach based on Byzantine fault tolerance strategies can also be utilised for unreliable processors (Hsieh & Chiang, 2011). The Byzantine fault tolerance method has been exploited in federated cloud computing platforms (Garraghan, Townend, & Xu, 2011). A Byzantine fault-tolerant framework has been designed and developed for service oriented applications (Zhao, 2009). An investigation of Byzantine fault tolerance approaches has been performed for MapReduce in big data platforms (Costa, Pasin, Bessani, & Correia, 2013). A fused data structure has been utilised for fault tolerance in distributed environments (Balasubramanian & Garg, 2013). However, execution time of algorithms based on Bayesian networks is a problem.

Some migration frameworks for applications into clouds have been proposed. A database has been placed intelligently in cloud computing platforms (Yu, Qiu, Reinwald, Zhi, Wang, & Wang, 2012). The services of applications can be placed in the federated hybrid clouds with the consideration of cost model (Altmann & Kashef, 2014). A new scheme for the migration of the cloud applications has been proposed in order to minimise resource consumption (Tziritas, Khan, Xu, Loukopoulos, & Lalis, 2013). A pattern based approach has been described for the migration of applications in clouds (Cai, Zhao, Wang, Yang, Qin, & Yin, 2015). A method for cost-effective migration to cloud computing platforms has been depicted (Huang, Yi, Song, Yang, & Zhang, 2014). The migration of a multi-tier application has been investigated (Liu & He, 2015). However, although these frameworks can compute the total resources, time and configuration cost, software reliability has not been considered during migration.

2.2.2 Literature review of the reliability of service oriented systems

For the reliability of applications, a lot of research has been proposed (Brun, Bang, Edwards, & Medvidovic, 2015; Jais, 2015; Manen, Brandt, Ekris, & Geurts, 2015; Sharma, Saket, & Sagar, 2015; Singh, Tripathi, & Vinod, 2015; Zhu, Zhang, & Pham, 2015). A roadmap of software reliability engineering has been proposed to describe the methods and the trends of fault tolerance strategies (Lyu, 2007). The Quality of Service (QoS) is an important feature for distributed applications. Additionally, QoS features of Web services in the real world have been investigated in order to provide data sets (Zheng, Zhang, & Lyu, 2014), which can be employed for the experiments in the Web service field. The ranking of the components and modules of applications can be performed according to usage relations (Inoue, Yokomori, Yamamoto, Matsushita, & Kusumoto, 2005). The page ranking approach has been exploited for large-scale applications (Brin &

Page, 1998). Therefore, they both need to identify significant software components.

A QoS aware fault tolerant middleware for dependable service composition has been proposed (Zheng & Lyu, 2008). A QoS aware fault tolerance strategy has been described, which enables a QoS system to dynamically adjust the fault tolerance approaches in order to achieve optimal reliability of the entire application (Zhu, Qin, & Qiu, 2011). These work apply fault tolerance strategies to the selected important components. The failures in the service of applications can be extracted from server logs and the reliability can be evaluated through the faults (Huynh & Miller, 2009). In these publications, it is clear that a service oriented system can be described as a network and the QoS features can be utilised to describe its performance.

Some models and patterns are designed to improve the reliability of service oriented systems. A fuzzy model has been depicted to measure software reliability, which can evolve procedures from the unreliable state to the reliable state of the applications (Kumar, Khatter, & Kalia, 2011). A model has been proposed for high-performance computing applications to resolve simultaneous failures (Thanakornworakij, Nassar, Leangsuksun, & Paun, 2013). A tool has been proposed so that the reliability of the applications can be conducted based on reliability patterns (Coppolino, Romano, & Vianello, 2011). The concurrent software applications with the consideration of software architecture are analysed for efficient reliability (El Kharboutly & Gokhale, 2014). The impact of the middleware service on the reliability of the systems has been investigated based on simulation analysis through the experiment on a Java application server (Huang, Wang, Liu, & Mei, 2011).

Some investigations have been performed for the prediction and analysis of reliability. A reliability model has been depicted for the reliability prediction of service oriented systems according to their architectures (Mirandola, Potena, Riccobene, & Scandurra, 2014). The work analyses and predicts software reliability in the abstract perspective. A case study was performed for an architectural reliability analysis of the applications (Rahmani, Azadmanesh, & Siy, 2014). A fault tolerance approach to enhance the reliability of service oriented applications based on business process execution language (BPEL) has been described (Wu, Xiong, Han, Huang, Hu, Gu, & Hang, 2013). However, the work may not suit to the situation when service oriented applications do not utilize BPEL.

Intelligent systems have been investigated for the reliability prediction of applications (Mohanty, Ravi, & Patra, 2013). A Bayesian network based model has been proposed for service oriented systems in terms of QoS assessment (Wang, Wang, Yang, Liu, Liu, & Zeng, 2013). The utilization of a Bayesian network may cost much execution time. Therefore, in practice, execution time may be a problem.

Some work takes software reliability in distributed environments into consideration. The research provides service generated big data and Data-as-a-Service (Zheng, Zhu, & Lyu, 2013). A method for SOA based applications has been proposed in order to improve their reliability (Delac, Silic, & Srbljic, 2015). An analysis of energy waste related to failures in the cloud computing platforms has been performed (Garraghan, Moreno, Townend, & Xu, 2014). A framework to predict the performance of service oriented systems has been proposed (Zhang, Zheng, & Lyu, 2014). An open architecture has been researched to design a system in distributed environments (Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994).

In service recommendations, QoS features are still important measure for software quality. The matrix factorisation has been exploited for the QoS prediction of Web services (Zheng, Ma, Lyu, & King, 2013). Parts of objects can be learnt via nonnegative matrix factorisation (Seung & Lee, 1999). Matrix factorisation as one of big data algorithms is widely utilized for recommendation.

In Web service recommendations, QoS features are also utilised (Zheng, Ma, Lyu, & King, 2011). The location is considered when the QoS based service recommendation is performed (Yu & Huang, 2014). The characteristics of non-functional requirements have been integrated to improve recommendation accuracy (Chen, Zheng, Liu, Huang, & Sun, 2013).

Fault tree analysis can be exploited to describe the faults of applications. Software fault trees are very useful for safety-focused applications (Cha & Yoo, 2011). Fault trees and unified modelling language (UML) state machine diagrams are utilised for safety analyses (Kim, Wong, Debroy, & Bae, 2010). The research focuses on efficient analysis via fault trees with voting gates (Xiang, Yanoo, Maeno, Tadano, Machida, Kobayashi, & Osaki, 2011). In fault tree analysis, defining gates is always considered to describe various states of services. Flexible analysis with fault trees is performed with component logic models (Forster & Schneider, 2010). The reliability of the dynamic fault tree has been evaluated via a tool combining FT and Monte Carlo Simulink (Manno & Chiaccho, 2011). Timing analysis has been performed via fault trees with the consideration of time dependencies and timed state-charts (Magott & Skrobanek, 2011). Fault detection, identification and recovery have been defined via dynamic decision networks and extended fault trees (Portinale & Codetta Raiteri, 2011). Although a lot of research has been investigated based fault trees, it is complex to represent all states of a distributed system.

2.2.3 Related work of Fault tolerance strategies

Some research takes the fault tolerance strategies into consideration in the design and development of systems. The hardware and software architectures have been defined and analysed in terms of fault tolerance (Laprie, Arlat, Beounes, & Kanoun, 1990). The design is performed to consider the fault tolerance feature in the protocol, SOAP 1.2 (Fang, Liang, Lin, & Lin, 2007). A fault tolerance mechanism can be integrated SOAP 1.2. However, for services implemented with REST, the mechanism may be not suitable. An architecture supporting fault tolerance for Web services has been investigated in order to resolve value, omission and stops (Santos, Lung, & Montez, 2005). The work in this thesis also considers fault tolerance to resolve stops. A middleware with Byzantine fault tolerance for Web service based applications has been designed and developed (Merideth, Iyengar, Mikalsen, Tai, Rouvellou, & Narasimhan, 2005). Byzantine fault tolerance needs more message transferring time for different stages. Additionally, once a fault is detected, a consistent protocol will be triggered and also needs more time. Therefore, in practice, Byzantine fault tolerance may be not suitable for distributed platforms. A framework for Web services with replications has been investigated (Salas, Sorrosal, Martínez, & Peris, 2006). This thesis also utilizes service replications to tolerate faults.

To improve software reliability, fundamental fault tolerance strategies can be involved. In this work, RB, NVP and Parallel are described and integrated into a multilevel redundancy allocation model. Approaches about service replication can be utilized and integrated in the model in this thesis.

2.2.4 Related work of redundancy allocation problems

An overview of reliability optimisation has been produced (Kuo & Prasad, 2000). An improved constraint method has been proposed to solve the redundancy allocation problem (Onishi, Kimura, James, & Nakagawa, 2007). In this thesis, software migration into cloud computing platforms is regarded RAPs. The work considers to maximize reliability and minimize cost and execution time under predefined constraints.

A two-stage discrete particle swarm optimisation method has been proposed to resolve the problem of multilevel redundancy allocation (Yeh, 2009). A hybrid particle swarm optimisation algorithm has been presented to solve the redundancy allocation problem (Beji, Jarboui, Eddaly, & Chabchoub, 2010). A hybrid particle swarm optimisation and local search have been exploited for series-parallel multi-state systems (Wang & Li, 2012). The artificial bee colony algorithm has been exploited for the solution of the redundancy allocation problem

(Yeh & Hsieh, 2011). In a parallel-series system, a novel dynamic programming approach has been illustrated (Yalaoui, Chatelet, & Chu, 2005). This thesis divides services into two categories: abstract and detailed services. In the view of the model, a service based application is a series-parallel system.

Some evolution algorithms have been investigated for series-parallel systems. A scaling approach has been utilised for reliability redundancy allocation (Ha & Kuo, 2005). The degraded ceiling algorithm and coupling ant colony have been performed for the redundancy allocation problem in series-parallel systems (Nahas, Nourelfath, & Ait-Kadi, 2007). A genetic algorithm has been proposed to solve multilevel redundancy allocation in series systems (Yun, Song, & Kim, 2007). A heuristic algorithm has been presented for multi-state series-parallel systems (Ramirez Marquez & Coit, 2004). A heuristic algorithm and a genetic algorithm have been utilised and some examples have been presented for series systems (Yun & Kim, 2004). A hierarchical genetic algorithm has been employed for the solution of multilevel redundancy allocation in series and series-parallel systems (Kumar, Izui, Yoshimura, & Nishiwaki, 2009). A hierarchical genetic algorithm can be utilized for series-parallel systems and some constraints can be predefined, which is suitable to the problem proposed in this thesis.

Three methods based on a differential evolution algorithm have been proposed (Beji, Jarboui, Siarry, & Chabchoub, 2012). A penalty guided bees search approach has been presented for series-parallel systems (Hsieh & Yeh, 2012). A three-phase scheme based on a corridor method has been proposed (Caserta & Voß, 2014). A combination of the dynamic programming and the depth-first search algorithms has been proposed (Ng & Sancho, 2001). Tabu search has been utilised for the solution of the redundancy allocation problem efficiently (Kulturel-Konak, Smith, & Coit, 2003). An ant colony optimisation algorithm has been presented for the redundancy allocation problem (Liang & Smith, 2004). A max-min approach has been presented for series-parallel systems (Ramirez Marquez, Coit, & Konak, 2004). A theoretical condition and an alternative method based on an approximated function have been proposed for a series-parallel system (Yalaoui, Chu, & Châtelet, 2005).

A tree heuristic has been proposed for the redundancy allocation problem (Ha & Kuo, 2006a). An improved realisation has been presented for nonconvex nonlinear programming problems (Ha & Kuo, 2006b). Simulated annealing algorithms have been presented for reliability redundancy optimisation (Kim, Bae, & Park, 2006). A multi-agent ant system has been described for a multi-state power system (Bendjeghaba & Ouahdi, 2008). An optimisation model has been presented for a multi-state series-parallel system (Tian, Zuo, & Huang, 2008). A two-phase heuristic approach has been proposed for performance evaluation (Kumar, Chaturvedi, & Pahuja, 2009). A fuzzy random parallel-series system has been analysed via a redundancy allocation model (Wang & Watada, 2009). A chaotic differential evolution method has been proposed for reliability redundancy optimisation (Coelho, 2009). A method has been presented to solve highly constrained redundancy optimisation problems in binary complex systems (Agarwal, Aggarwal, & Sharma, 2010). A redundancy allocation method has been utilised for reliable storage in a distributed environment (Xu, Lin, Wang, Liu, Shi, & Zhang, 2012). A discrete-binary transformation has been illustrated for the reliability redundancy allocation problem (Caserta & Voß, 2015).

A hierarchical genetic algorithm has been exploited for multilevel redundancy allocation optimisation (Kumar, Izui, Masataka, & Nishiwaki, 2008). Multilevel redundancy allocation has been resolved via a memetic algorithm (Wang, Tang, & Yao, 2010). A hybrid genetic algorithm with a new encoding mechanism based on two dimensional arrays has been proposed for multilevel redundancy allocation (He, Wu, Xu, Wen, & Jiang, 2013). A bacterial-inspired evolutionary algorithm has been exploited for multilevel reliability systems (Hsieh, 2014). Therefore, the multilevel redundancy allocation model can describe the structure of service oriented systems under constraints.

In summary, there may be no research on the utilisation of the multilevel redundancy allocation model for service oriented systems. However, the model can describe the structure of the systems and redundant modules can be assigned to ones at different levels. Additionally, few studies have been performed on the migration of service oriented systems to clouds with constraints. However, this may be a possible and significant challenge, for the consideration of constraints may reduce companies' expenses for the migration of service oriented systems in clouds.

2.3 Fault tolerance strategies for service oriented systems

Fault tolerance strategies are commonly utilised for significant and critical systems. For example, airplane flight control systems and nuclear power station management systems adopt fault tolerance strategies to improve their reliability (Qiu, Zheng, Wang, Yang, & Lyu, 2014). These strategies always have diverse requirements for the implementation of services, in terms of various programming languages and technologies. The possible reasons are as follows.

Design methods: For an identical functional requirement, various companies or organisations may have different design methods and plans for its implementation, which tends to lead to different reliability (Zheng & Lyu, 2010). If a service is equipped with fault tolerance strategies, and crashes or fails to execute requests from clients because of a bug, redundant services with the same functional requirement from different companies may replace the broken service and continue working. Therefore, for fault tolerance approaches, diversity of service resources from different organisations or designers is required. They have the same specifications, different design and implementation. However, their qualities cannot be guaranteed.

Integration approach: The Web service technique changes the integration approach of traditional software. Different services communicate with each other via the SOAP protocol, which do not need to be on the same server. This allows services to integrate with ease. The new integration approach can lead to easy maintenance of service oriented systems. If one service in the system has bugs or its reliability is not sufficient, an existing similar service with identical functions is integrated to replace the original one. This integration approach makes the utilisation of the fault tolerance strategies easier for service oriented systems. Additionally, the communication of all the services in the system is based on the SOAP protocol, which adopts the HTTP protocol to exchange data and messages. For almost all network facilities, the HTTP protocol can work well without any change of their configuration, which may be another widely accepted reason for Web service technology.

Programming languages: The Web service technique allows services to be implemented with various programming languages, which causes different reliability for a service. Fault tolerance strategies need a diversity of services. Ensuring the diversity of services comes from either the development involvement of varied organisations or the utilisation of various programming languages.

In the Web service field, the fault tolerance strategies recovery block (RB), N-version programming (NVP), and Parallel are commonly utilised for the reliability of distributed systems (Zhang, Zheng, & Lyu, 2011; Zheng & Lyu, 2010; Zheng, Ma, Lyu, & King, 2013; Zheng, Zhou, Lyu, & King, 2012). In this section, the quantitative and qualitative analysis of these fault tolerance strategies are described in terms of the failure rate f, the response time t and the required resources r.

2.3.1 Recovery block (RB) (Bonvin, Papaioannou, & Aberer, 2010)

This section describes recovery block (RB), which will be described in terms of system structure and quantitative and qualitative analysis. The first design and implementation of RB was in the early 1970s at Newcastle, which stemmed from a discussion about software reliability. Error-free programs and applications are required in the real world. To tackle the challenge of the creation of error-free systems, one method is that a better testing approach is needed to check for bugs or residual faults. Another approach to achieve the same task is to add a fault tolerance strategy to hide residual faults.

Some research shows that one of the most significant reasons for application failures is residual faults (Randell & Xu, 1995). Service oriented applications are becoming more and more complex in terms of software structure and source codes. This has more potential to cause residual software faults. RB is designed to utilise redundancy allocation to solve this type of problem in order to hide the occurrence of faults.

2.3.1.1 System structure

In this section, the system structure of RB is described. Before the description of the structure, an idealised module with RB is depicted. The idealised module is a sample of a component with RB and almost all fault tolerance strategies are proposed based on this module.

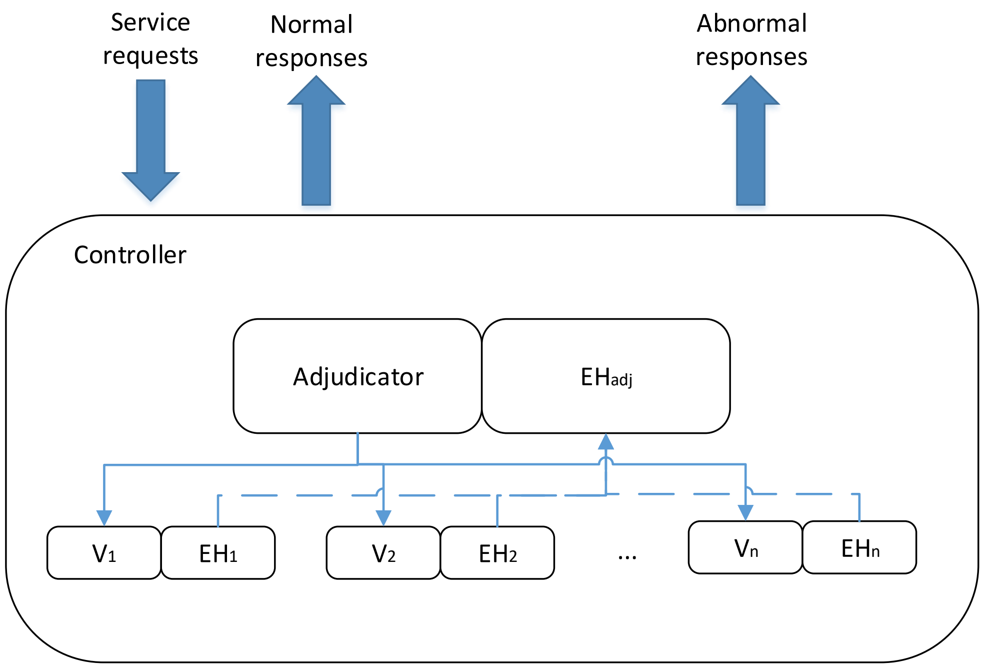


Figure 2.1: Idealised module with RB (Randell & Xu, 1995)

Figure 2.1 shows the idealised module with fault tolerance strategies. The aim of the module is to explain that the reliability of a software component can be improved with redundant units. From the figure, the component includes two parts: one is the controller and the other is the exception handler (EH). Specifically, the controller is the manager of all the submodules in the idealised module. A module can complete the detailed requirements of a request. The idealised module comprises a lot of submodules, which are called variants and they follow an identical specification. Namely, all the variants in an idealised module complete an identical task. Additionally, each variant has an exception handler. From the name “exception handlers”, it is clear that they are proposed to handle exceptions and errors. Since these submodules within an idealised module are able to cause errors or exceptions, an exception handler is required for each variant. The idealised module is utilised to deal with requests from end users or an application with strategies. Therefore, in the figure, a request can be sent to the idealised module. Thereafter, the module can respond with two types of results. One type of result is the normal response, which means that the idealised module has successfully completed the request. Note that, in this case, this does not mean that there are no exceptions. For example, perhaps one or more exceptions is performed, but the correct response still arises and is sent back to the requester. When the adjudicator cannot decide on the final response or all the variants in the idealised module cannot complete the request, an abnormal exception would be sent back to the requester. All fault tolerance strategies focus on the design of interactions between these submodules and communication between different roles in an idealised module.

The steps of the idealised module handling a request in Figure 2.1 are first that a requester sends a request to the module. When the module receives the request, it will send the request to the first variant or some of the variants. The detailed situation about the assignment of the request depends on specific fault tolerance strategies. If the request can be successfully completed, the module would send back the final response to the requester. Otherwise, if some exceptions or errors arise, the idealised module would deal with them. If they can be solved, the final and correct response is still returned to the requester and the requester would not realise the detailed processing of the module. However, if they cannot be solved, an abnormal response would be returned.

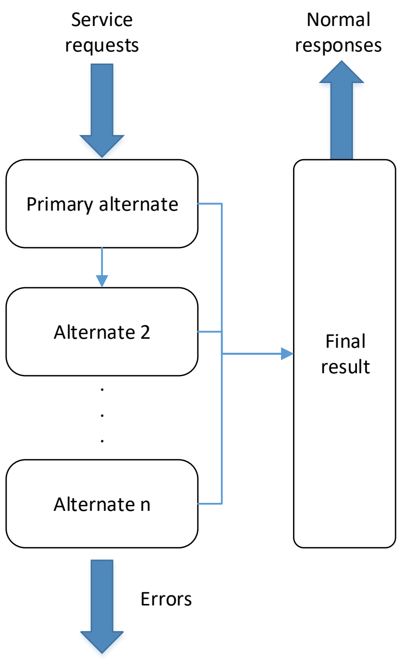


Figure 2.2: The execution steps of RB

Figure 2.2, Figure 2.3 and Figure 2.4 show the execution steps of RB, which needs to include a lot of functionally equivalent units, which are called alternates. All the alternates need to follow identical specifications, which denote that they can complete the same requirement. It is apparent that all the alternates are sequential. The responses may be errors or the final correct results.

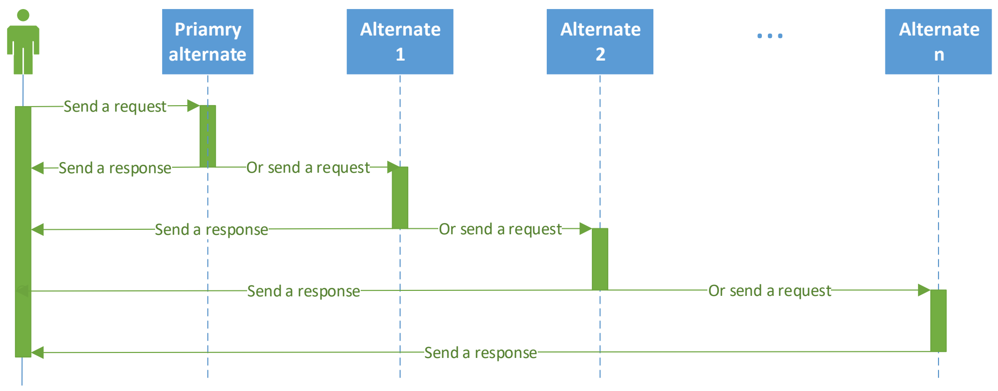


Figure 2.3: The sequence diagram of the execution steps of RB

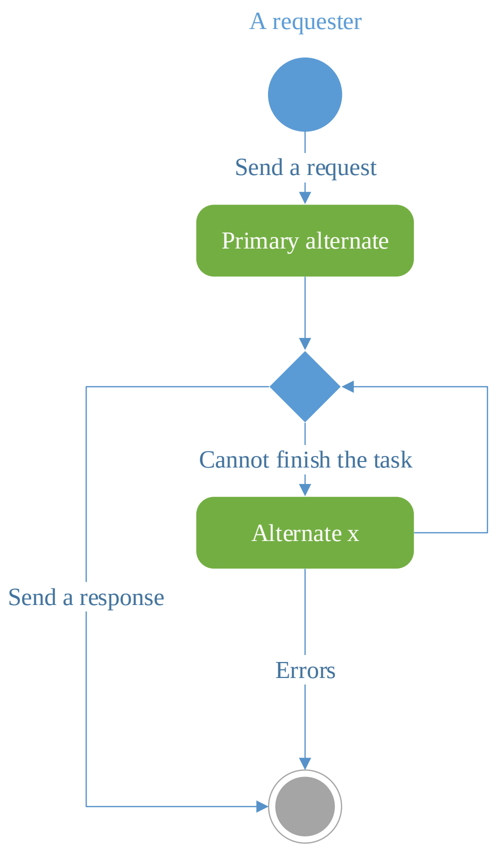


Figure 2.4: The UML activity diagram of the execution steps of RB

```
Input:      A request
Output:     A response or errors
----------------------------------------------
            The main execution steps of RB
Step 1      If the primary alternate is OK
Step 2          Process the request
                Return a response
Step 3      else if the alternate 2 is OK
Step 4      .
            .
            .
Step 5      else if the alternate n is OK
Step 6          Process the request
                Return a response
Step 7      else
Step 8          Return errors
```

Figure 2.5: The programming description of the execution steps of RB

Figure 2.5 illustrates the programming description of the execution time of the RB. In terms of the utilisation of RB in the service field, each alternate can be regarded as a service. All the services in the strategy are developed and implemented based on an identical specification. It is obvious that RB needs more than one service to be involved to provide a fault tolerance ability.

2.3.1.2 Quantitative and qualitative analysis (Zheng & Lyu, 2015)

All the functionally equivalent components in a module are sequential. If a component crashes or fails to respond to the requester, the next component in order would receive the request, execute it and send back responses.

$$f = \prod_{i=1}^{n} f_i \tag{2.1}$$

This formula is to calculate the failure rate of the module with RB. In the formula, f_i is the failure rate of the ith component. The reliability of the module is computed as follows,

$$R = 1 - \prod_{i=1}^{n} f_i \tag{2.2}$$

The following formula is employed to calculate the execution time that RB utilises for the processing of a request. The time is related with failure rate. The formula means that the previous $i-1$ functionally equivalent components fail to execute the request and the ith component successfully handles it.

$$t = \sum_{i=1}^{n} t_i \prod_{k=1}^{i-1} f_i \tag{2.3}$$

$$r = \sum_{i=1}^{n} r_i \prod_{k=1}^{i-1} f_i \tag{2.4}$$

This formula shows the number of resources when the number of components is n. Whether the ith component is invoked depends on the operation situation of the previous $(i-1)$ components.

2.3.2 N-version programming (NVP) (Avizienis, 1995)

N-version programming (NVP) is a fault tolerance strategy which tends to generate a final response with a decision algorithm according to the responses delivered by two or more functionally equivalent components. The strategy needs these responses to be produced independently in order to minimise the relationship between any two components. In this case, an exception from a component possibly does not influence others and the final result is determined by the majority votes from all these components. Therefore, the number of the components in a module with NVP needs to be more than 2, namely, $n > 2$. All the components in the module are designed and developed based on an identical specification. When a request is sent to the module, all the components would receive it. Since these components have the same initial state, they theoretically return the same responses if no errors arise. If there are faults or exceptions in some of the components, the final results would be not identical. Therefore, in this case, a comparison of all the results would be performed and the final result with the majority votes would be sent back to the requester.

Whether the scheme can be successfully employed for an application depends on whether the faults are produced by *n* versions of programs (Avizienis, 1995). Therefore, the development and implementation of independence can reduce the possibility of the concurrent occurrence of the faults from *n* versions of programs. Therefore, for a service oriented application, the services based on the same specification need to be developed by different individuals or organisations. This can improve design and implementation diversity.

2.3.2.1 System structure

In this section, the system structure of NVP is described. A simple and generic system structure is depicted as follows.

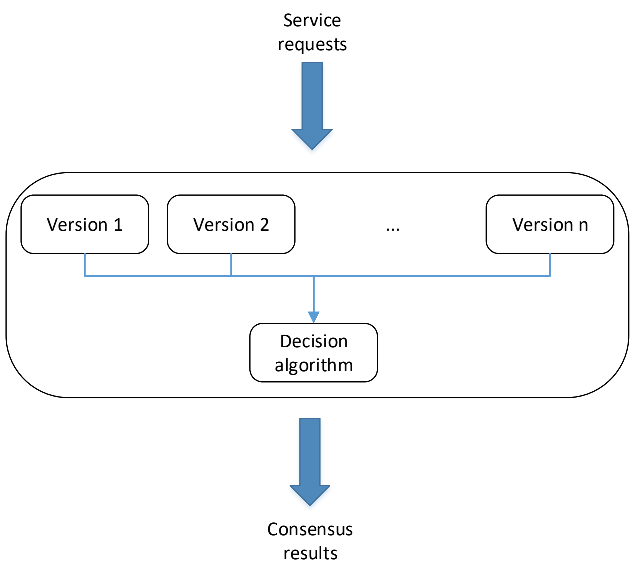


Figure 2.6: The system structure of NVP

Figure 2.6 shows the system structure of NVP. To equip NVP to a component, the $n-1$ components need to be developed based on an identical specification. All the components can perform the same business logic and they are required to be designed and developed separately and individually, which tends to reduce the possibilities of the occurrence of faults at the same time. In the figure, it is apparent that there are two parts to the module. One is the *n* versions of programs and the other one is the decision algorithm. The decision algorithm is adopted to compare the results produced by these *n* versions of components. The final consensus result would be sent back to the requester.

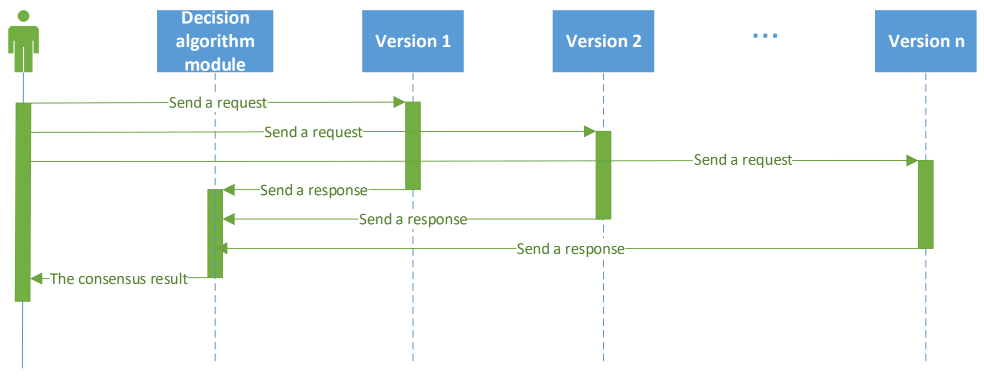


Figure 2.7: The sequence diagram of the execution steps of NVP

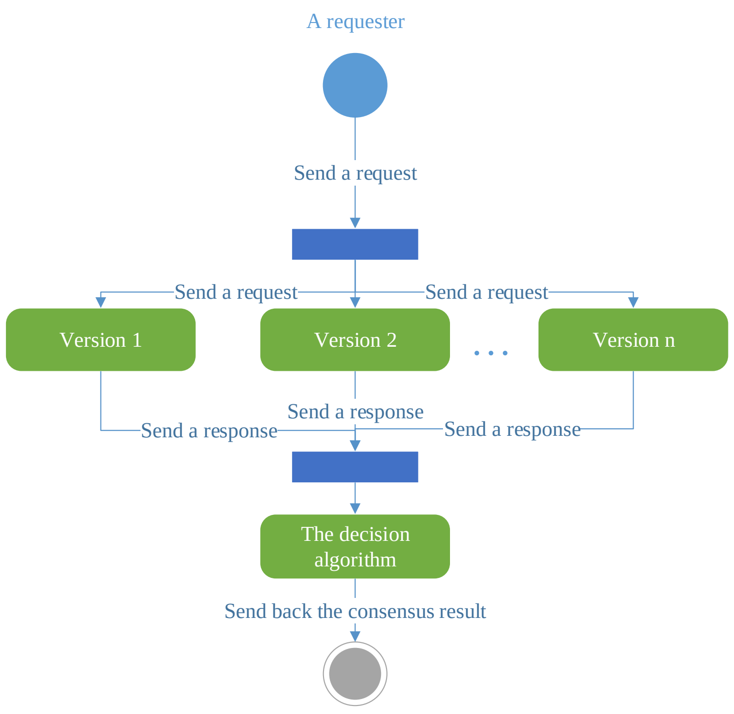


Figure 2.8: The activity diagram of the execution steps of NVP

Input:	A request
Output:	The consensus response
	The main execution steps of NVP
Step 1	All the versions of the components receive the request
Step 2	Execute the request
Step 3	Send the responses to the decision algorithm
Step 4	Compare the results
Step 5	Send back the consensus response to the requester with the majority votes

Figure 2.9: The programming description of the execution steps of NVP

Figure 2.7, Figure 2.8 and Figure 2.9 illustrate the sequence diagram, the activity diagram and the programming description of the execution steps of NVP. When a request is sent to the module with NVP, all the components, namely, *n* versions of programs, can receive the request and execute it. After they complete the execution of the request, they would send their responses to the decision algorithm which is responsible for the comparison of the results. All the arrived responses would be compared. The response with the majority of votes would be set as the final consensus result. Once the consensus result is determined, it would be transmitted to the requester.

2.3.2.2 Quantitative and qualitative analysis

All the components in the module with NVP are in parallel. When a requester sends a request to the module, the request would be transmitted to all the components. Subsequently, each component handles the request and sends a response back to the decision manager. The final response to the requester is determined by the majority of votes of all the components in the module.

$$f = 1 - \sum_{i=\frac{n+1}{2}}^{n} F(i) \tag{2.5}$$

This formula is the failure rate of the module with NVP. $F(i)$ represents the failure rate of the ith version of the component, since all the components may be designed and developed by different individuals or organisations. The design and implementation variety cause different failure rates of all the components. $F(i)$ is based on the actual situation of the module. $i = (n+1)/2$ denotes the number, which is more than half the amount of components. The final consensus response is the result with the majority of votes.

$$t = maxt_i \tag{2.6}$$

This formula shows the execution time of NVP. Since all the components in the module are in parallel and the final response is determined by the majority of votes, the execution time is the value of the last component completing the request processing.

$$r = \sum_{i=1}^{n} r_i \tag{2.7}$$

This formula describes the total resources required by NVP. The total resources of NVP should be the sum of the resources of each component in the scheme.

2.3.3 Parallel (Qiu, Zheng, Wang, Yang, & Lyu, 2014)

In this section, the fault tolerance Parallel is described. Parallel is similar to NVP. The main difference is that the first arrival response from a component in a module to the decision manager is regarded as the final result. Therefore, the system structure can be seen in Figure 2.6, Figure 2.7, Figure 2.8 and Figure 2.9.

2.3.3.1 Quantitative and qualitative analysis

Parallel does not need to decide the final result with the majority of votes. All the components in the module need to execute the request. Therefore, the failure rate is shown as follows.

$$f = \prod_{i=1}^{n} f_i \qquad (2.8)$$

Since the final result is determined by the first arrived response from one of the components, the execution time of the entire module is the least time.

$$t = min\ \ t_i \qquad (2.9)$$

Parallel still needs to be equipped with n redundant components. Therefore, the resource cost is represented as follows.

$$r = \sum_{i=1}^{n} r_i \qquad (2.10)$$

2.3.4 The comparison of the fault tolerance schemes

In this section, a comparison of the fault tolerance strategies is performed. The failure rate, the execution time and resource costs are listed. RB, NVP and Parallel are compared in terms of these three aspects. Table 2.1 shows the summary of the strategies.

Table 2.1: The summary of the fault tolerance strategies

Strategies	Failure rate	Execution time	Resource cost	Reference
RB	$f = \prod_{i=1}^{n} f_i$	$t = \sum_{i=1}^{n} t_i \prod_{k=1}^{i-1} f_i$	$t = \sum_{i=1}^{n} t_i \prod_{k=1}^{i-1} f_i$	(Bonvin, Papaioannou, & Aberer, 2010)
NVP	$f = 1 - \sum_{i=(n+1)/2}^{n} F(i)$	$t = maxt_i$	$r = \sum_{i=1}^{n} r_i$	(Avizienis, 1995)
Parallel	$f = \prod_{i=1}^{n} f_i$	$t = min\ \ t_i$	$r = \sum_{i=1}^{n} r_i$	(Qiu, Zheng, Wang, Yang, & Lyu, 2014)

In Table 2.1, it is apparent that the strategies RB and Parallel have an identical failure rate when they have the same number of components. NVP needs to take advantage of the majority of votes from the components in the module in order to decide the final consensus response. Hence, the correctness of the $(n+1)/2$ components needs to be guaranteed.

The execution time of RB is based on the number of the invoked components. The execution time of NVP is the maximum time of all the returned responses. However, Parallel needs the least time of all the components.

Apart from the execution time and reliability, another difference of these strategies is the resource cost. For RB, not all the components are possibly invoked. Once one component successfully processes the request, the remaining components would not be invoked. For NVP and Parallel, all the components are needed.

In summary, the selection of the different strategies depends on the specific requirements from end users. If they want an efficient response from a module, RB or Parallel may be the best choice.

2.4 Related technologies

In this section, related technologies utilised in this work are described in terms of their concepts and tools.

2.4.1 Web service

The Web service technique provides a new communication method for software functionalities deployed on distributed computers over a network. The utilisation of a Web service offers an interoperable machine-to-machine interaction between modules in systems or between different systems, which provides a method for integration of service oriented systems with extensible markup language (XML), Web service description language (WSDL) and universal description discovery and integration (UDDI).

2.4.1.1 Concepts

The Web service technique involves XML, WSDL and UDDI concepts. These concepts are described as follows,

XML: Extensible markup language (XML) is a markup language. The language defines a set of rules and specifications, which makes the encoded documents human-readable and machine-readable (Karunamurthy, Khendek, & Glitho, 2012). XML is commonly used in the Web service technique, because XML can hide the detailed implementation of distributed services developed in various programming languages.

WSDL: Web service description language (WSDL), based on XML, is an interface definition language. WSDL is utilised to depict the functionalities of Web services, which comprise the number and structure of input and output parameters (Wu, Chen, Zheng, Lyu, & Wu, 2014).

UDDI: Universal description, discovery and integration (UDDI) is an XML based registry, which allows the publication of Web services from various organisations and individuals in the world. The registered services can be found by other organisations or individuals and accessed through UDDI, which records the detailed network access address of the Web services.

SOA: A service oriented architecture (SOA) enables applications from different places in the world to be integrated without the consideration of detailed implementation of programming languages. SOA is a software architectural pattern for software architecture design, which enables services of distributed applications to be accessed by other software systems over a network (Delac, Silic, & Srbljic, 2015).

2.4.1.2 Major Classes of Web services

There are two major classes of Web services in terms of implementation. One class is simple object access protocol (SOAP) and the other one is representational state transfer (REST).

SOAP: Simple object access protocol (SOAP) includes rules and specifications for the format of exchanging information between Web services. SOAP uses XML rules to organise information and is based on other protocols, such as hypertext transfer protocol (HTTP) and simple mail transfer protocol (SMTP) (Delac, Silic, & Srbljic, 2015).

REST: Representational state transfer as the other implementation of Web service technique enables developers to perform the invocation of Web services via specified uniform resource identifiers (URIs). REST is easy to be understood and can be easily supported by servers or clients, although they can support the hypertext transfer protocol (HTTP) or HTTP secure (HTTPS). Web services can be accessed through the HTTP GET, POST, PUT, DELETE, HEAD, OPTIONS methods. Therefore, developers can implement applications easily based on the existing IT infrastructure.

2.4.1.3 Communication

In this section, communication between UDDI, service consumer and service provider are shown and described.

Figure 2.10 illustrates the relationship between UDDI, service consumers, service providers, SOAP and WSDL. The detailed explanation is shown as follows,

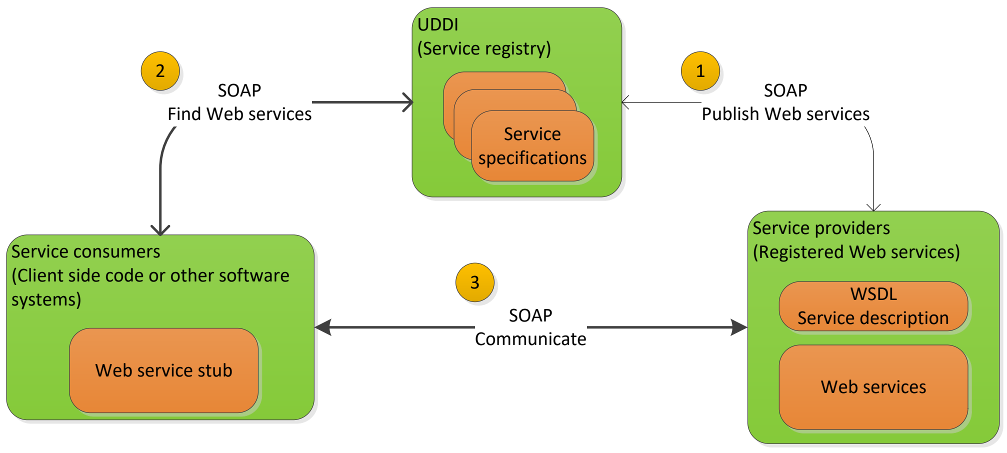


Figure 2.10: The relationship between UDDI, service consumers and service providers

Service providers: They are companies, organisations or individuals, which provide Web services to the world. Additionally, they may be from different places in the world and utilise varied techniques to implement services.

Service consumers: They send requests to the services and deal with the corresponding responses.

The main workflow in Figure 2.10 is that service providers firstly publish their developed Web services in UDDI, namely, registration of their Web services in the service registry. Meanwhile, these service providers also offer the specifications of their services. Service consumers can find services according to their requirements in UDDI. If the services are confirmed, the actual address of the services can be received and the requests can be sent.

For a REST web service, a request can be sent by service consumers to a URI. The response may be in XML, JSON or other defined formats.

2.4.1.4 Tools

In this section, the tools for manipulating Web services are described. In this work, the Java programming language is employed to develop Web services. A detailed description of the tools is as follows.

Apache Axis2: A Web service engine for Java and C programming languages. Web services developed by Apache Axis2 use SOAP and are exposed to the outside of service oriented systems via WSDL.

Eclipse: Eclipse is a free integrated development environment (IDE). When Eclipse is downloaded from its official website, there are already some basic plugins integrated in the IDE. However, developers can expand the functions of the IDE through the addition of new plugins. Eclipse is developed mostly in Java programming language and it is mainly an environment for Java. In this work, Apache Axis2 plugins are utilised and integrated in Eclipse to generate Web services and stub source code.

Service archive generator and code generator wizard (Apache Axis2 Eclipse plugins): These two plugins can be downloaded from their official Website and integrated with Eclipse, which can work well with the Eclipse EE edition. The service archive generator can generate Java Web services and WSDL files according to their source code. The code generator wizard uses WSDL files to build stub source code, which is used to access Web services.

Web service call composer: Stylus Studio includes a Web service call composer, which can locate and call Web service methods from Stylus Studio. Additionally, the studio can generate SOAP envelopes and supports the technologies (e.g., WSDL, SOAP, UDDI, etc.).

2.4.2 Cloud computing

2.4.2.1 Concepts

Cloud computing as a computing model integrates computing, networking and storage resources provided by ubiquitous computers as a resource tool for users. Virtual machines (VMs) share these resources and users work in VMs. Specifically, if a user finishes their work in the VM and want to quit the cloud computing platform, the computing, networking and storage resources would be released for other users. The method maximises the utilisation of these resources and users do not need to consider the specific configuration modifications. End users can use these resources in third-party data centres, namely, in cloud computing platforms. When users work in a VM, their working manner is similar to their local computers. Cloud computing is also an available and convenient choice for small scale companies to deploy their systems and develop their applications in VMs. For administrators of a whole cloud computing platform, they can efficiently manage the platform. Specifically, once the cloud computing platform is set up, almost all the computing, networking and storage configurations can be performed in a single computer, which specifically comprises the management of VMs, resources pools, networking access, data centres and storage.

2.4.2.2 Theoretical models

According to the different services provided by clouds, cloud computing platforms are divided into the following types. They are employed for varied purposes.

Infrastructure as a service (IaaS): IaaS provides physical resources where cloud computing managers set up operating systems for end users. The physical resources are computing, network and storage hardware.

Platform as a service (PaaS): Platform vendors provide PaaS to other organisations and individuals. Platform vendors set up development toolkits, operating systems, programming language execution environments, databases, Web servers, etc. Users can utilise these resources to develop applications and deploy them in VMs.

Software as a service (SaaS): Service oriented systems can be deployed in VMs and then the services can be accessed by users outside of the cloud computing platform.

2.4.2.3 Structures

Figure 2.11 shows the structure and the relationship between cloud computing platforms and users. The cloud computing platform can be divided into three layers. The infrastructure layer includes servers, network devices (such as routers, switches, etc.), cables, etc., which mainly offer the hardware foundation to the higher layers (the working environments layer and the applications layer).

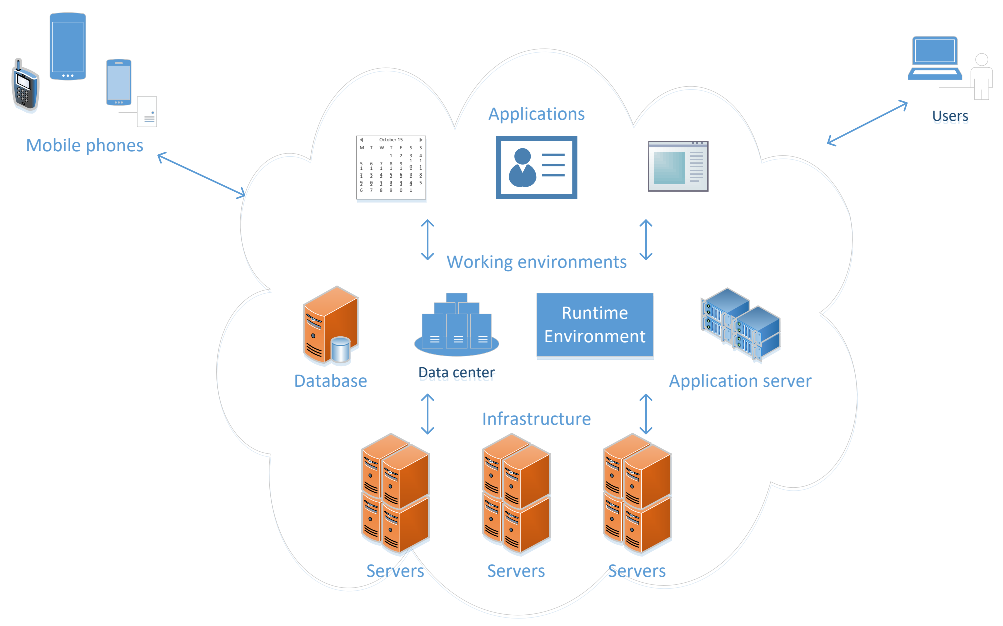
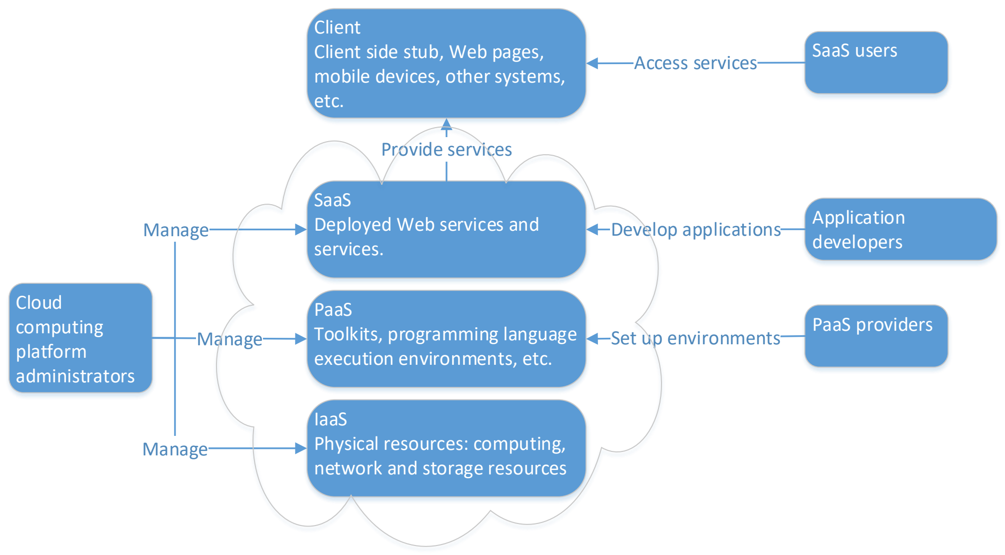


Figure 2.11: The relationship between users and cloud computing platforms (Rajaraman, 2014)

The working environments layer supports VMs where database management systems, data centres, runtime environments and application servers (such as IIS, Apache Tomcat, etc.) are provided. Cloud users log onto the VMs in the cloud to utilise these resources. Users not only do their work within a VM as their local computer, but develop and deploy their applications as well. The applications layer includes installed applications for users, which may be Microsoft Office, integrated development environments (such as Eclipse, Microsoft Visual Studio), etc.

2.4.2.4 Communications

In this section, the communications between IaaS, PaaS, SaaS and different types of users are described. Users include clouds administers, PaaS providers, application developers and SaaS end users.



Client
Client side stub, Web pages, mobile devices, other systems, etc.
Access services
SaaS users
Provide services
Manage
SaaS
Deployed Web services and services.
Develop applications
Application developers
Cloud computing platform administrators
Manage
PaaS
Toolkits, programming language execution environments, etc.
Set up environments
PaaS providers
Manage
IaaS
Physical resources: computing, network and storage resources

Figure 2.12: The relationship between SaaS, PaaS and IaaS (Rajaraman, 2014)

Figure 2.12 shows the relationship between IaaS, PaaS, SaaS, the clients and roles in clouds. IaaS provides physical computing, network and storage resources to PaaS, which offers toolkits, programming language execution environments, etc. to SaaS. The deployed services are at the SaaS layer and are accessed by clients, which include client side stub, Web pages, mobile devices, other systems, etc. Cloud computing platform administrators manage the entire platform (IaaS, PaaS and SaaS). PaaS providers are responsible for the configuration of PaaS platforms. Application developers design, develop and deploy systems at the SaaS layer. SaaS users can access services supported by the SaaS layer.

Chapter 3 Application Migration and Deployment Framework

In this chapter, an application migration and deployment framework is proposed, which utilise a genetic algorithm to compute the optimal migration plan under the cost and execution time constraints. The framework can guide designers or developers to migrate and deploy their service oriented systems on a cloud computing platform. The detailed method of how the framework should work will be depicted in the following. The framework as an interface of the genetic algorithm is responsible for the data collection and the display of migration plans.

3.1 Introduction

In this section, the proposed migration and deployment framework in this work is described. The existing migration and deployment frameworks of service oriented systems in cloud computing platforms are depicted and possible insufficient features are proposed.

Figure 3.1 shows an example of a service oriented system, which is based on components and there are some relationships between these m modules. Module 1 is at the top level of the system. In this work, this type of module describes an abstract function of an application, or an entire application. An application possibly comprises a lot of modules, which represent varied logical functions and all of them make up the entire system. From Figure 3.1, it is clear that this system includes m modules and p database management systems.

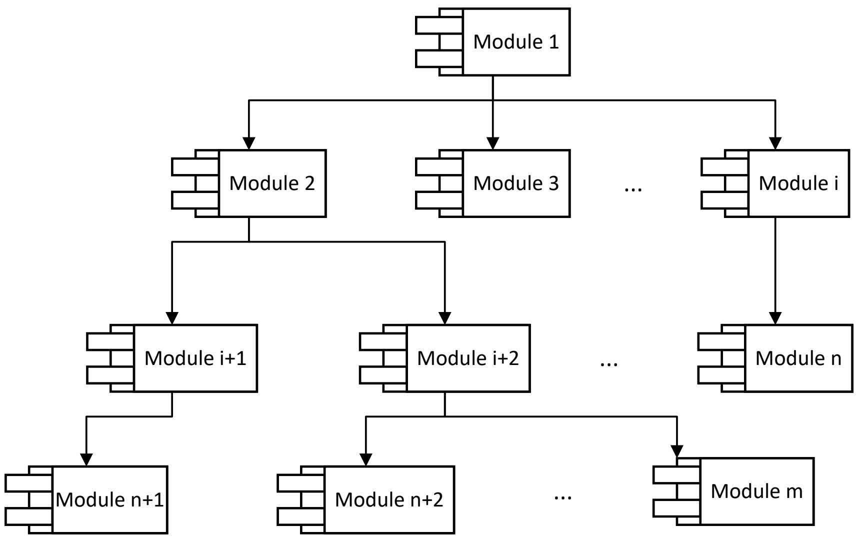


Figure 3.1: An example of a service oriented system

Additionally, from the figure, multiple levels are shown. There are four total levels for data processing and one level for data storage. The modules at the high levels can invoke ones at the low levels. For example, component 1 can send a request to modules 2 and 3. Moreover, these modules can also send requests to the modules at the lower levels. The modules at the lowest level are the components of an application, which means that they actually process business logic and may be implemented in different programming languages. Therefore, an application or a function can be described as a tree as the example in the figure.

In terms of data storage, an application or a system may take advantage of different types of database management systems (e.g., relational database management systems, object oriented database management systems, XML database management systems, Hadoop distributed file system, HDFS, etc.). When a service oriented system is migrated and deployed in a cloud, the migration of the data storage needs to be considered as well. The corresponding fault tolerant approaches need to be employed for this part of work and resource costs for the placement of the database management systems need to be planned in advance. However, in this work, only the problem of service reliability is taken into consideration.

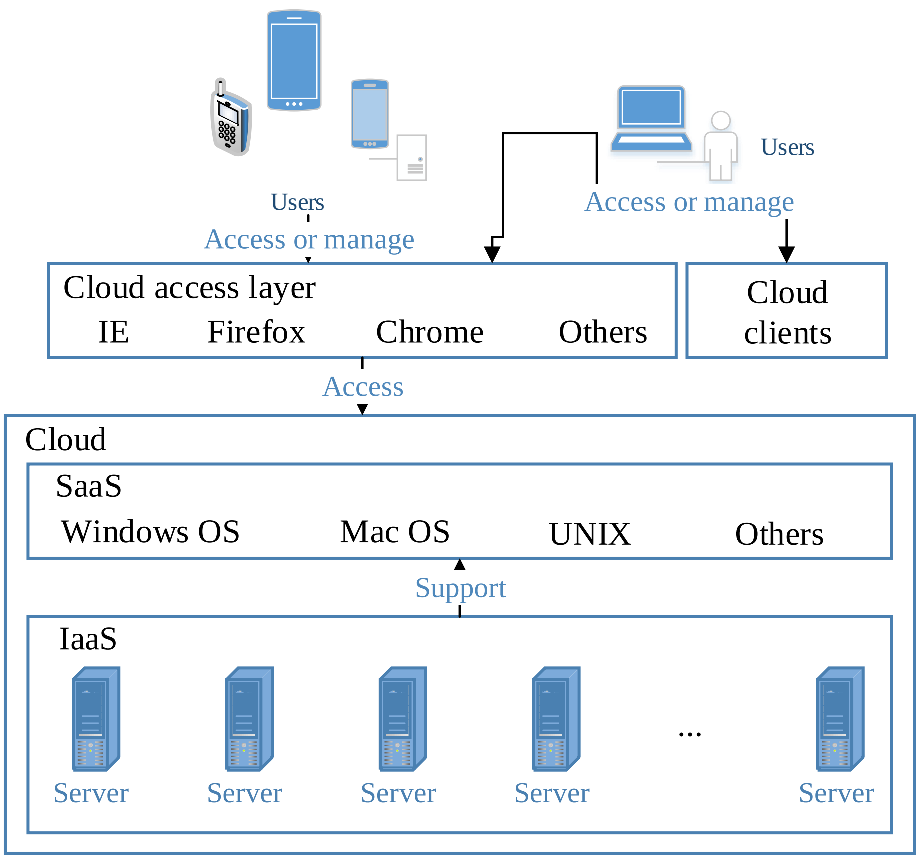


Figure 3.2: An example of SaaS and IaaS in a cloud computing platform

Figure 3.2 describes an example of SaaS and IaaS in a cloud computing platform. The reason the example is depicted here is that the structure of a cloud computing platform should be shown and then the method to migrate and deploy service oriented systems can be explained clearly. In the figure, there are four layers which are the IaaS layer, the SaaS layer, the cloud access layer and the user layer.

The IaaS layer mainly includes various types of network facilities, servers and other types of hardware. Servers are employed to provide computing ability for the entire cloud computing platform. The distributed servers in the cloud can work together to offer powerful computing ability. Network facilities are responsible for the connection of different servers and other network facilities, such as switches, routers, bridges, etc. All the servers, computers and workstations can theoretically communicate with each other via these network facilities. Additionally, cloud computing platforms always offer a network interface. It means that all data between the inside and outside of a cloud computing platform are exchanged via the unified port.

The SaaS layer mainly comprises different types of operation systems. In the figure, Windows operating systems, Mac operating systems and UNIX are the examples shown. A cloud computing platform can comprise more than one operating system for end users. Their types and number mainly depend on the ability that a cloud computing platform can support. Note that different cloud computing providers may provide various operating systems on VMs. In these operating systems, there are some applications which are set up for users to deploy their services. For example, Apache Tomcat and Apache Axis 2 can be deployed in these operating systems to support the execution of applications developed in Java programming language. For data storage, corresponding database management systems are required to be built on these VMs. The commonly used systems are MySQL, SQL Server, Oracle, Apache Hadoop, etc (Song, Guo, Wang, Zhang, Yu, & Pierson, 2015).

The cloud access layer is the bridge between end users and cloud computing platforms, and mainly consists of two types of tools. One is Web browsers and the other one is specific cloud clients. The cloud clients from different clouds are not always compatible with others.

The tools at the user layer comprise mobile devices, laptops, etc. Furthermore, end users can access a cloud computing platform via wireless and wired networks.

Based on the above description, the services of a migrated system are distributed in VMs at the SaaS layer. Users can utilise various devices to access them.

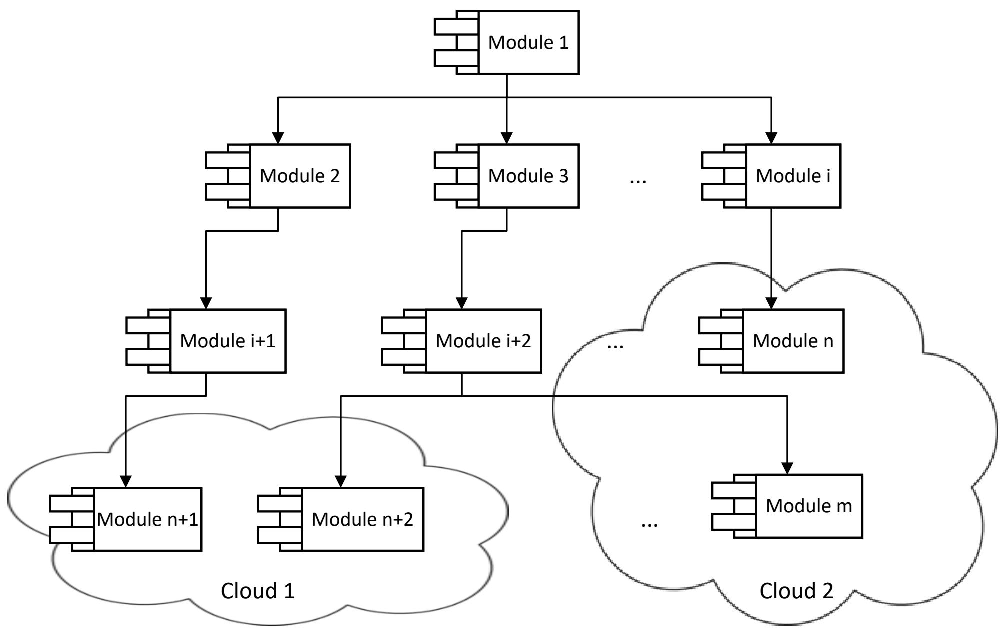


Figure 3.3: An example of the deployment of a migrated service oriented system in clouds

Figure 3.3 shows an example of the deployment of a migrated service oriented system shown in Figure 3.1 in clouds. As explained and described above, the modules from $n+1$ to m are the services of the system, while other modules describe abstract business logic. In Figure 3.3, there are two cloud computing platforms, which are named cloud 1 and 2. In this work, the migration and deployment framework can be utilised for service oriented systems in multiple clouds.

There are already frameworks (Qiu, Zheng, Wang, Yang, & Lyu, 2014; Zheng & Lyu, 2010, 2015; Zheng, Zhou, Lyu, & King, 2010) for the migration and deployment of service oriented systems, which guide designers to migrate the systems in a cloud computing platform with the consideration of some QoS features (such as availability, price, popularity, data size, success probability, response time, overall success probability, overall response time, etc.).

The similarity of these frameworks is the main workflow. When designers use these frameworks to create a migration plan, they need to provide some information about the service oriented system including its structure files and log files. Thereafter, the frameworks can automatically calculate the structure of the application in order to generate a network comprising all the components. In the network, the components with more degrees are selected to utilize fault tolerance strategies. On the other hand, the components with lower reliability tend to be selected. The selected top k components are arranged with fault tolerance strategies. In the group of fault tolerance redundant units, they follow fault tolerance strategies (Duan, Peisert, & Levitt, 2014; Kotla, Alvisi, Dahlin, Clement, & Wong, 2009 ; Qiu, Zheng, Wang, Yang, & Lyu, 2014), which will be depicted in the following section in terms of reliability, execution time and cost. Note that all the fault tolerance strategies are only utilised for the selected components in the frameworks.

The work (Zheng & Lyu, 2010) collects QoS data from collaborated users in order to adjust optimal fault tolerance configuration to achieve good performance. The work (Zheng, Zhou, Lyu, & King, 2010) has proposed a component ranking based framework to identify significant units in a Web base application. Thereafter, fault tolerance strategies (e.g., NVP, RB, etc.) are assigned to these units. The framework (Qiu, Zheng, Wang, Yang, & Lyu, 2014) based on the framework (Zheng, Zhou, Lyu, & King, 2010) takes public and private clouds into consideration.

However, the units selected via these frameworks are distributed in a Web based application. This work can assign fault tolerance strategies to a group of Web services under the same abstract layer. One example is given as follows.

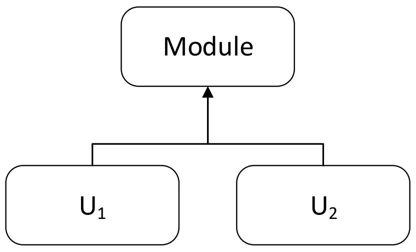


Figure 3.4: An example of an original module

Figure 3.4 shows an example of an original module, which comprises two components, U1 and U2. They are different functions and have a serial relationship.

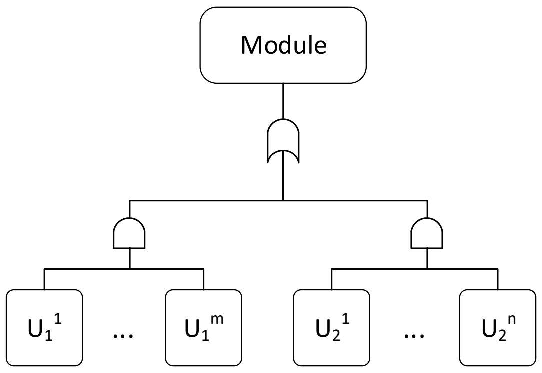


Figure 3.5: Redundancy arrangement for components

Figure 3.5 describes redundancy arrangement for the components in Figure 3.4. There are m redundant components for the component U_1 and n components for the component U_2. The relationship between these two groups is serial, for the module needs to invoke both U_1 and U_2. The traditional frameworks consider the addition of a fault tolerance strategy for the components at the bottom level. Therefore, the possible optimal solution of the redundancy assignment of the entire system may be not received under predefined cost and execution time constraints.

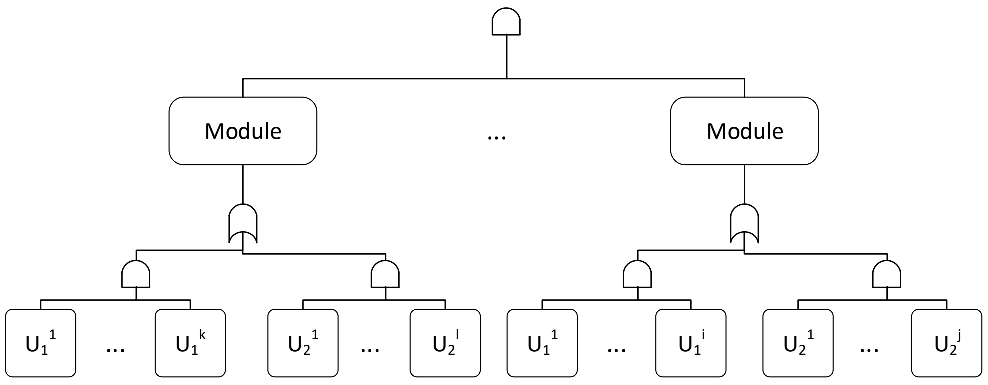


Figure 3.6: Redundancy arrangement for the entire module

Figure 3.6 depicts the redundancy arrangement for the entire module. This approach is adopted in this work, which allows redundant units to be arranged to both components and modules at the different levels. The framework proposed in the work allows redundancy arrangement for modules at both the bottom and higher levels. Therefore, it can result in a larger solution space to search for an optimal solution. In the larger solution space, more solutions can be generated and the optimal solution can be selected to guide designers or cloud managers to migrate and deploy their service oriented systems.

Although the work in (Qiu, Zheng, Wang, Yang, & Lyu, 2014) considers the cost of different fault tolerance strategies, it does not take the global cost constraint into consideration. However, before designers and developers decide to migrate their service oriented systems into a cloud, the cost of the entire system may be computed in advance. From the description of the migration and deployment of service oriented systems, it is clear that a predefined cost constraint may be required and is a key factor of computing the optimal migration solution. Therefore, the cost constraint should be considered in the new migration framework.

Additionally, the execution time of the entire system may be considered in the migration and deployment framework in clouds. The execution time can measure the performance of an application or system. For example, if an application or system can complete all the requests in a shorter execution time, it means that it may have better performance. Based on this reason, the execution time should be considered in the migration and deployment framework. Therefore, the framework in this work considers it to allow the designers to specify the execution time constraint in advance for the integrated genetic algorithm to compute the optimal solution.

The framework in this work considers service reliability, the execution time and cost constraints of a service oriented system, which can employ different fault tolerance strategies (such as RB, NVP and Parallel) for the services. Additionally, it describes a service oriented system as a tree intuitively and all the levels can be assigned with redundant units. The detailed description of the framework and the corresponding integrated algorithm are shown in the following section.

To get the optimal solution to the migration and deployment of service oriented systems, the framework in the work utilises a genetic algorithm. A set of solutions can be generated via the predefined number of iterations. Thereafter, the optimal solution can be selected under the predefined execution time and cost constraints.

3.2 Design of the migration and deployment framework

In this section, the structure of the migration and deployment framework in the work is proposed. The usage environment and components are described.

3.2.1 Usage environment

Software faults are hard to identify, especially the faults of service oriented systems (Zheng & Lyu, 2010). Therefore, various fault tolerance strategies are developed which utilise the redundant units. Namely, more than one redundant functionally equivalent unit deals with a request and then they send back their responses. A decision mechanism collects these responses in order to send back the final response to the requester.

The migration and deployment framework in this work can be used when a company plans to migrate their service oriented systems into a cloud computing platform with the consideration of the addition of fault tolerance strategies in order to improve reliability. Before the company uses the framework, the structure of their systems needs to be described as a tree. At the same time, the reliability of services, execution time and cost constraints are required as input parameters of the framework. Thereafter, the framework will compute the solutions to the migration of the systems. After the certain number of iterations, an optimal solution would be received and sent back to the designers.

There are four main roles involved in the framework when the optimal solution is automatically generated, which are developer, tester, designer and cloud administrator. They are described as follows.

Developers: The developers are responsible for the development of service oriented systems. They have been involved in the stages of software development and realise the structure of the systems. Since the framework needs the structure of the system as an input parameter, the developers need to be involved in the framework.

Testers: The testers of the systems need to be involved in the migration and deployment, for they can provide the log files of all the services. Once they work for a period of time, log files would be generated in servers. Testers can extract the reliability of services and organise them in files. The reliability can reflect the states of these services. For example, the stability and invocation times can be described in the log files. At the same time, the location of end users invoking certain services can be recorded in the files as well. According to the location information, statistical facts can be collected and analysed. The framework in this work needs these log files to define the reliability of each service of an application.

Administrators: They focus on the management of cloud computing platforms. They are responsible for deploying the applications and systems and manage VMs in the cloud computing platform. Companies planning to migrate service oriented systems to a cloud need to purchase computing, storage and networking resources in advance. After they finish the purchase of these resources from cloud computing providers, the migration plan for their applications needs to be generated. Thereafter, the administrators of the cloud computing platform are to deploy their applications and perform corresponding configurations.

Designers: The designers take responsibilities for the management of the migration plans for the systems. When a service oriented system is required to be migrated into a cloud, fault tolerance strategies may be required in order to enhance the reliability of the services. The optimal solution for the migration plan is really hard for a designer to compute manually under cost and execution time constraints. Therefore, a migration and deployment framework with a genetic algorithm is needed to help designers to build up optimal migration and deployment plans.

The framework in this work can automatically compute the solutions as a migration and deployment plan with a genetic algorithm. Developers only need to provide the structure of the service oriented systems, the cost and execution time constraints and log files which describe failure information and warning messages from all of the services. After the certain number of iterations, the genetic algorithm can generate the optimal solution for service oriented systems in clouds.

3.2.2 Components of the framework

In this section, the migration and deployment framework proposed in the work is described. The framework utilises four stages to compute the optimal solution as follows.

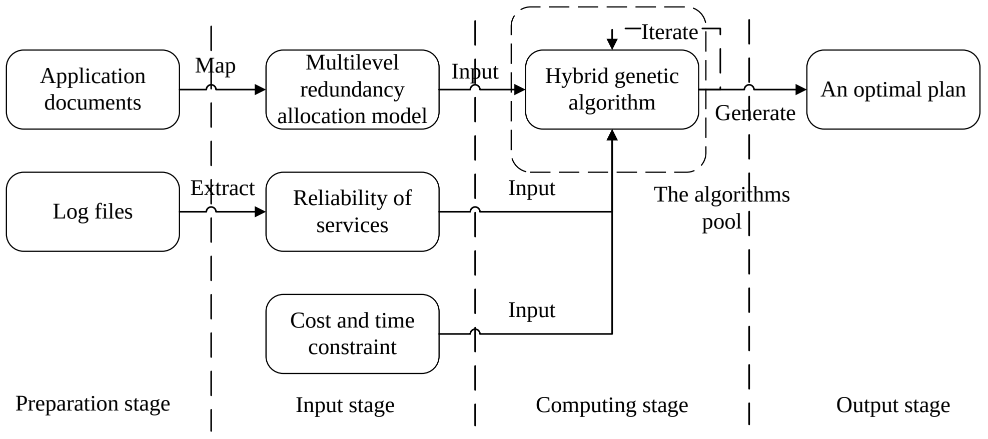


Figure 3.7: The migration and deployment framework for service oriented systems

Figure 3.7 shows the migration and deployment framework for service oriented systems in clouds. Based on the previous work (Qiu, Zheng, Wang, Yang, & Lyu, 2014; Zheng & Lyu, 2015; Zheng, Ma, Lyu, & King, 2013), this thesis proposes cost and time constraints during software migration, uses multilevel redundancy allocation model to describe service oriented applications structure and utilizes hybrid genetic algorithm to calculate an optimal migration plan under these constraints.

Intuitively, there are four stages in the figure, which are the preparation stage, the input stage, the computing stage and the output stage. The detailed functions of all the stages are illustrated as follows.

Preparation stage: This stage mainly prepares all the necessary information for the migration and deployment framework, which includes two components: one is for application documents and the other one is for log files. The component to deal with the application documents mainly receives the structure of service oriented systems as the input parameter for the framework in this work. The application documents are always provided by the developers or administrators in a company. From the application documents, all the information about the design of the application can be collected. Information about the structure of the service oriented system is needed, from which it can be described as a tree, where the abstract functions are regarded as modules at the high levels and the detailed services are at the bottom levels. The tree representation of the system is employed for the multilevel redundancy allocation model, since the genetic algorithm in our work needs to assign redundant units to different levels. From the log files, the reliability of the services can be extracted for input to the framework.

Log files: There are two approaches to generate log files. One method is to get the log files from Web servers (e.g., Apache Tomcat). Another approach is that developers can manually insert source code into Web services to generate log files. In this work, the latter is adopted. In the log files, the states of services (errors and warning messages), execution time, client IP addresses and access time are recorded.

Input stage: Includes three components, which are the multilevel redundancy allocation model, the reliability of the services and the cost and execution time constraints. The multilevel redundancy allocation model describes the structure of the service oriented system. The reliability of the modules can be extracted from the log files. The cost and execution time constraints can be predefined for the generation of the solution. Execution time constraints can measure the performance of a service oriented system.

Computing stage: This stage mainly includes the genetic algorithms. After a certain number of iterations, an optimal solution can be obtained.

Output stage: This stage is responsible for showing the generated optimal solution and sending it to the designers.

Chapter 4 Multilevel Redundancy Allocation Model

4.1 Introduction

In this chapter, the multilevel redundancy allocation model is utilised for service oriented systems. Fault tolerance strategies are integrated into the model for the services at the bottom level. An improved genetic algorithm based on (He, Wu, Xu, Wen, & Jiang, 2013) is proposed and adopted in the migration and deployment framework described in the previous chapter.

Complex software requirements make current software systems larger and larger (Qiu, Zheng, Wang, Yang, & Lyu, 2014). If an error or a warning can be detected and solved, the reliability of the corresponding module can be improved. However, for a service oriented system in a cloud, it is complex and almost impossible to check each service and solve all the possible errors. Therefore, fault tolerance strategies are needed to improve software reliability.

In distributed environments, the reliability of the large-scale software systems cannot be guaranteed (Zheng & Lyu, 2010). Therefore, reliability is always taken into account at the stage of system design in the multilevel redundancy field (He, Wu, Xu, Wen, & Jiang, 2013), which can be calculated from the bottom layer to the top layer. For a multilevel system, there are two approaches to improve reliability: one is to enhance the reliability of each component and the other one is to allocate unit redundancy to the components at different levels. Service oriented systems can be represented as a multilevel redundancy model.

For the optimisation of the multilevel redundancy allocation model, the encoding mechanism plays a key role in the performance of a genetic algorithm. An encoding mechanism called two dimensional arrays encoding has been proposed (He, Wu, Xu, Wen, & Jiang, 2013). Instead of the utilisation of an unfixed length of arrays in the genetic algorithm, a two dimensional array with a fixed length represents a solution to multilevel redundancy allocation problem (MLRAP). In the array, the first dimension means the units assembling the multilevel system. The second dimension of the array denotes the redundant number of the current unit. The new encoding mechanism reduces the complexity of the development of generic algorithm (GA). However, the mechanism also brings new problems. When the algorithm runs, solutions calculated in previous iterations need to be refined. To expand the exploration space of GA, the redundancy of units need to be modified, which leads to the movement of the elements in the array. The disadvantage may be that when a column of children units is added or removed, the following columns must be moved forward or backward. The movement of other columns may increase the additional operations of the algorithm and decrease its efficiency.

To tackle the problem, this work proposes a new encoding mechanism named the linked list based encoding mechanism. A linked list based array is employed to represent the solution, which is also the structure of a multilevel redundancy allocation model. The array can avoid the movement of the elements when a column of the related children units needs to be added or removed. A detailed description of the array is in the following sections.

4.2 Term definition

In this section, the utilised terms of MLRAP and the new encoding mechanism are depicted. To avoid description confusion, the definition of the following items in this work is listed.

1) Unit: The nodes of a multilevel redundancy allocation model.

2) System unit: The root node of a multilevel redundancy allocation model. The unit comprises a lot of subsystem units.

3) Subsystem unit: All the units between the system unit at the top level and the components at the bottom level.

4) Component: Components are the units at the bottom level. They represent the detailed functions of business logic. In service oriented systems, they denote the services.

4.3 Adoption of a multilevel redundancy allocation model

In this section, a multilevel redundancy allocation model (He, Wu, Wen, & al, 2010; He, Wu, Xu, Wen, & Jiang, 2013; Wang, Tang, & Yao, 2010) is adopted for the migration and deployment framework. The model is always utilized to resolve hardware reliability problems. In this thesis, the model is integrated with fault tolerance approaches, RB, NVP and Parallel to describe a service based system in clouds.

For a multilevel redundancy allocation model, redundancy allocation can be performed for the different layers. If the units at the subsystem or system layers are allocated with redundancy, their children units at the lower layer need to be changed a lot.

The entire service oriented system is described as a tree, where the relationship between units is the invocation. The components can work together and constitute an abstract unit. For example, a waybill management unit may consist of the waybill creation, waybill modification, waybill delivery units, etc. The structure of applications can be extracted from software documents.

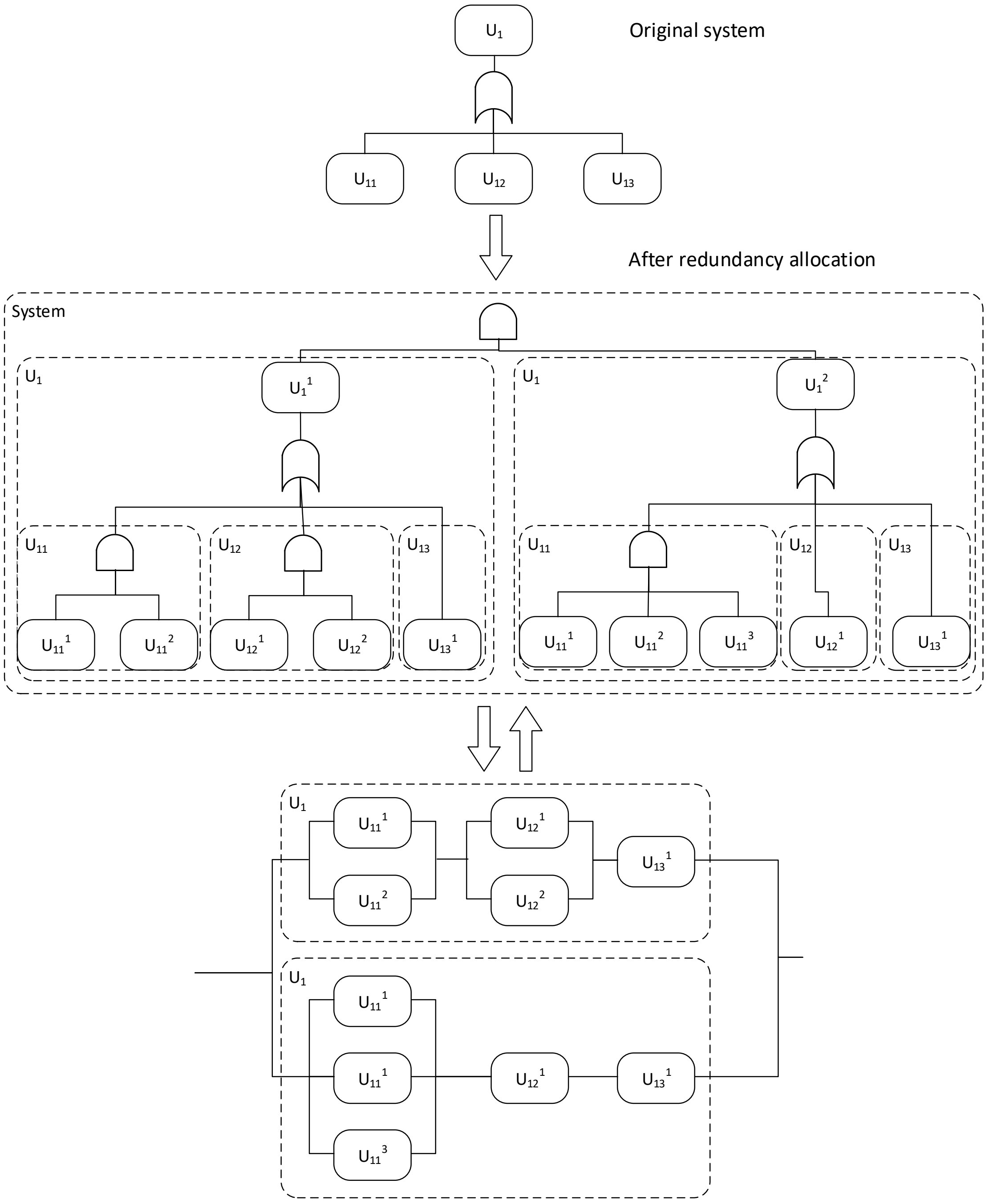


Figure 4.1: An example of the transformation of a multilevel redundancy allocation model

Figure 4.1 shows an example of the transformation of an original multilevel system to a redundant structure. The original model is a bi-level system, which includes one system unit and three components, which have a serial relationship. The original system could be regarded as a function of a service oriented system where the components are the services. U_1 is an abstract unit and U_{11}, U_{12} and

U_{13} are components. In this work, we assume that all the components are atomic services.

In Figure 4.1, the middle graph is a bi-level system after redundancy allocation. The system unit and the components of the original system are separately assigned with redundant units. The system unit U_1 is assigned with redundancy. Therefore, for U_{11}, U_{12} and U_{13}, redundant units would be assigned. In Figure 4.1, for U_1^1, the numbers of the redundant units of U_{11}, U_{12} and U_{13} are 2, 2 and 1. For U_1^2, the numbers of the redundant units of U_{11}, U_{12} and U_{13} are 3, 1 and 1. Note that the relationship of the redundant units with the identical unit is parallel.

In Figure 4.1, the graph at the bottom is another description style of the redundant model, which shows the business flow of a service oriented system. It is obvious that the redundant units are not only allocated to the bottom layer, but the abstract layers as well. Therefore, the reliability of the represented system is enhanced via the redundancy of the units at the abstract layers and the component layer.

4.4 Adoption and modification of an optimisation problem

In this section, the multilevel redundancy allocation optimisation problem is described. In this work, three features of a service oriented system are taken into consideration, which are the reliability, the resource cost and the execution time. The following sections describe the computing approach of the features of a model. An example is given as follows.

Figure 4.2 shows an original system and the system with redundant units. In the original system, there are two levels. After redundancy allocation, the units, U_{11}, U_{12} and U_{13} are equipped with five, three and two redundant units.

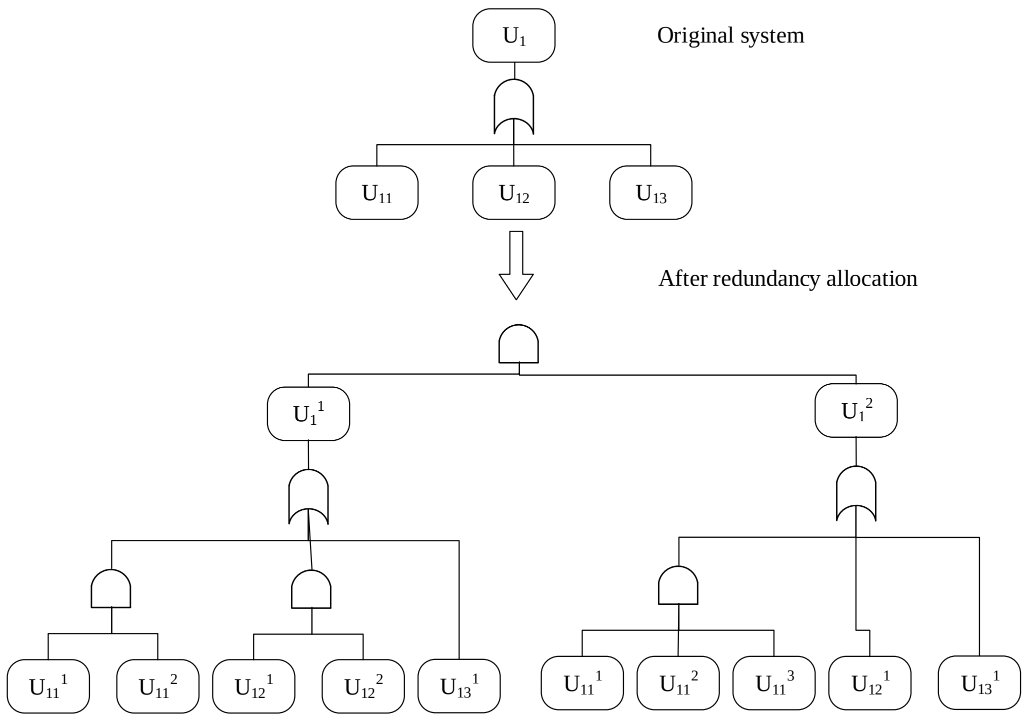


Figure 4.2: An example of redundancy allocation model

4.4.1 Adoption of reliability computing

In this section, the reliability computing method of the multilevel redundancy allocation model is described. The reliability of the entire model can be computed via the addition of redundant units from the bottom level to the top level (He, Wu, Wen, & al, 2010; He, Wu, Xu, Wen, & Jiang, 2013; Wang, Tang, & Yao, 2010). Since all the components have their own reliability, the total reliability of the model needs to add all the values together. There are two types of relationships between the units in the model at different levels. One is parallel and the other one is serial. For these two types, the computing approaches are different, which are described as follows.

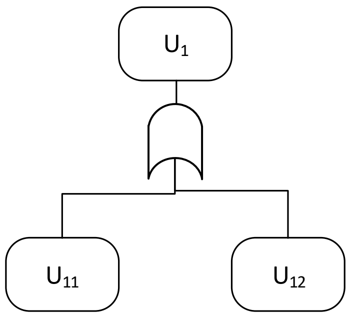


Figure 4.3: The serial relationship between two units

Figure 4.3 shows the serial relationship between two units. The serial relationship means that they have different functions and perform different tasks. Therefore, the reliability of the model shown in Figure 4.3 is computed as follows.

$$R = (1 - f_{11}) \times (1 - f_{12}) \tag{4.1}$$

In the formula, R is the reliability of the entire model. f_{11} and f_{12} are the failure rates of the units, U_{11} and U_{12}, respectively.

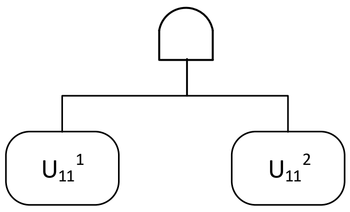


Figure 4.4: The parallel relationship between two units

Figure 4.4 shows the parallel relationship between two units. The units are identical, for the original unit is replicated. Hence, they could finish the same request. In this case, the reliability of the module in Figure 4.4 can be computed as follows.

$$R = 1 - f_{11}^{1} \times f_{11}^{2} \tag{4.2}$$

f_{11} denotes the failure rate of the original unit. Since they are replicated, a minimum of one unit needs to be guaranteed to perform the processing of a request and return its response.

$$R_i = \prod_{k=1}^{m_i} \left[1 - \prod_{j=1}^{x_{ik}} \left(1 - R_{ik}^{j}\right) \right] \tag{4.3}$$

Based on the analysis above, the reliability of a multilevel redundancy allocation model with a serial and parallel relationship of the units can be calculated in the formula. R_i represents the reliability of the subsystem unit i. R_{ik}^{j} denotes the reliability of the jth redundant unit of the kth unit. The Figure 4.5 illustrates the algorithm for computing reliability.

```
Algorithm name:    Compute_reliability
Input:             $S$: The structure of a multilevel redundancy allocation
                   model
Output:            $R$: The reliability
Steps:
// Compute all the nodes from the bottom level to the top level
for(int $i$ = levels_num; $i$ > 0; $i$--) {
   if($S(i)$ is a component) {
      $R_i$ = Compute_parallel($S(i)$); // Compute the parallel relationship
   } else if($S(i)$ is not a component) {
      $R_{sys}$ = Compute_serial($S(i)$, $R_i$, $R_{sys}$);
   }
}
return $R_{sys}$;
```

Figure 4.5: The algorithm of computing reliability

Therefore, the reliability of the model in Figure 4.2 can be calculated in the formula. R_{sys} is the reliability of the entire model.

$$R_{sys} = 1 - \left\{1 - \left[\left(1 - \prod_{j=1}^{2}(1 - R_{11}^{j})\right)\left(1 - \prod_{j=1}^{2}(1 - R_{12}^{j})\right)(R_{13}^{1})\right]\right\} \times \left\{1 - \left[\left(1 - \prod_{j=1}^{3}(1 - R_{11}^{j})\right)(R_{12}^{1})(R_{13}^{1})\right]\right\} \quad (4.4)$$

In this thesis, service reliability can be calculated via collecting data from logs. Since a migration plan needs to be generated before an application migrates into a cloud computing platform, it is impossible to get service logs in the cloud. Therefore, to get the service reliability, a program can be completed to send some requests to these services. Thereafter, the states of requests and responses can be collected. Service reliability is the ratio of correct and total responses.

4.4.2 Adoption of cost computing

In this section, the algorithm for computing the cost of a multilevel redundancy allocation model is illustrated. The cost of the entire model can be calculated through the addition of the cost of units from the bottom level to the top level (He, Wu, Xu, Wen, & Jiang, 2013; Wang, Tang, & Yao, 2010).

$$C_i = \sum_{k=1}^{m_i} \left(\sum_{j=1}^{x_{ik}} C_{ik}^{j} + (\partial_{ik})^{x_{ik}} \right) \tag{4.5}$$

The formula shows the cost of a model. ∂_{ik} is the configuration cost of the redundancy allocation for U_{ik}.

Algorithm name:	Compute _cost
Input:	S: The structure of the multilevel redundancy allocation model
Output:	C: The cost

```
Steps:
// Compute all the nodes from the bottom level to the top level
for(int i = levels_num; i > 0; i--) {
   if($S(i)$ is a component) {
      $C_i$ = Compute_component_cost ($S(i)$);
   } else if($S(i)$ is not a component) {
      $C_{sys}$ = Compute_sum($S(i)$, $C_i$, $C_{sys}$);
   }
}
return $C_{sys}$;
```

Figure 4.6: The algorithm of computing cost

According to the above formula and algorithm, the cost of the model in the Figure 4.2 is shown as follows. C_{sys} is the cost of the entire model.

$$\begin{aligned} C_{sys} = {} & \left[\left(\sum_{j=1}^{2} C_{11}^{j} + (\partial_{11})^2 \right) + \left(\sum_{j=1}^{2} C_{12}^{j} + (\partial_{12})^2 \right) + (C_{13} + \partial_{13}) \right] + \\ & \left[\left(\sum_{j=1}^{3} C_{11}^{j} + (\partial_{11})^3 \right) + (C_{12} + \partial_{12}) + (C_{13} + \partial_{13}) \right] \end{aligned} \tag{4.6}$$

4.4.3 Development of execution time computing

In this section, the computing method for the execution time is described. To improve processing correctness, a request needs to be received and processed by all the redundant units. Therefore, in the parallel relationship at the bottom level, the execution time of a request is the maximum value, which means the execution time of the last response.

$$T_i = \sum_{j=1}^{m} \left(\max\left(t_{ij}^{k}\right) + \lambda^{|M_k|}\right), k\epsilon Y \tag{4.7}$$

This formula shows the execution time of a multilevel redundancy allocation model. t_{ij}^{k} represents the execution time of the kth redundant unit of the jth unit in the model i. λ is the extra time of the redundancy configuration. M_k is the redundant number of the unit j.

Algorithm name:	Compute_execution_time
Input:	S: The structure of the multilevel redundancy allocation model
Output:	T: The execution time

```
Steps:
// Compute all the nodes from the bottom level to the top level
for(int i = levels_num; i > 0; i--) {
   if($S(i)$ has redundant units) {
      $T_{sys}$ = Compute_max_time ($S(i)$);
   } else if($S(i)$ does not have redundant units) {
      $T_{sys}$ = Compute_sum($S(i)$, $T_i$, $T_{sys}$);
   }
}
return $T_{sys}$;
```

Figure 4.7: The algorithm for computing the execution time

Figure 4.7 shows the algorithm for computing the execution time. In the algorithm, there are two types of relationship: one is a parallel relationship and the other is a serial relationship. Therefore, as with the algorithm for computing the reliability, the computing method for the execution time depends on the specific structure of the model. For a parallel relationship, the execution time is the maximum value of time of all the responses. On the other hand, for a serial relationship, the execution time of the model needs to add the maximum time of all the responses. Therefore, in Figure 4.7, the functions, Compute_max_time() and

Compute_sum(), are responsible for computing the maximum value of the execution time and the sum of all given ones.

Based on the above analysis and description, the execution time in Figure 4.2 can be shown as follows. T_{sys} is the execution time of the entire model.

$$T_{sys} = (Max(t_{11}^2) + Max(t_{12}^2) + t_{13}) + (Max(t_{11}^3) + t_{12} + t_{13}) \tag{4.8}$$

4.4.4 Modification of the optimisation problem

In this section, the optimisation problem is defined based on the computing approaches for the reliability, the cost and the execution time of a service oriented system. The reliability problem of a multilevel redundancy allocation model is an optimisation problem. Since redundant units can be allocated to all the levels of a model, the optimal solution for redundancy allocation needs to be determined, which can maximise the reliability of the entire model and meet the predefined constraints. In our work, the predefined constraints are the cost constraints and the execution time constraints.

The reason to propose the cost of a unit is that the redundant units both bring higher reliability and cause extra cost. The redundant units lead to extra configuration cost as well. In clouds, the computing, networking and storage resources commonly need to be purchased. The optimal migration and deployment plan allows companies to adopt the minimum cost in order to maximise reliability. Therefore, the cost constraints need to be considered.

The execution time is a feature to measure the performance of a service oriented system. The quick processing of a request from end users is pivotal. Apart from the cost, the execution time is also taken into consideration in this work. Additionally, the performance of a service oriented system is also influenced by the network condition. However, in this work, the network condition will be ignored, since in the actual deployment, an application or a system ideally can be deployed in an identical cloud. Therefore, the optimisation problem is described as follows.

$$\text{Maximize} \quad R_{sys} = R(x_{11}, \dots, x_{ik}) \tag{4.9}$$
$$\text{Subject to} \quad C_{sys} = C(x_{11}, \dots, x_{ik}) \leq C_0, \qquad 1 \leq x_{im} \leq \alpha_i \tag{4.10}$$
$$T_{sys} = T(x_{11}, \dots, x_{ik}) \leq T_0 \tag{4.11}$$

The formula shows the optimisation problem of a multilevel redundancy allocation model. R_{sys} is the reliability of the entire model. x_{ik} is the units of the model. C_{sys} denotes the cost and the configuration cost of the entire model. C_0 represents the predefined cost constraint. α_i is maximum redundant number of a unit. T_{sys} is the execution time of the entire model and T_0 is the predefined execution time constraint. In the following section, an available and efficient algorithm is employed to generate the optimal solution for the migration and deployment plan.

4.5 Adoption and modification of genetic algorithm

In this section, a hybrid genetic algorithm is adopted with a new encoding mechanism to generate an optimal solution. An improved genetic algorithm based on (He, Wu, Xu, Wen, & Jiang, 2013) is employed for the generation of the migration plan for service oriented services in clouds. To improve performance, a new encoding mechanism is proposed. Additionally, the hybrid genetic algorithm can integrate with NVP, RB and Parallel for the components at the bottom level in order to acquire the optimal solution.

The genetic algorithm needs to take reliability, cost and execution time into consideration, which are the purposes that the migration and deployment framework will achieve. Additionally, the genetic algorithm needs to be efficient, for it needs to continue generating solutions. Therefore, the encoding mechanism to operate the increase and decrease of units of a model can affect the performance of the genetic algorithm.

4.5.1 Traditional encoding mechanism

The hierarchical encoding mechanism is commonly utilised in genetic algorithms in order to generate the optimal solution for a multilevel redundancy allocation model (He, Wu, Xu, Wen, & Jiang, 2013; Kumar, Izui, Masataka, & Nishiwaki, 2008; Wang, Tang, & Yao, 2010). The hierarchical encoding mechanism (He, Wu, Xu, Wen, & Jiang, 2013) is at first described. Based on the mechanism, a novel encoding mechanism is proposed, which can save the storage space and improve the performance of the genetic algorithm.

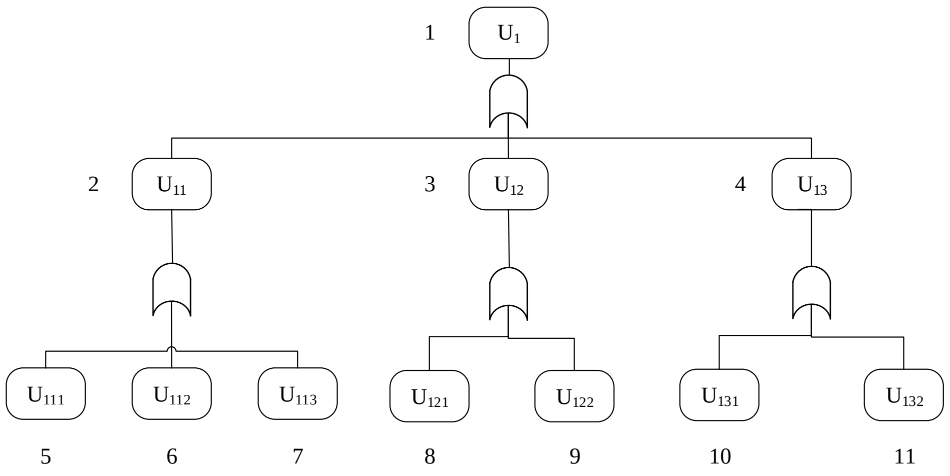


Figure 4.8: An example of a tri-level multilevel redundancy allocation model

To explain the traditional hierarchical encoding mechanism, an example of a multilevel redundancy allocation model with three levels is described in Figure 4.8, which has three levels and eleven units.

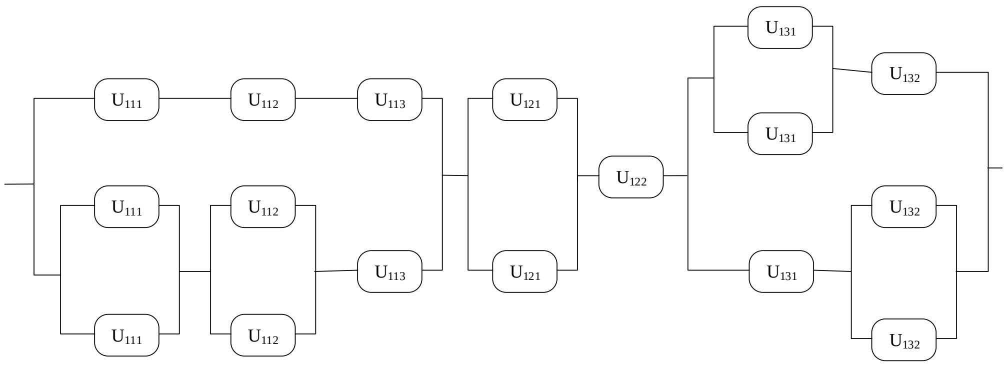


Figure 4.9: The model after redundancy allocation

Figure 4.9 describes the model after redundancy allocation. In the figure, the unit, U_{11} is allocated one redundant unit. Since there are three children units of U_{11}, the units are replicated as well. In this case, it is apparent that the redundancy allocation for a unit at a higher level leads to bigger changes to the structure of the model.

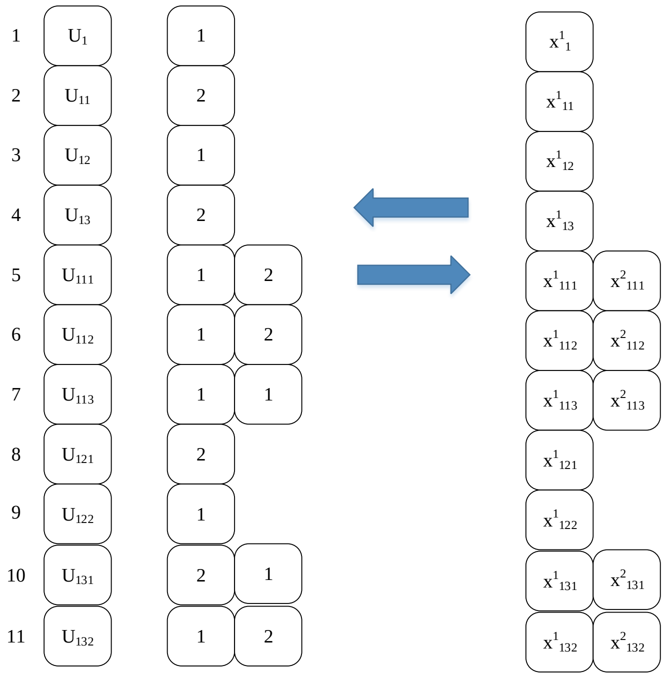


Figure 4.10: The traditional encoding mechanism

Figure 4.10 illustrates the traditional encoding mechanism, which utilises a two dimensional array to store the redundancy of all the units at different levels. The structure of the model can be described via the indices of the array. For example, the root of a model is in the first row. In the same manner, the units, U_{11}, U_{12} and U_{13} at the second level are in the second, third and fourth rows. Therefore, the model shown in Figure 4.9 is described in Figure 4.10. In the two dimensional array, all the elements describe the redundant number of each unit. For example, U_1 does not have redundant units. Hence, the redundant value in the array is 1. Additionally, since U_{11} has one redundant unit, the values of U_{111}, U_{112} and U_{113} are $(1,2)$, $(1,2)$ and $(1,1)$. The redundant number of the parent unit determines the amount of columns of children units in the array. The utilisation of the two dimensional array can intuitively describe the structure and the number of the redundant units.

The encoding mechanism needs more storage space than it actually requires. To define the array, the length of the array needs to be predefined in most programing languages. However, in Figure 4.10, it is obvious that the abstract units commonly utilise some of the storage space, namely, some of the columns of the array. When the redundancy allocation model has a lot of levels, the problem may be serious. For example, the array in Figure 4.10 must have more than $11 \times 25 \times 5 = 1375$ elements, when the maximum redundancy number of each unit in the model is 5. Specifically, 11 is the number of the units in the model. The situation with the maximum redundant units is that the root has 5 redundant units. Therefore, its children units have a maximum of 5 columns and the maximum value of each element is 5. Thereafter, the maximum number of the column of the units at the bottom level would be 25. Therefore, when the two dimensional array is defined, the row and column number needs to be 11 and 25, respectively. We can imagine that in this situation the number of the empty columns at the first row is 24 and the amount of the empty columns at the second level is 20. According to Figure 4.10, the total number of the maximum empty elements is $24 + 20 \times 3 = 84$. Therefore, it is clear that the encoding mechanism wastes a lot of storage space.

Another problem of the encoding mechanism may be the efficiency of the operation of each element in the array. Specifically, the genetic algorithm needs to generate and obtain the optimal solution via iteration. In each generation, new solutions will be generated. The approach to generate a new solution is to increase or decrease the redundant number denoted in the array. A detailed explanation of the operation is shown as follows.

Figure 4.11 shows the example of increase operation of a redundant unit. The root unit adds a redundant unit. Since the root unit is at the top level of the model, the change would alter the structure of the entire model at different levels. At first, the units at the second level add a new column to store new redundant units. In Figure 4.11, the units, U_{11}, U_{12} and U_{13}, separately add a column with the values of x_1, x_2 and x_3. The transformation of the units at the second level leads to the increase operation of the components at the bottom level, which add x_1, x_2 and x_3 columns, separately. After the increase operation of the redundant units for the different levels, the new redundant number is generated randomly. Thereafter, the solution would be computed and the reliability, the cost and the execution time would be compared with the predefined constraints.

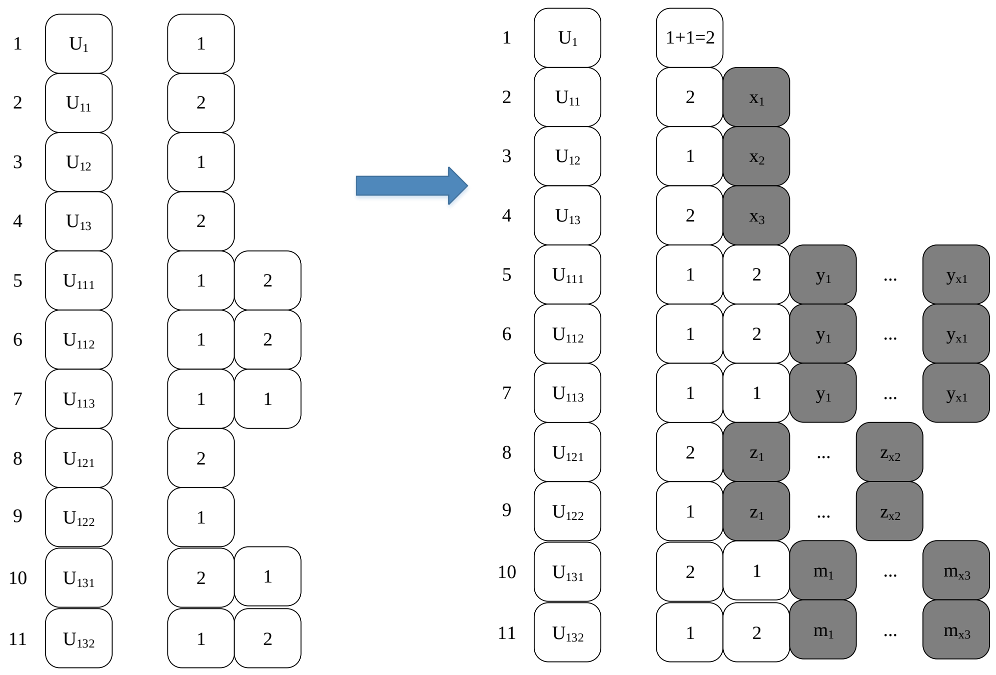


Figure 4.11: The example of the increase operation of a redundant unit

In another example, if the new generated column is between the first and the second column at the second row, the second column with the value x_1 would be moved backward. When a unit has a lot of redundant units, its children units can have more columns to store redundant numbers. In this case, the newly generated column makes the elements of the children units move a lot of columns backwards. The number of the moved columns equals the increase amount of the added redundant values of the parent unit. Similar to the increase operation of the redundant units, the approach to decrease redundant units is illustrated as follows.

Figure 4.12 describes an example of the decrease operation of a redundant unit, which utilises the original model shown in Figure 4.8. The decrease operation is performed for the unit U_{13}. When the number of the redundant units is changed from 2 to 1, the number of the columns of its children units would be changed to 1. Originally, these children units had two columns. Based on the genetic algorithm (He, Wu, Xu, Wen, & Jiang, 2013), the element of a random column would be removed. In the example of Figure 4.12, the element of the first column is removed. After the removal operation, the elements after the column in the same row would be moved forward. Similar to the increase operation of a redundant unit, the decrease operation would influence a lot of columns, if its parent unit has several redundant units.

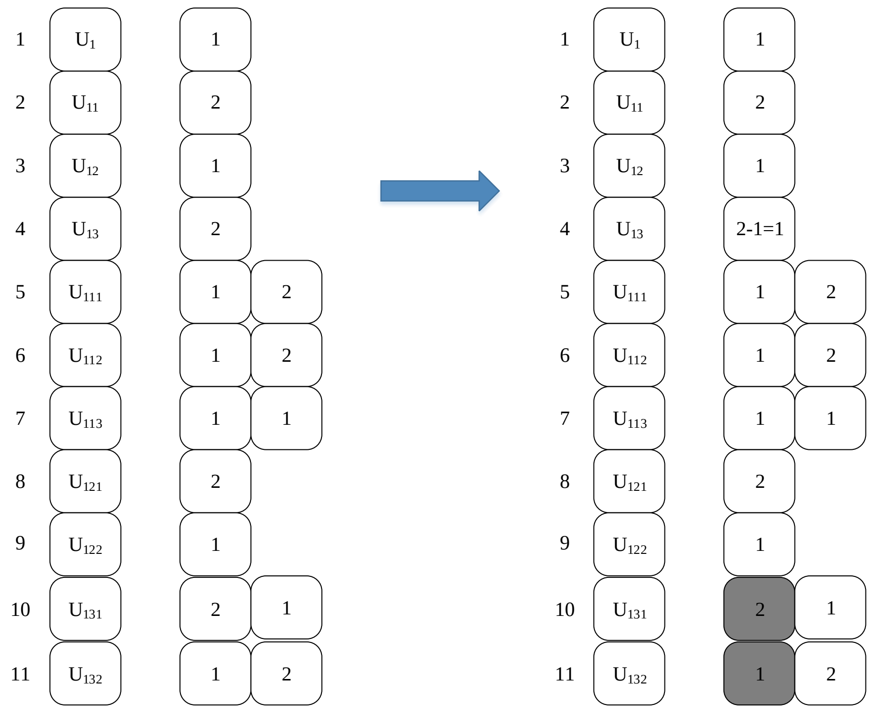


Figure 4.12: An example of the decrease operation of a redundant unit

In summary, the traditional encoding mechanism needs to be improved in terms of data storage and the increase and decrease operation of redundant units. In terms of data storage, if the mechanism can utilise the storage space when it needs, space can be saved. For the increase and decrease operation of redundant units, if it would not influence the elements in other columns in the same row, a few steps would be performed. In each iteration, new solutions will be explored. Therefore, fewer movement steps during solution generation can result in higher performance. Therefore, in the following section, the new encoding mechanism will be proposed to resolve these two problems.

4.5.2 Development of linked list based encoding mechanism

In this section, a linked list based encoding mechanism is proposed, which is based on the mechanism (He, Wu, Xu, Wen, & Jiang, 2013). The new encoding mechanism can save storage space and improve the performance of the increase and decrease operation of redundant units.

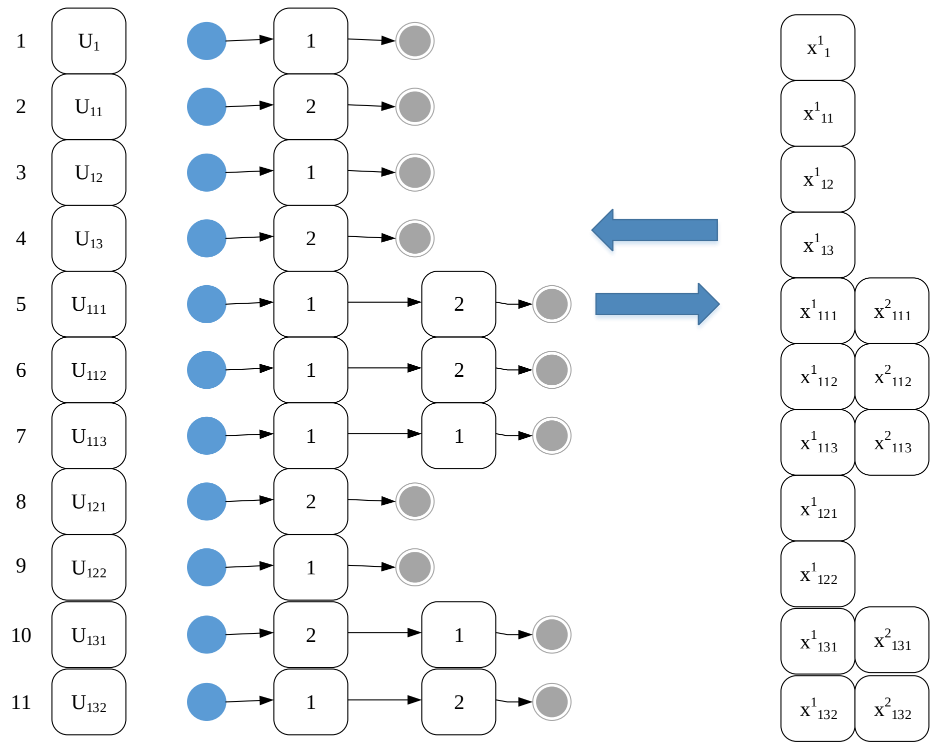


Figure 4.13: The structure of the new encoding mechanism

Figure 4.13 describes the structure of the new encoding mechanism, which stores the model based on the original system shown in Figure 4.8. In the encoding mechanism, the structure of a solution is stored as a linked list based array. The indices of the array describe the structure of the model. However, for each unit at a row in the array, the elements of the columns are connected based on a linked list structure. The value of the redundant units is stored in a node of the linked list.

The mechanism can save storage space and reduce the overhead of the increase and decrease operation of redundant units. The detailed description is shown as follows.

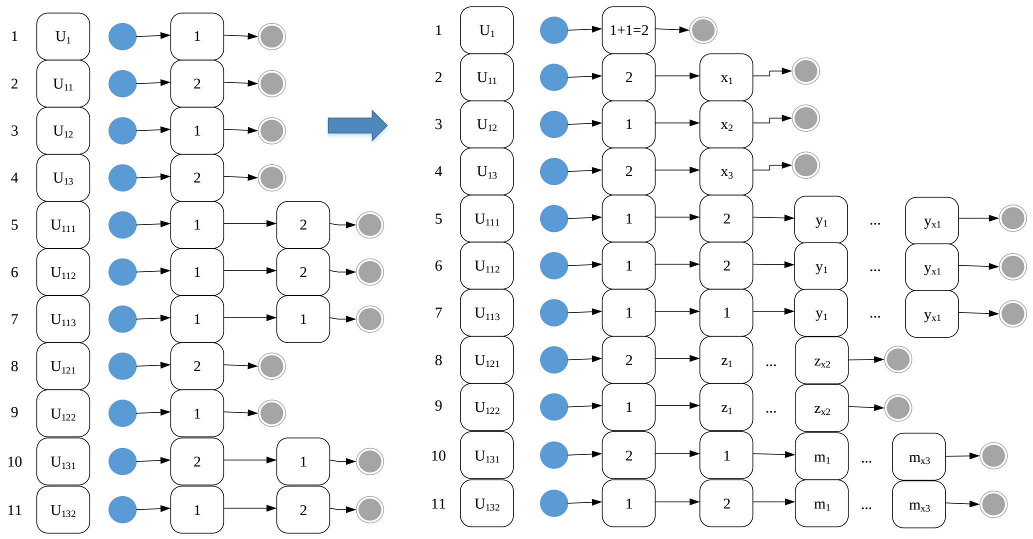


Figure 4.14: The increase operation of a redundant unit

Figure 4.14 depicts the increase operation of a redundant unit for the root unit. With the addition of 1 redundant unit to the root unit, the children units, U_{11}, U_{12} and U_{13}, need to add more nodes to the corresponding linked list. The encoding mechanism does not need the array with the predefined fix length for each unit in advance, which arranges the storage space according to the actual requirements. When a redundant unit is added to a unit, its children units only need to create new nodes and find out the position where the nodes would be inserted, and added them in the linked list. Additionally, the increase operation of the elements of a unit in the traditional mechanism is replaced by the change of the corresponding pointers. The linked list can avoid the movement of these elements.

Figure 4.15 illustrates the decrease operation of a redundant unit. To explain the decrease operation, the similar structure shown in Figure 4.12 is given as an example. In the figure, the redundant value of the first node of the unit U_{13} is changed from 2 to 1. Accordingly, the children units, U_{131} and U_{132}, need to be changed. The first nodes of these two linked lists are removed. The corresponding resources of the nodes are released. Similar to the increase operation of a redundant unit, the decrease operation in this work does not need to move other elements. Only the position needs to be confirmed and the corresponding links need to be modified.

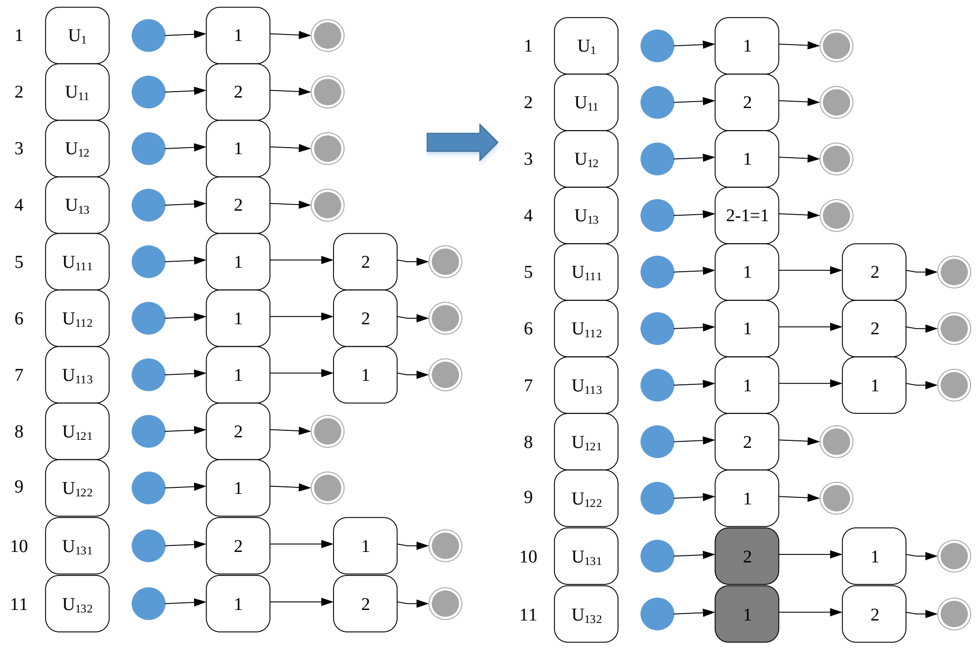


Figure 4.15: The decrease operation of a redundant unit

In summary, the utilisation of the linked list not only saves the storage space, but improves the performance of the change of redundant units as well. Since the utilised genetic algorithm focuses on the operation of the array, this can reduce the overhead of the entire algorithm. This work takes advantage of the linked list based array as the data storage of the solutions generated by the genetic algorithm. The following section would describe the utilised genetic algorithm.

4.5.3 Adoption of genetic algorithm framework

In this section, the genetic algorithm framework is described. The framework describes the main steps of the genetic algorithm. All the parts of the algorithm will be described in the following sections. In this work, the similar framework (He, Wu, Xu, Wen, & Jiang, 2013; Wang, Tang, & Yao, 2010) is proposed as follows.

Figure 4.16 shows the genetic algorithm framework. Note that there is iteration in the framework. The genetic algorithm is to find out the optimal solution via a lot of attempts. In each generation, some new solutions are generated and the solution set is refined. At last, the optimal solution can be got.

```
Algorithm name: Genetic algorithm framework
Input parameters: None
Output parameters: None

1. Initialize population $P0$ with $N$ individuals.
2. // $t < \max\_generation\_val$ is defined in advance
3. While $t < \max\_generation\_val$ {
4.     Randomly select $X$ solutions from $P0$ and invoke local_search method to
       generate new solutions to $Q$
5.     $R = P0 + Q$
6.     Invoke fitness_value method to compute the fitness values of the solutions
       in $R$
7.     Sort $R$ in descending order according to the fitness values
8.     Select top $N$ solutions from $R$ and store them to $P0$ for the next
       generation
9. }
```

Figure 4.16: The genetic algorithm framework

4.5.4 Adoption and modification of fitness function

In this section, the fitness function is proposed. Since the multilevel redundancy allocation problem needs to be optimised under the predefined constraints, there needs to be a mechanism to evaluate the quality of the solutions generated in each generation, which would influence the final result, namely, the optimal solution of the genetic algorithm.

$$fitness_{value(X)} = R(X) \times p(X) \tag{4.12}$$

The formula is the fitness function, where $\mathrm{p(x)}$ denotes a penalty function proposed by (Gen & Cheng, 1996). Since the fitness function evaluates the quality of the solutions generated by the genetic algorithm, it should take the reliability, the cost and the execution time constraints into consideration.

$$p(X) = w_1 p_c(X) + w_2 p_t(X) \tag{4.13}$$

Where

$$w_1 + w_2 = 1 \quad (4.14)$$

The formula shows the penalty function, which considers the cost and the execution time of a solution. Since there are two predefined constraints, the solutions need to meet the constraints. In the formula, the weights, w_1 and w_2, are employed to adjust the influence of the cost and the execution time, which can be set in advance by the designers for the migration and deployment of service oriented systems. The higher reliability and less resource cost and execution time are the purpose of the quality evaluation. $p_c(X)$ has been shown in (He, Wu, Xu, Wen, & Jiang, 2013)

$$p_t(X) = 1 - pow((T(X) - T0)/\Delta T_{max}), K) \quad (4.15)$$

Similar to the representation of the cost, the formula shows the penalty function of the execution time. $T(X)$ is the execution time of the current solution, X. T_0 is the predefined execution time constraint. $T_{max}(X)$ is the maximum value of the execution time of all the solutions in the current generation. K is a parameter proposed in the work (He, Wu, Xu, Wen, & Jiang, 2013). The Figure 4.17 shows the algorithm of the fitness function.

In summary, the fitness function mainly evaluates the quality of the solutions generated by the genetic algorithm in terms of the reliability, the cost and the execution time. The penalty function can be expanded to comprise more constraints.

```
Algorithm name: fitness_value
Input parameters: A solution $X$, the resource cost constraint $C_0$, the execution time constraint $T_0$, the weights $w_1$ and $w_2$
Output parameters: The fitness value of $X$

Steps:
1. Compute the reliability of $X$ and set the result to $R(X)$.
2. Compute the resource cost of $X$ and set the result to $C(X)$.
3. Calculate the execution time of X and give the value to T(X).
4. Calculate $\Delta C = C_{max} - C_i$ and get the maximal $\Delta C_{max}$
5. Compute $\Delta T = T_{max} - T_x$ and get the maximal $\Delta T_{max}$
6. If $\Delta C_{max} == 0$, $\Delta C_{max} = 1$. The same operation is performed to $\Delta T_{max}$
```

```
7. Compute  pcost = $1 - pow((C(X) - C0)/\Delta C_{max}), K)$
8. Compute  $pt = 1 - pow((T(X) - T0)/\Delta T_{max}), K)$
9. return  $R_x * (w_1 * pcost + w_2 * pt)$
```

Figure 4.17: The algorithm of the fitness function

4.5.5 Development of population initialisation

In this section, the population initialisation approach (He, Wu, Xu, Wen, & Jiang, 2013) is utilised and described. Before the execution of the genetic algorithm, a set of solutions would be generated as the initial ones. Thereafter, the genetic algorithm performs the local search based on the solutions in order to generate the optimal solution.

The population initialisation approach tries to generate a set of multilevel redundancy allocation solutions. The redundancy numbers of the different units are randomly created in the range of 1 and the predefined maximum value. The generation of a solution begins from the top level to the bottom level. The following presents the detailed method.

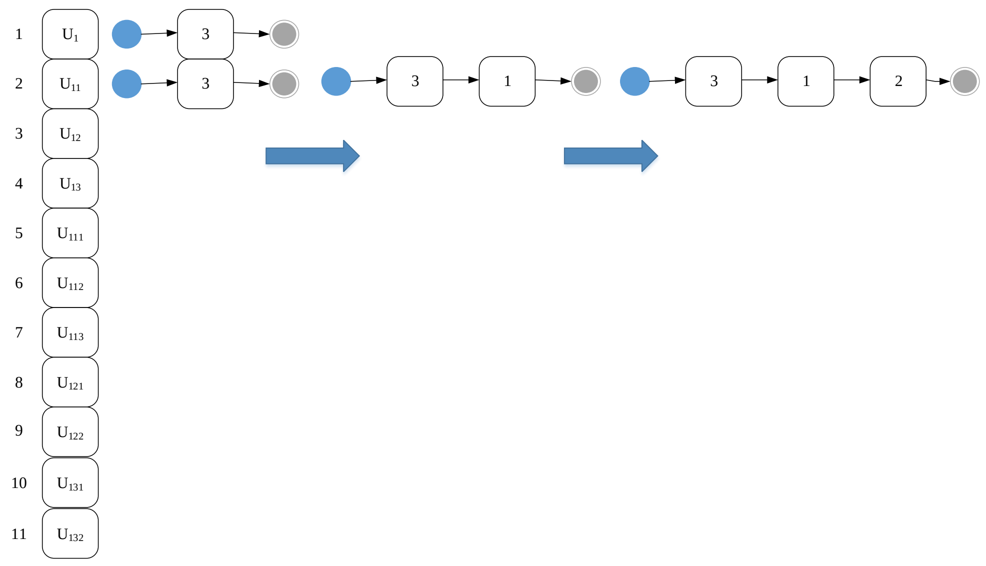


Figure 4.18: An example of the population initialisation approach

Figure 4.18 shows an example of the population initialisation approach. The root unit, U_1, has 3 redundant units. Therefore, its children units have three nodes in the linked list. When a node in the linked list is generated, the value between 1 and the predefined maximum redundant value is randomly generated.

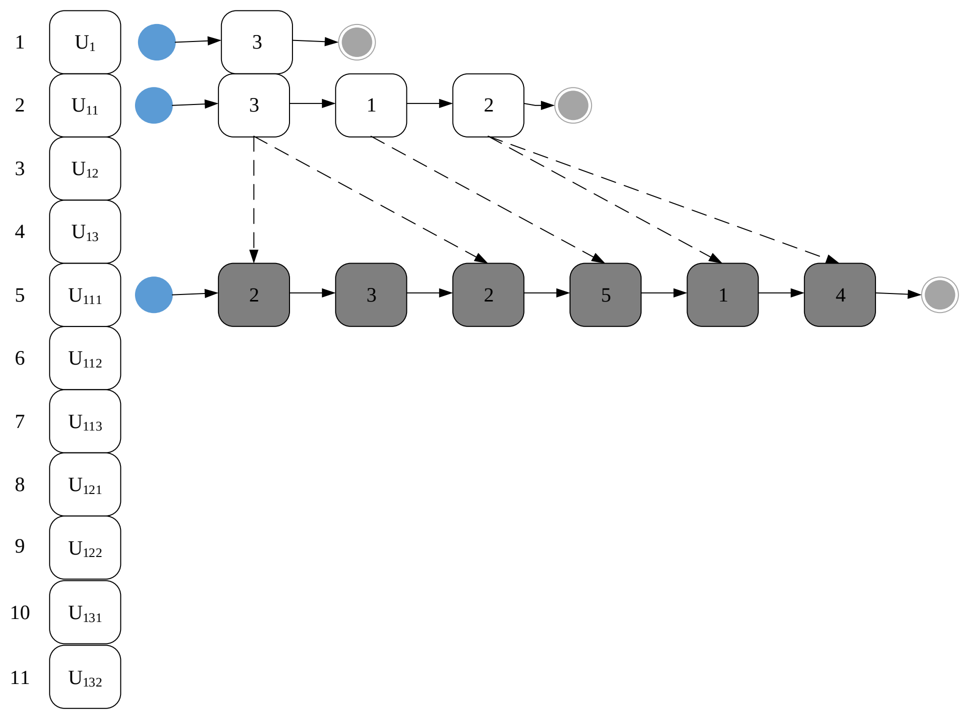


Figure 4.19: The relationship between the parent and children units

Figure 4.19 describes the relationship between the parent and children units, when the population initialisation is performed. After the redundant units are allocated to the units at the second level, the corresponding components at the bottom level would be handled. Therefore, after the unit, U_{11} is allocated with redundancy, U_{111} is arranged with redundant units. In the figure, there are 6 nodes in the linked list of the unit, U_{111}, for the total redundancy number of the parent unit, U_{11}, is $3 + 1 + 2 = 6$. Therefore, the redundancy number of the children units equals the sum of the values of the nodes in the parent's linked list.

All the units with the identical parent unit have the same node numbers in its linked list. Additionally, when the population initialisation is performed, the operation is actually iteration. After a parent unit is allocated with a set of redundant nodes, its children units would be handled in the same manner. When the operation of a branch of the tree-like structure is completed, another branch begins to be processed. The detailed algorithm is described as follows.

```
Algorithm name: population_initialization
Input parameters: The empty fundamental population $P$, the current unit index $i$ndex, the parent's redundant number $parent\_num$
Output parameters: The initialized population $P$
Steps:
1.  If($P[index]$ is not a component at the bottom level)
2.    create a linked list $list$
3.    While($j < parent\_num$)
4.      $list = create\_node(list)$
5.    $P[index] = list$
6.    If($P(index)$ has children units)
7.      While($j < children\_num$)
8.        $population\_initialization(P, index\_child, redundant\_num)$
9.      End
10.   End
11. Else // $P(index)$ is a component at the bottom level
12.   Return $P$
13. End
```

Figure 4.20: The algorithm of the population initialisation

Figure 4.20 shows the algorithm of the population initialisation. It is clear that the entire algorithm is iteration. The performance of the population initialisation is mainly influenced by the level number of a model, the branch number and unit number.

4.5.6 Development of the wheel selection method

In this section, the wheel selection method is described in order to select an index representing a unit in a model from the top level to the bottom level due to the values computed via the sensitivity analysis. The utilised wheel selection method is shown as follows.

Figure 4.21 shows the algorithm of the wheel selection method, which selects the index of the units in the array according to the values computed through the sensitivity analysis described in the section, 4.5.7. Since the change of the redundant number of a unit at a high level can lead to the bigger transformation of the structure of the entire model, the sensitivity analysis is configured to give a smaller value to the units at a higher level. This makes the exploration space of new solutions to be a valid range.

```
Algorithm name: wheel_selection
Input parameters: The change values of all the units from sensitivity analysis, $SA()$
Output parameters: The selected index $index$
Steps:
1.  Get the sum of all the values in $SA()$.
2.  $rand = rand\_num()$ // Randomly generate a value between 0 and the sum.
3.  While($j < length(SA())$)
    1) $sum += SA(j)$
    2) If($rand < sum$)
    3)     Return $j$
4.  End
```

Figure 4.21: The algorithm of the wheel selection function

4.5.7 Adoption and modification of sensitivity analysis

In this section, the sensitivity analysis for the solutions of the genetic algorithm is described based on the sensitivity analysis (He, Wu, Xu, Wen, & Jiang, 2013), which can be used for the service oriented services. The unit can be selected according to the sensitivity values, which are computed in terms of the reliability, the cost and the execution value. Since it is complex to consider modification of units of each solution, the method to compute the reliability and cost values of sensitivity analysis according to the change of units in original system has been proposed (He, Wu, Xu, Wen, & Jiang, 2013). Therefore, the approach to compute the execution time is given as follows.

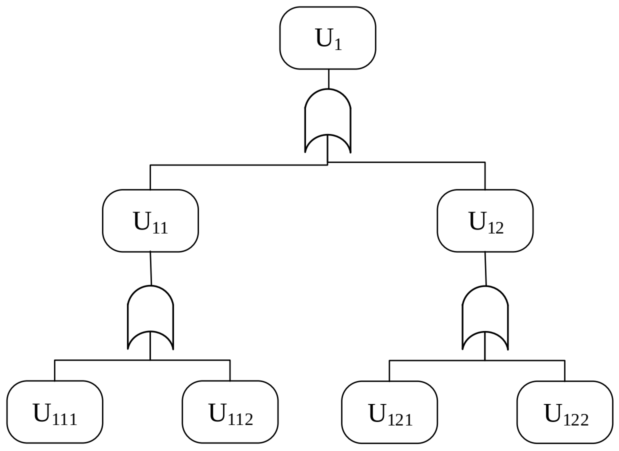


Figure 4.22: An original system as the example of the sensitivity analysis

Figure 4.22 shows an original system as the example of the sensitivity analysis. There are seven units and three levels.

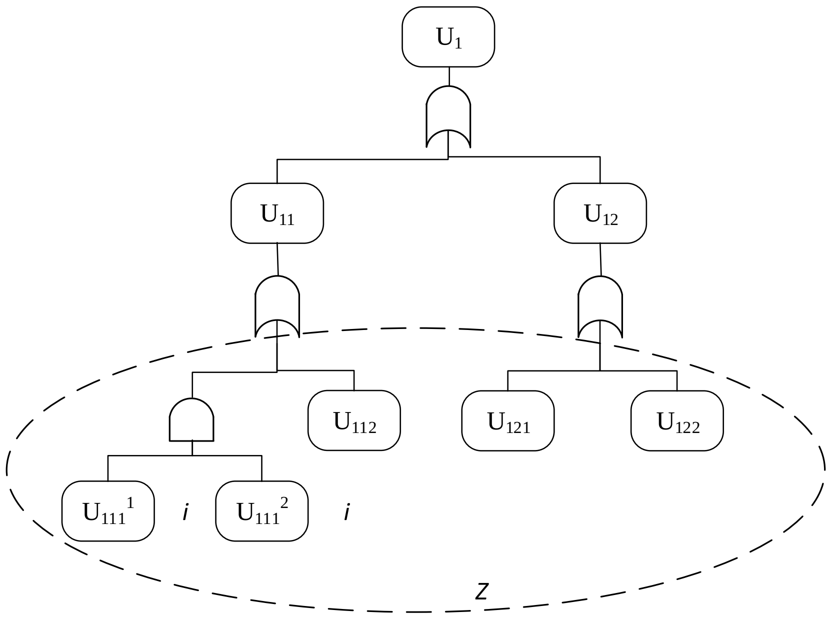


Figure 4.23: A redundant unit added to a unit at the bottom level

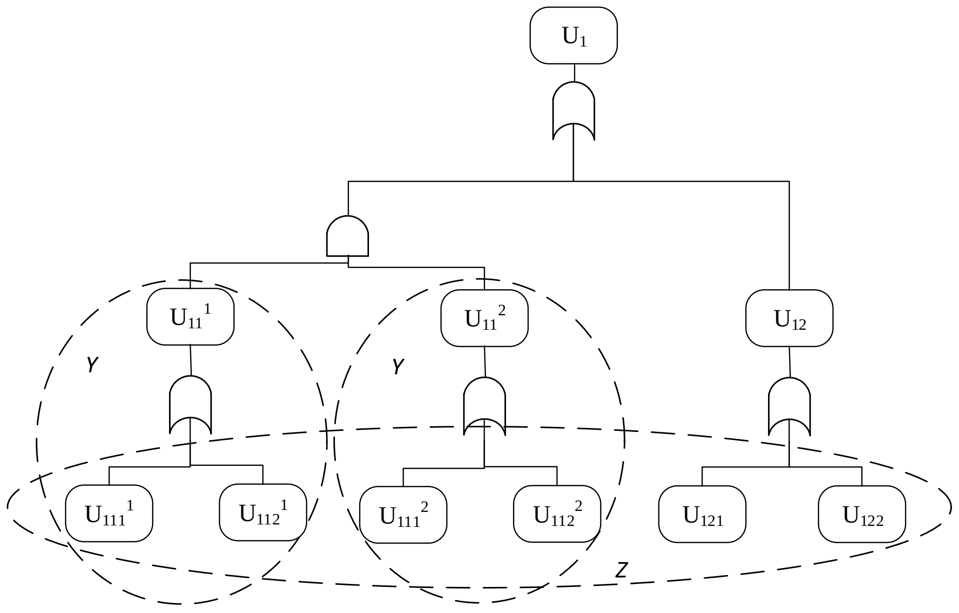


Figure 4.24: A redundant unit added to a unit at the subsystem level

$$T_{sys_i} = \begin{cases} \left(\sum_{j \in Z \& i \neq j} T_j \right) + [T_i + (\lambda_i)^2] & i \in Z, \\ \left(\sum_{j \in Z \& j \notin Y} T_j \right) + 2 \sum_{k \in Y} T_k & i \notin Z. \end{cases} \tag{4.16}$$

The formula illustrates the execution time of a multilevel redundancy allocation model, which includes two situations illustrated in Figure 4.23 and Figure 4.24. Z is the set of the units at the bottom level.

To measure the influence of a redundant unit, the reliability, the cost and the execution time should be taken into consideration. Therefore, the weights in the wheel selection method described can be calculated as follows.

$$w_i = \begin{cases} w_{i-1} + \dfrac{\frac{\Delta R_{sys_i}}{w_1 \Delta C_{sys_i} + w_2 \Delta T_{sys_i}}}{\sum_{j=1}^{N} \frac{\Delta R_{sys_j}}{w_1 \Delta C_{sys_j} + w_2 \Delta T_{sys_j}}} & 1 \leq i \leq N, \\ 0 & i = 0. \end{cases} \tag{4.17}$$

Where,

$$\frac{\Delta R_{sys_i}}{w_1 \Delta C_{sys_i} + w_2 \Delta T_{sys_i}} = \frac{R_{sys_i} - R_{sys}}{w_1 \left(C_{sys_i} - C_{sys} \right) + w_2 \left(T_{sys_i} - T_{sys} \right)} \tag{4.18}$$

4.5.8 Development of local search method

In this section, the local search method is described, which is responsible for generating new solutions and refining current solutions. Similar to the local search approach (He, Wu, Xu, Wen, & Jiang, 2013), the method in this work does not use genetic operators, but the linked list based array, which can save more storage space and improve the efficiency of generating new solutions.

Figure 4.25 shows the algorithm of the local search method. The iteration procedure generates the new solutions. In the iteration procedure, the increase and decrease operation of the redundancy numbers would be performed. The encoding mechanism in this work can save storage space and reduce the overhead of the movement of redundant elements.

```
Algorithm name: local_search
Input parameters: a solution $x$, sensitivity analysis values, $w\_i$, current population
                  $P0$, iteration number $n = 1$
Output parameters:   Generated solutions $Q$

1. Randomly select a unit $i$ according to $w\_i$
2. Randomly select a node number $j$ from the unit $i$
   // $\alpha_i$ is the maximal redundant value
3. Add a redundant unit to the position $x_{i,j}$ to generate a new solution $x'$, if
   $val(x_{i,j})$ is smaller than $\alpha_i$
4. Decrease a redundant unit from the position $x_{i,j}$ to generate a new solution
   $x''$, if $val(x_{i,j})$ is greater than 1
5. Add $x'$ and $x''$ to $Q$
   // $M\_I\_N$ is the maximal exploration value
6. if $n < M\_I\_N$ {
7.       local_search($x', w_i, P0, n+1$)
8.       local_search($x'', w_i, P0, n+1$)
9. return $Q$
```

Figure 4.25: The algorithm of the local search method

Figure 4.26 shows the operation of generating new solutions as an example of the local search method. In the figure, the index 4 is selected according to the wheel selection method, which would be depicted in the following section. Therefore, the unit, U_{13} is selected. Since there is only one node in the corresponding linked list, the value of the node is added by 1 and subtracted by 1 shown in Figure 4.26.

After the increase and decrease operation, the redundancy number of its child units would be performed. For the increase operation, a node is inserted to the end of the linked list of the child units, U_{131} and U_{132}, respectively. For the decrease operation, a node in the linked list is randomly selected. In the example, the first node is chosen and removed. The new encoding mechanism allows the change of the links in order to avoid the movement of other nodes.

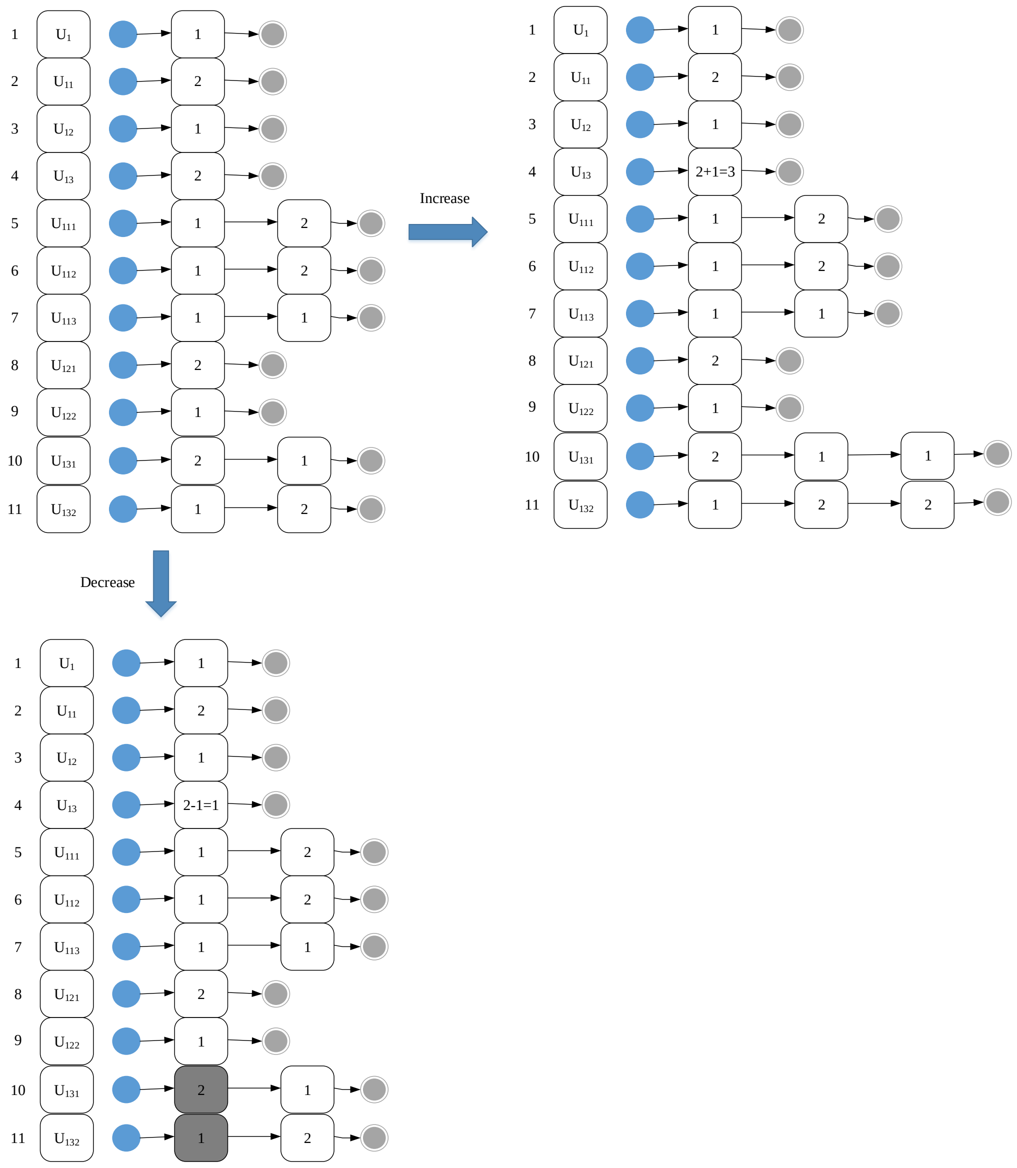


Figure 4.26: The increase and decrease operation of generating new solutions

4.5.9 Development of MLRAP with RB, Parallel and NVP

In this section, the formulas of RB, Parallel and NVP are described in terms of the reliability, the cost and the execution time. In this work, the strategies can be integrated with MLRAP for the fault tolerance of the components at the bottom level of a model.

Table 4.1: The formulas of the fault tolerance strategies

Fault tolerance strategies	Formulas
MLRAP(He, Wu, Xu, Wen, & Jiang, 2013)	$R_i = \prod_{k=1}^{m_i}\left[1 - \prod_{j=1}^{x_{ik}}\left(1 - R_{ik}^{j}\right)\right]$ $C_i = \sum_{k=1}^{m_i}\left(\sum_{j=1}^{x_{ik}} C_{ik}^{j} + (\partial_{ik})^{x_{ik}}\right).$ (4.19) $T_i = \sum_{j=1}^{m}\left(\max\left(t_{ij}^{k}\right) + \lambda^{\lvert M_k\rvert}\right), k\epsilon Y.$
RB(Bonvin, Papaioannou, & Aberer, 2010)	$R_i = \prod_{k=1}^{m_i}\left[1 - \prod_{j=1}^{x_{ik}}\left(1 - R_{ik}^{j}\right)\right].$ $C_i = \sum_{k=1}^{m_i}\left(\sum_{j=1}^{x_{ik}} C_{ik}^{j} + (\partial_{ik})^{x_{ik}}\right).$ (4.20) $T_i = \sum_{j=1}^{m}\left(\sum_{i=1}^{n} t_{ij} \prod_{k=1}^{i-1}(1 - R_{i,j}) + \lambda^{\lvert M_k\rvert}\right).$
Parallel(Qiu, Zheng, Wang, Yang, & Lyu, 2014)	$R_i = \prod_{k=1}^{m_i}\left[1 - \prod_{j=1}^{x_{ik}}\left(1 - R_{ik}^{j}\right)\right].$ $C_i = \sum_{k=1}^{m_i}\left(\sum_{j=1}^{x_{ik}} C_{ik}^{j} + (\partial_{ik})^{x_{ik}}\right).$ (4.22) $T_i = \sum_{j=1}^{m}\left(\min\left(t_{ij}^{k}\right) + \lambda^{\lvert M_k\rvert}\right), k\epsilon Y.$
NVP(Brin & Page, 1998)	$R_i = \prod_{k=1}^{m_i} \sum_{i=(n+1)/2}^{n} F(i).$ $C_i = \sum_{k=1}^{m_i}\left(\sum_{j=1}^{x_{ik}} C_{ik}^{j} + (\partial_{ik})^{x_{ik}}\right).$ (4.23) $T_i = \sum_{j=1}^{m}\left(\max\left(t_{ij}^{k}\right) + \lambda^{\lvert M_k\rvert}\right), k\epsilon Y.$

In Table 4.1, the formulas of MLRAP, RB, Parallel and NVP are illustrated. Therefore, the reliability, the cost and the execution time of a multilevel redundancy allocation model can be computed based on the formulas.

4.6 Summary

In this section a migration and deployment framework for service oriented systems is proposed, which is different from previous research, for it considers the execution time and global constraints. The framework can guide the designers and the administrators to migrate their applications. The genetic algorithm integrated in the framework can be utilised to compute the optimal solution, namely, the optimal migration plan. A service oriented system is described as a tree-like structure. The redundancy can be allocated to each node of the structure.

The migration and deployment framework is utilised to generate the optimal migration plan for the service oriented systems in clouds. The designers need to offer the reliability, the execution time and the cost of all the services. Thereafter, the framework automatically computes the available plans and after a certain number of generations, the optimal plan can be selected as the final result sent to them.

The improved genetic algorithm based on (He, Wu, Xu, Wen, & Jiang, 2013) is utilised in the migration and deployment framework in clouds. The multilevel redundancy allocation model has not been employed for service oriented systems before. This work considers it for the migration and deployment of the services in clouds. At first, the model is employed to describe the structure of the systems. Thereafter, the genetic algorithm is utilised to compute the optimal solutions under the predefined constraints. However, the genetic algorithm needs to take the execution time into consideration.

In this work, the linked list based encoding mechanism is proposed to improve the performance of the genetic algorithm. The genetic algorithm attempts to find the possible and available solutions in predefined number of iteration. In each generation, it needs to decrease and increase the redundant units. Therefore, the execution time of the genetic algorithm is mainly influenced by the decrease and increase operations. Therefore, the new mechanism can enhance the performance of the entire genetic algorithm. In this work, the linked list based encoding mechanism is utilised, which describe the multilevel redundancy allocation model via a linked list based array. In each generation, the nodes of the linked list do not need to be moved forward or backword. The only operation is to locate the node in a linked list and modify the pointers or change its value.

Chapter 5 Design of Experiments

5.1 Introduction

In this chapter, the test bed design and implementation are described. The genetic algorithm is implemented in Java programming language, which is not related to the platforms. Once the Java programs are developed, they can be executed in various operating systems. In terms of the integrated development environment (IDE), Eclipse is selected to edit Java programs.

Two examples of the multilevel redundancy allocation models are utilised to test the genetic algorithm in the migration and deployment framework, which are always employed to evaluate the performance of the algorithms (He, Wu, Xu, Wen, & Jiang, 2013; Wang, Tang, & Yao, 2010).

5.2 Test bed design and implementation

5.2.1 Objectives and requirements

In this section, the objectives and the requirements of the empirical studies are described. Detailed descriptions of the objectives and the requirements are as follows.

- The implementation of the migration and deployment framework: To maximise the compatibility of the framework, Java programming language is selected, which allows the developed applications to execute on various platforms. With the consideration of end users, the framework needs to work on different operating systems.

- The implementation of the genetic algorithm: The algorithm with the new encoding mechanism has been implemented according to the design described in the previous sections. The genetic algorithm also uses Java programming language to implement all the functions and methods.

The comparison of MLRAP, MLRAP with RB, Parallel and NVP is performed in terms of reliability under the cost and execution time constraints. The genetic algorithm is implemented and integrated with RB, Parallel and NVP separately.

5.2.2 Development tools

5.2.2.1 The tools for the migration and deployment framework

The section describes the tools for the migration and deployment framework. Here, the framework includes the implementation of the framework, the genetic algorithm and the encoding mechanism proposed in this work. At first, Java programming language is selected for the development of the framework. The advantages of the utilisation of this programming language are described as follows.

- The compatibility of the programs: The programs developed in Java programming language can be implemented once and executed anywhere. Since the end users may have various operating systems (such as, Windows, Mac OS, Linux, etc.), the framework will need to work on them. Therefore, Java programming language may be the optimal choice. In this work, the framework, the genetic algorithm and the encoding mechanism are all implemented in Java programming language.

- The packaging of the future service: There are some existing applications in the lab and the staff are very familiar with the Java programming language. Hence, the maintenance and the improvement of the framework can be completed with ease. Additionally, different client applications may be designed and developed in future and the framework may be packaged as Web services. In this case, the existing services based on Java programming language can be invoked or integrated with the packaged framework.

- Scientific computing support: Java programming language supports scientific computing well. Some classes are designed in JDK to set up programs in the field of scientific computing. For example, Big Decimal is an internal class defined in the JDK to represent exact numbers. In this work, reliability, the execution time and the cost need to be computed and some operations are performed based on Big Decimal, which needs exact computing. Additionally, the new encoding mechanism utilises the linked list data structure. All the units of a solution with redundancy need to be represented by one array. Therefore, the class ArrayList in JDK can be directly utilised to denote the solutions of the genetic algorithm. The Java programming language has already some functions to operate ArrayList, which can make the programs concise and save the developers' time in order to make them focus on the algorithm.

Based on these reasons, Java programming language is employed for the implementation of the migration and deployment framework. Once the programming language is confirmed, the tools for Java will be utilised. At first, JDK 8u111 is selected, which was the latest version when this work was undertaken.

After the selection of the foundation of Java applications, the integrated development environment (IDE) should be taken into account. With the consideration of the utilised operating systems, Eclipse IDE for j2ee developer (Europa packages) is chosen, which is free and can be installed on Windows, Mac OS and Linux platforms.

5.2.2.2 The tools for service oriented systems and the cloud

In this section, the tools for the service oriented systems and the cloud computing platform are described. Similar to the description of the tools for the implementation of the migration and deployment framework, the programming language is firstly selected for the service oriented systems. Thereafter, the tools are expanded based on it.

Two programming languages are selected for the systems. One is Java programming language and the other one is PHP programming language. The advantages of Java programming language are described in the previous section. Therefore, the reasons for the selection of PHP programming language are described as follows.

- The utilisation is easy: Knowledge of PHP programming language can be learnt with ease. Additionally, it is open source and examples can be found on the Internet.
- The support from some open source tools: PHP programming language is popular and there are a lot of tools to support the development of applications based on PHP. For example, XAMPP including the tools (such as, Apache, MySQL, PHP, etc.) can be installed and utilised with ease on various operating systems.

Due to these reasons, PHP is selected for the implementation of the service oriented systems. The IDE, Eclipse for PHP developers, is selected. The selection reasons for the IDE are the same as Java programming language. In terms of the execution environment, XAMPP is utilised, which is the most popular PHP development environment. This tool is free and easy to install on various operating systems.

Since some services of the service oriented systems are developed with Java, the execution environment, Axis2, is selected for the service execution, which is a Web service, SOAP and WSDL engine developed by the Apache software foundation (Zheng, Zhang, & Lyu, 2014). The plugins, service archive generator and code generator wizard, can be integrated with the IDE Eclipse in order to automatically generate the Web services according to Java source code.

For the setup of the cloud computing platform, the tools provided by VMware Company are selected. In this work, ESXi is utilised as the hypervisor of the VMs to be deployed on each physical server. The vCenter server as a management platform is utilised to manage all the ESXi hosts. Through the vCenter server, the computing, networking and storage resources can be collected and configured. The operating systems utilised for the VMs of the cloud computing platform are Windows Server 2016 and Ubuntu 16.04.1.

5.3 Problem-A and Problem-B

In this section, the test bed design is described based on two multilevel redundancy allocation models shown in and , which are utilized in the works (He, Wu, Xu, Wen, & Jiang, 2013; Kumar, Izui, Masataka, & Nishiwaki, 2008; Wang, Tang, & Yao, 2010). The testing of all the fault tolerance strategies in this work is performed based on them.

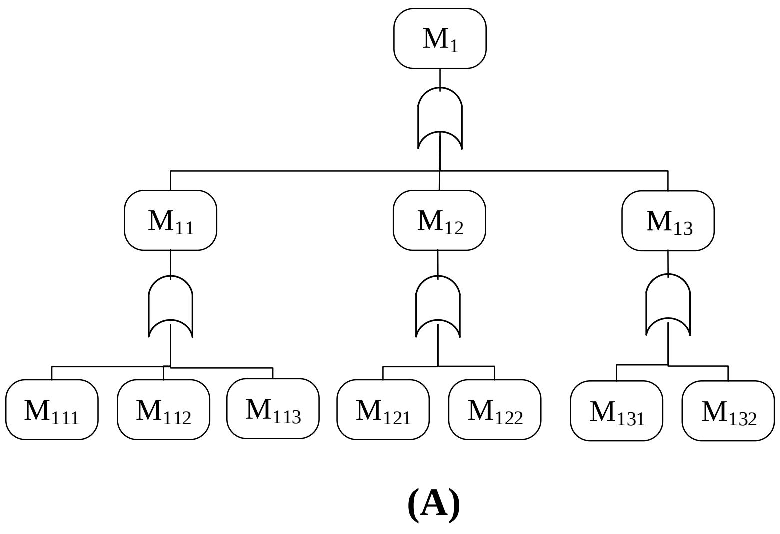
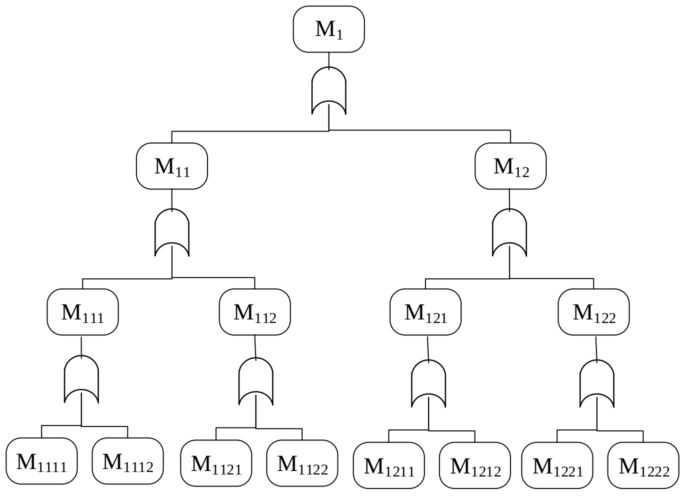


Figure 5.1: The multilevel configuration of Problem-A



(B)

Figure 5.2: The multilevel configuration of Problem-B

The first system includes three levels and the second system comprises four levels, which are referred to as Problem-A and Problem-B. Both of them have been utilised by (He, Wu, Xu, Wen, & Jiang, 2013; Wang, Tang, & Yao, 2010). Table 5.1 lists the input parameters of the units in Problem-A and Problem-B utilized in this thesis.

Table 5.1: The input parameters of the units in Problem-A and Problem-B

Module	**Reliability**	**Cost**	**∂**	**Execution time**	**λ**
Problem-A					
M_{111}	0.9	5	3	(4,5,6)	3
M_{112}	0.95	6	4	(5,6,7)	4
M_{113}	0.85	5	4	(4,5,6)	4
M_{121}	0.85	7	4	(6,7,8)	4
M_{122}	0.9	6	4	(5,6,7)	4
M_{131}	0.9	8	3	(7,8,9)	3
M_{132}	0.8	7	4	(6,7,8)	4
Problem-B					
M_{1111}	0.9	7	4	(6,7,8)	4
M_{1112}	0.8	6	4	(5,6,7)	4
M_{1121}	0.75	8	4	(7,8,9)	4
M_{1122}	0.7	9	4	(8,9,10)	4
M_{1211}	0.95	5	4	(4,5,6)	4
M_{1212}	0.9	6	4	(5,6,7)	4
M_{1221}	0.85	5	4	(4,5,6)	4
M_{1222}	0.8	8	4	(7,8,9)	4

One computer is exploited with the parameters, Intel Celeron CPU 1007U 1.5 GHz and RAM 4.00GB. The maximum generation number is set to 500. The maximum redundancy number for each unit at different layer is set to 5 and the population size is set to 100. The local search rate is set to 0.2. The iteration number in the local search method is set to 3.

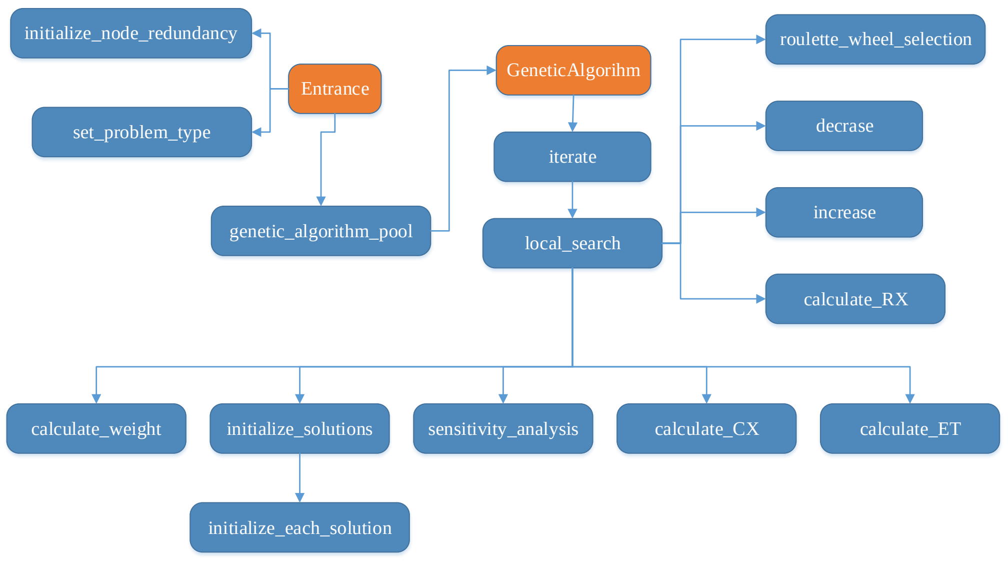


Figure 5.3: The classes and functions of the genetic algorithm

Figure 5.3 shows the classes and functions of the genetic algorithm. Two classes, Entrance and GeneticAlgorithm, are created. Detailed descriptions of the functions are illustrated as follows.

- Entrance: This is a Java class, which is the entrance of the migration and deployment framework. All the predefined parameters and constraints are sent to the specific genetic algorithms via this class.

- GeneticAlgorithm: This class is the implementation of the genetic algorithm in this work. More genetic algorithms can be integrated with the framework and each algorithm is regarded as a Java class.

- initialise_node_redundancy: The storage space of the initial solutions is generated by this method, which is the foundation of the new solutions.

- set_problem_type: The different multilevel redundancy allocation models can be set in this method. After the model is predefined, the structure of the models represented by an array is sent to the genetic algorithm.

- genetic_algorithm_pool: This method is the access interface for different genetic algorithms. It means that if a new genetic algorithm is added, it should be registered in this method.

- iterate: This method is the core of the genetic algorithm and each algorithm needs to have the similar method. Since all the genetic algorithms attempt to generate more solutions, this method can manage their iterations.

- local_search: This method is utilised to invoke other methods. The detailed function has been described in the previous sections.

- roulette_wheel_selection: This method is exploited to randomly select a unit from the structure of the multilevel redundancy allocation model according to the value computed by the sensitivity_analysis method.

- calculate_weight: This method can calculate the weights of all the units of the multilevel redundancy allocation models for the roulette_wheel_selection method.

- decrease: After the selection of a unit of the model via the roulette_wheel_selection method, the decrease method can operate the structure in order to generate a new solution.

- increase: Similar to the decrease method, the increase method can generate a new solution via the increase operation of a selected unit.

- calculate_RX: A solution can be inputted into the method to generate its reliability.

- calculate_CX: This method is utilised to compute the cost of a solution.

- calculate_ET: This method is adopted for the computing of the execution time of a solution.
- initialise_each_solution: Since the genetic algorithm needs to initialise a set of solutions before the operation of the local_search method, this method is exploited to randomly generate a solution.
- initialise_solutions: This method invokes the initialise_each_solution method to generate the predefined number of solutions as the fundamental ones.
- sensitivity_analysis: This is the sensitivity analysis method and the detailed description of the method is performed in the previous sections.

Chapter 6 Results and Discussions

6.1 Results

In this section, the experiments of the genetic algorithm are performed and the comparison between different algorithms is shown.

6.1.1

6.1.1 Performance comparison of encoding mechanisms

In this section, the performance of traditional and linked list based encoding mechanisms has been compared in terms of increase and decrease operations on problem A and B. In the experiment, $\boldsymbol{w}$ is set to $\mathbf{0.5}$. In the following graphs, traditional mechanism is the two dimensional array based encoding mechanism and the new mechanism is proposed in this thesis, linked list based encoding mechanism.

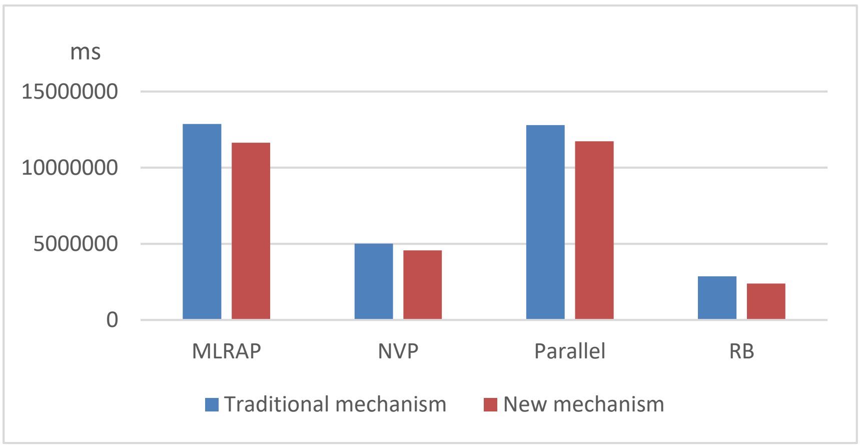


Figure 6.1: Performance comparison of increase operation on Problem-A

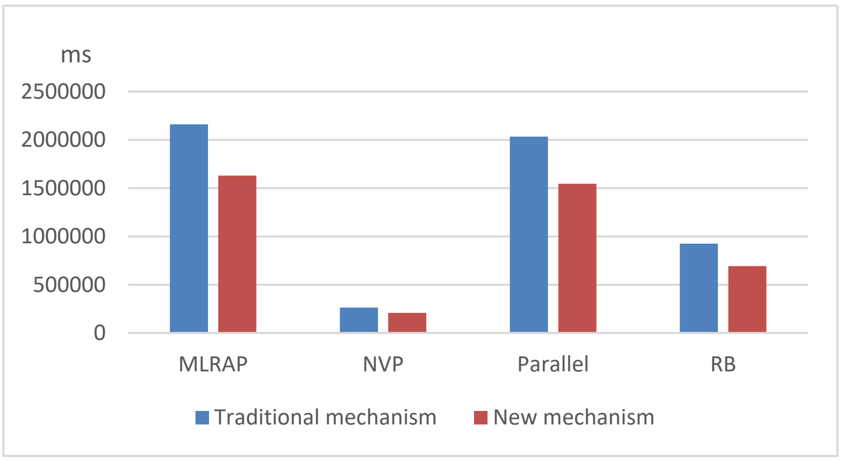


Figure 6.2: performance comparison of decrease operation on Problem-A

Figure 6.1 and Figure 6.2 show the experimental data on Problem-A. In all algorithms, traditional mechanism needs more time than the new mechanism to complete the increase and decrease operations.

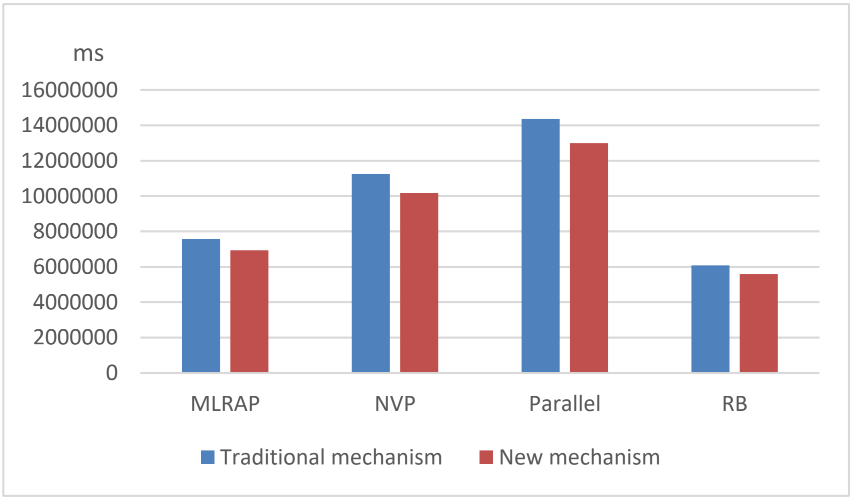


Figure 6.3: Performance comparison of increase operation on Problem-B

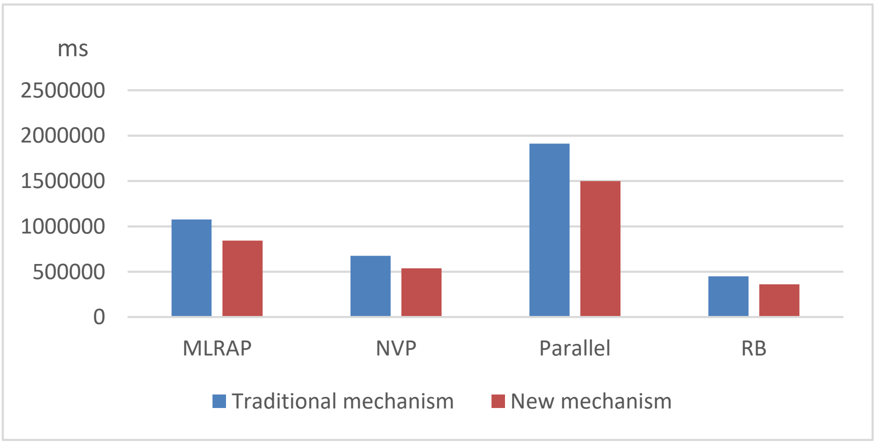


Figure 6.4: Performance comparison of decrease operation on Problem-B

Figure 6.3 and Figure 6.4 show the experimental data on Problem-B. In all algorithms, traditional mechanism needs to spend more time to increase and decrease nodes in multilevel redundancy allocation models.

6.1.2 Experimental results on Problem-A

In this section, the experiment results for Problem-A are shown. The cost and time constraints are set to 300. w is set to 0.4, 0.5 and 0.6. The generation is set to 500. The reliability of MLRAP, MLRAP with NVP, Parallel and RB are compared.

The purpose of the experiment is to get the reliability of the algorithms with different values of w.

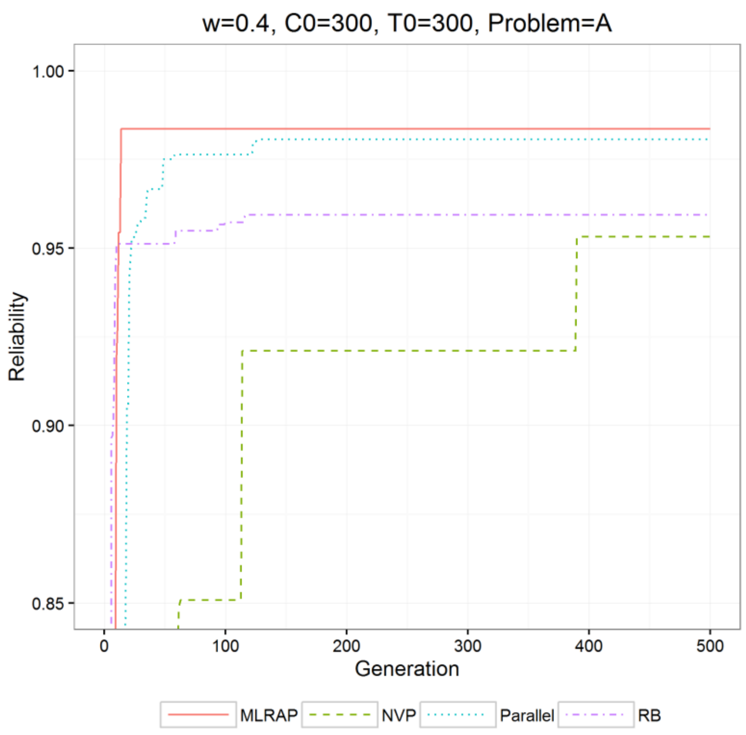


Figure 6.5: Reliability on Problem-A (w=0.4)

Figure 6.5 shows the experimental data on Problem-A when w is set to 0.4. MLRAP, Parallel and RB can get convergence, namely, achieve the optimal solution within 150 generations, while NVP needs about 400 generations to complete the convergence. Additionally, the optimal solutions that MLRAP and Parallel generate are higher than RB and NVP. The highest reliability is about 0.9835. MLRAP can get the optimal solution within 500 generations.

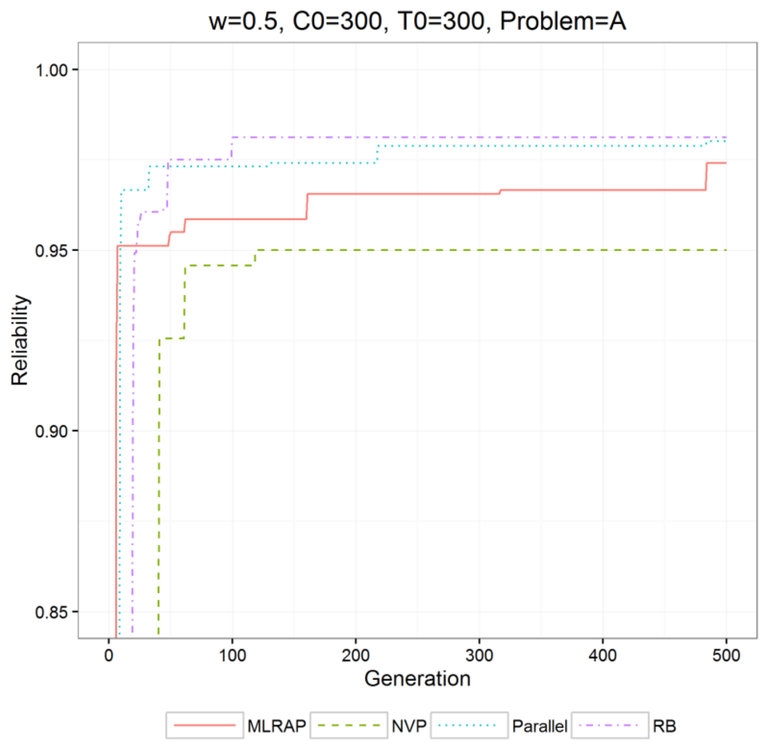


Figure 6.6: Reliability on Problem-A (w=0.5)

Figure 6.6 presents an overview of the performance of the four algorithms on Problem-A when w is set to 0.5. In this situation, RB and Parallel can get the optimal solutions with higher reliability, 0.9811. MLRAP can get a similar result, but it needs more generations. In the experiment, about 500 generations are needed. NVP still gets the lowest reliability. What is interesting in this data is that the algorithms tend to converge after 150 generations.

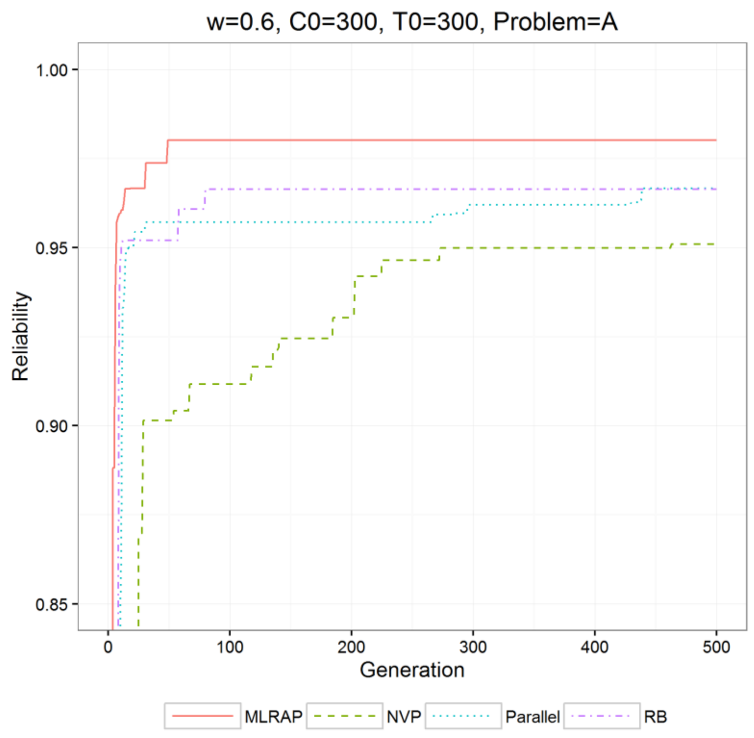


Figure 6.7: Reliability on Problem-A (w=0.6)

Figure 6.7 compares the reliability of the four algorithms when w is set to 0.6. MLRAP and RB can get convergence within 100 generations, while NVP and Parallel need more than 250 generations. In this experiment, Parallel needs more than 400 generations in order to get the optimal solution. MLRAP can get the highest reliability, 0.9801.

6.1.3 Experimental results on Problem-B

In this section, the experiment results for Problem-B are provided. Since Problem-B has a more complex structure than Problem-A, the cost and time constraints are set to 500. w is set to 0.4, 0.5 and 0.6. The generation is set to 500. The reliability of MLRAP, MLRAP with NVP, Parallel and RB are compared. The purpose of the experiment is to get the reliability of the algorithms on Problem-B with different values of w.

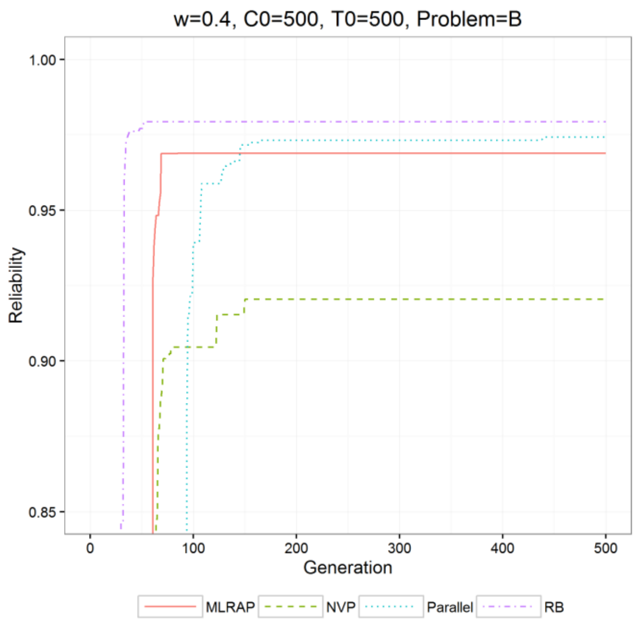


Figure 6.8: Reliability on Problem-B (w=0.4)

Figure 6.8 shows the results obtained from the performance experiment on Problem-B. All the four algorithms can get convergence within 150 generations. RB and MLRAP need fewer generations to get the optimal solution. In this experiment, NVP still cannot get higher reliability than other algorithms. The highest reliability is obtained by RB and it is 0.9792.

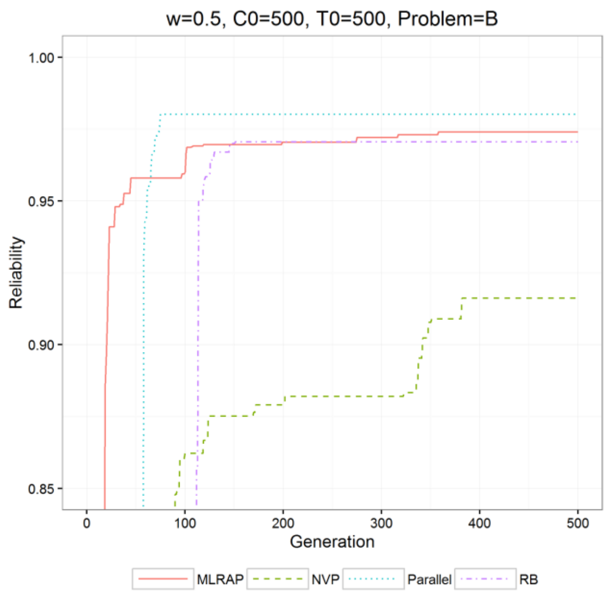


Figure 6.9: Reliability on Problem-B (w=0.5)

When w is set to 0.5, the execution time and cost have identical weights on the genetic algorithm. Figure 6.9 presents some of the main characteristics of the convergence of the four algorithms. It is apparent from this figure that MLRAP, Parallel and RB can get higher reliability than 0.95. NVP only gets the reliability 0.928 within 400 generations. In terms of reliability, Parallel can obtain the highest reliability in the experiment.

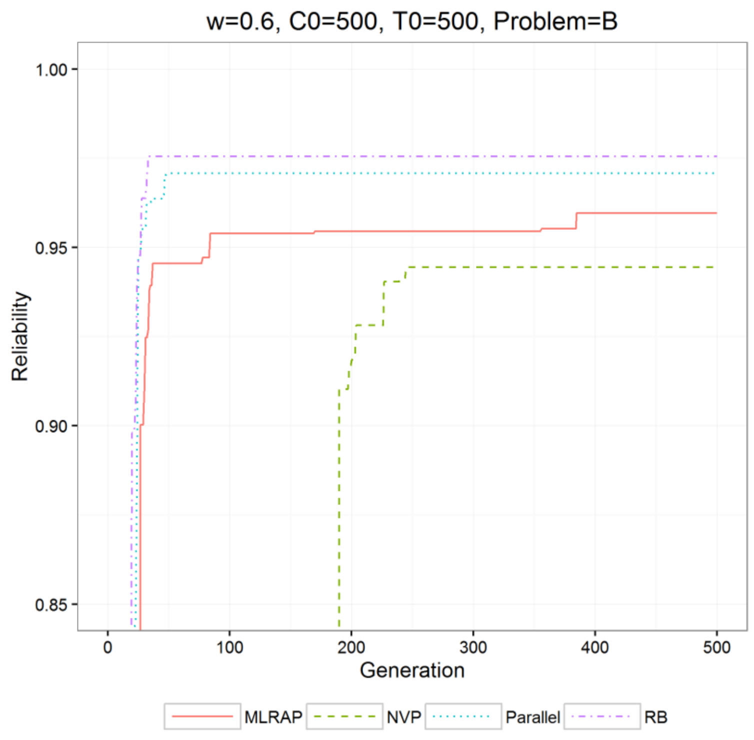


Figure 6.10: Reliability on Problem B (w=0.6)

As shown in Figure 6.10, RB and Parallel get more significant reliability than other algorithms. Additionally, they need fewer than 100 generations to get the optimal solutions. Although MLRAP can get higher reliability than 0.95, it needs about 400 generations. Similar to the above experimental results, NVP needs more generations to generate the optimal solution.

6.1.4 Experimental results on Problem-A and Problem-B

The value of w can influence the performance of the genetic algorithm. Therefore, different values are set to w in the experiment for the four algorithms on Problem-A and Problem-B.

6.1.4.1 $w = 0$

When w is set to 0, only the execution time is taken into consideration in the genetic algorithm. Therefore, the cost constraint cannot influence the generated solutions.

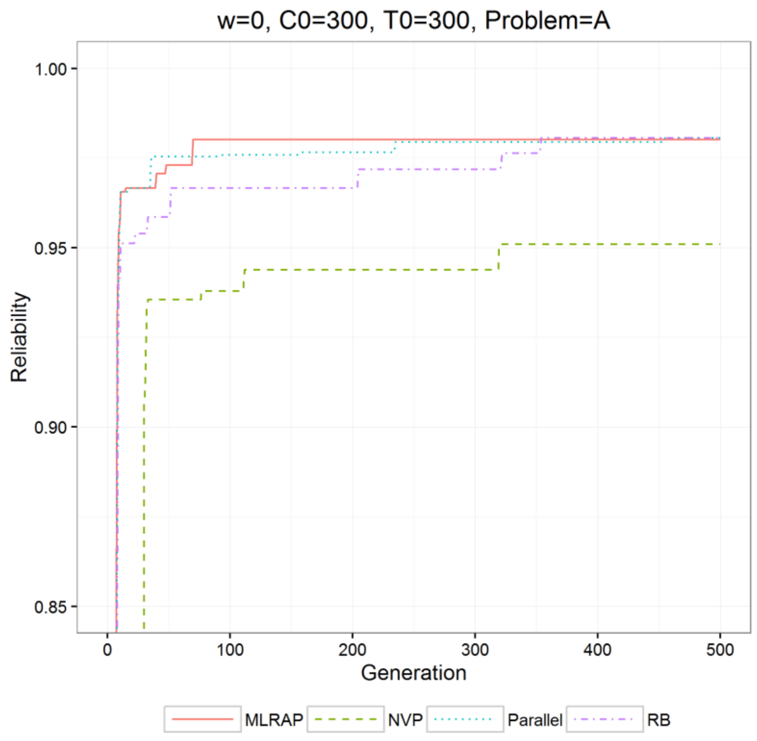


Figure 6.11: Reliability on Problem-A (w=0)

Figure 6.11 shows the experimental data on Problem-A when w is set to 0. MLRAP and Parallel can get convergence within 300 generations, while RB and NVP need more than 320 generations to get the optimal solutions. In the experiment, MLRAP, Parallel and RB can obtain the highest reliability, 0.9805. Compared with MLRAP and Parallel, RB needs more generations to achieve the identical reliability.

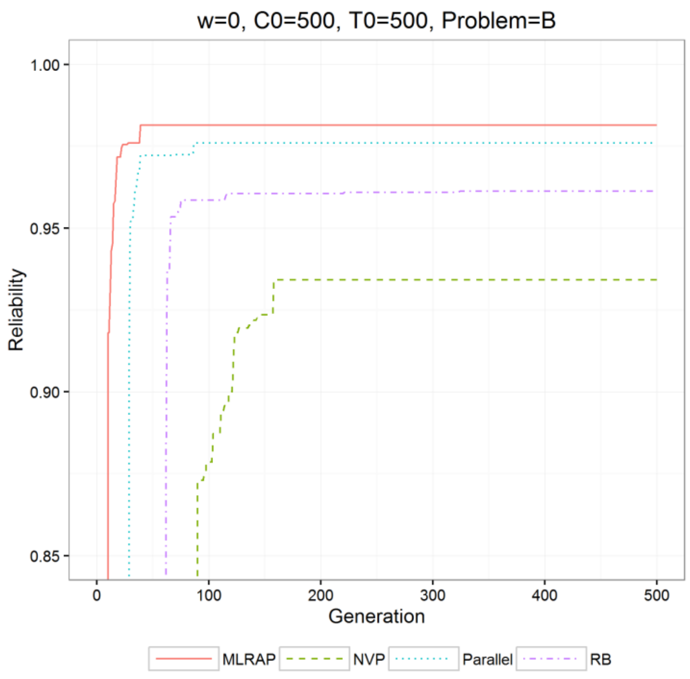


Figure 6.12: Reliability on Problem B (w=0)

As shown in Figure 6.12, MLRAP, Parallel and RB can get convergence within 100 generations compared with NVP. The experimental results on the performance of the four algorithms present that MLRAP can generate the highest reliability.

6.1.4.2 $w = 1$

When w is set to 1, only the cost is considered in the genetic algorithm. The following is to describe the performance of the four algorithms on Problem-A and Problem-B.

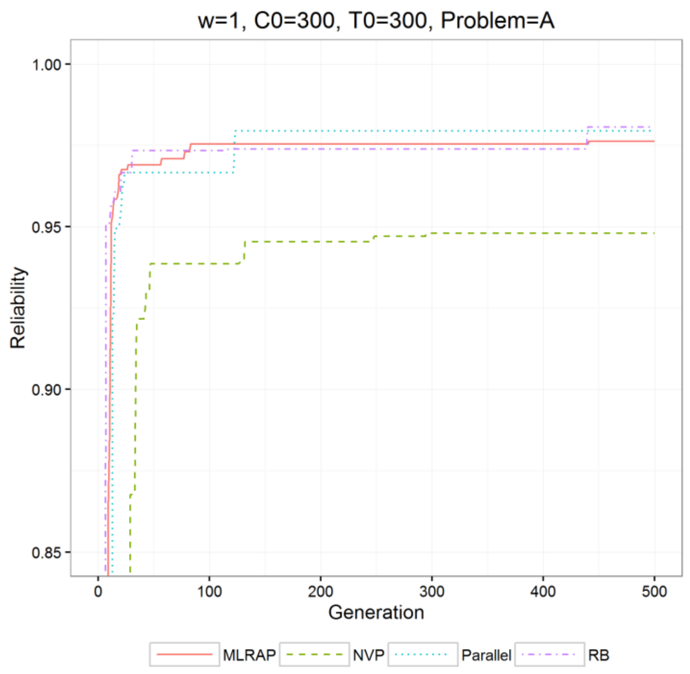


Figure 6.13: Reliability on Problem-A (w=1)

Figure 6.13 compares an overview of the reliability of the four algorithms on Problem-A when $w = 1$. Parallel and RB can get the highest reliability compared to MLRAP and NVP. What is interesting in this data is that both of them need about 400 generations to generate the optimal solutions.

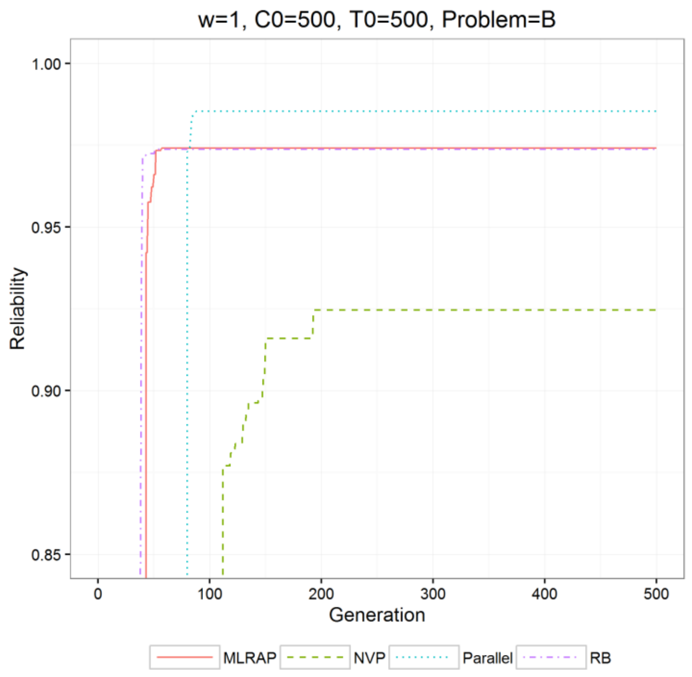


Figure 6.14: Reliability on Problem B (w=1)

The results shown in Figure 6.14 present that Parallel can generate the optimal solution with the highest reliability. It is apparent from this figure that Parallel almost directly generates the optimal solution. Compared with Parallel, MLRAP and RB can generate the solution with identical reliability. These algorithms can get convergence within 100 generations.

6.1.4.3 $\boldsymbol{w = 0.5}$ and $\boldsymbol{K = 3}$

The parameter K can influence the fitness function so as to filter the generated solutions in each generation. In the experiment, K and w are set to 3 and 0.5, respectively.

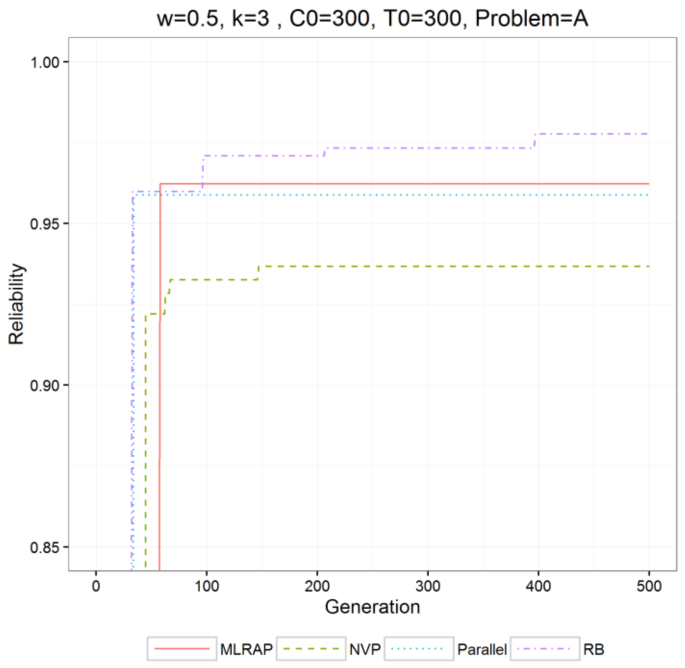


Figure 6.15: Reliability on Problem-A (w=0.5 and K=3)

Figure 6.15 shows the experimental data on Problem-A when $w = 0.5$ and $K = 3$. It is apparent that RB can generate the optimal solution with the highest reliability. However, it needs about 400 generations to get the solution. In terms of reliability, MLRAP and Parallel obtain more than 0.95.

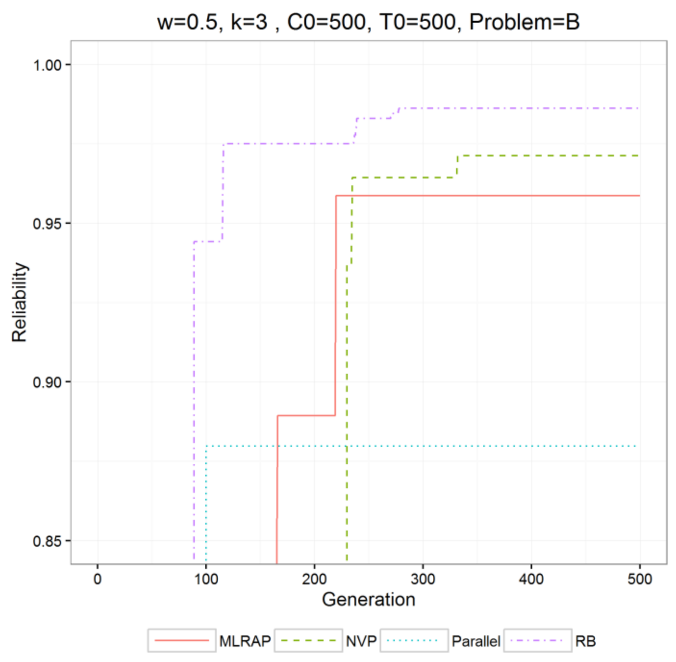


Figure 6.16: Reliability on Problem-B (w=0.5 and K=3)

Figure 6.16 provides the experimental data on the reliability for Problem-B when $w = 0.5$ and $K = 3$. All the algorithms need more generations to generate the optimal solutions.

6.1.5 Analysis

In this section, the analysis of the empirical results is described. The analysis is mainly based on Problem-A and Problem-B.

The reliability of MLRAP, MLRAP with Parallel, MLRAP with RB and MLRAP with NVP: In terms of different structures of the system, MLRAP, Parallel and RB tend to get the optimal solution with the highest reliability. Compared with the algorithms, NVP always generates the solutions with the lowest reliability. Additionally, NVP needs more generations to generate the optimal solutions whether the problem is A or B.

6.1.6 Discussion

The discussion mainly focuses on the multilevel redundancy allocation, the performance of MLRAP, MLRAP with NVP, MLRAP with RB and MLRAP with

Parallel.

In the Web service field, there are already some migration and deployment frameworks for service oriented systems. The traditional frameworks take fault tolerance approaches into consideration and arrange the redundant units to the selected top k services, which are considered to be the significant units of the applications, and their reliability may be not high. The frameworks tend to improve the reliability of the applications with redundancy allocation to some certain services. However, sometimes there may be a set of services which have relationships and need to be improved. Additionally, redundancy allocation to the entire system may need to be considered as well in order to obtain the optimal solution with the highest reliability.

This work describes the structure of service oriented systems as a tree. All the units in the model can be allocated with redundant units. The algorithm can generate the optimal solution in terms of the entire system under execution time and cost constraints. The algorithm takes more features into consideration when a service oriented system is deployed in clouds. The execution time as an important feature is exploited in order to measure the performance of the systems. Additionally, the migration and deployment of the applications needs various resources in cloud computing platforms. These resources need to be purchased. Therefore, the migration and deployment frameworks need to take the cost into consideration. The cost and execution time constraints can be predefined by the designers or the administrators. Additionally, the performance of the genetic algorithms to generate the optimal solution is always a problem. To reduce the execution time of the genetic algorithm, a new encoding mechanism is proposed based on the linked list based array. The utilisation of the array can reduce the number of the steps to generate new solutions.

Chapter 7 Conclusion and Future Work

7.1 Conclusion

In this section, a conclusion is described. In this work, a migration and deployment framework is proposed.

1) The migration and deployment framework: A migration and deployment framework is proposed in this work in order to guide designers and administrators to migrate and deploy service oriented systems into clouds. The structure of the application as a tree needs to be applied to the framework, because the genetic algorithm integrated in the framework considers the applications as a tree structure. Additionally, the related information of all the services of the systems needs to be given to the framework. The framework can generate new solutions via the increase and decrease operations of the redundant units. To achieve the optimal solution, namely, the optimal deployment plan, reliability, execution time and cost need to be taken into consideration. Therefore, the features of each service need to be given to the framework. The reliability of the services can be collected in two ways. One method is to get all the information of the errors and exceptions from the log files of servers which depict the states of the services in the actual environment. The other approach is to use a program to access each of the services a certain number of times. After the number of invocations, the reliability of the services can be computed. The migration and deployment framework takes the actual requirements of end users into consideration. The computing, networking and storage resources need to be purchased. Therefore, the migration and deployment framework needs to consider these features. End users can provide the execution time and the cost constraints to the framework. Either of these constraints can be defined for the generation of the optimal migration plan.

The execution time may represent the performance of a service oriented application. Some services or applications need quick responses to the requests from the end users. In terms of the cost, companies need to purchase resources to deploy their systems. However, if they have their own cloud computing platforms and a lot of VMs can be exploited, the cost constraint probably does not need to be considered. Therefore, the designers and the administrators using the framework in this work can select to provide both of the constraints or one of them via the defined weight. After the computing of the solutions based on the constraints, the optimal migration and deployment plan can be generated and provided to the designers and the administrators, and can clearly describe the structure of the system with redundant units.

2) The genetic algorithm: The algorithm is proposed based on the research (He, Wu, Xu, Wen, & Jiang, 2013). The multilevel redundancy allocation model is utilised for Web service based applications for the first time, which describes service oriented systems as a tree and the services are regarded as the components at the bottom level. The abstract functions are the units at the higher level and they can comprise a set of detailed services. The genetic algorithm can generate new solutions based on this model.

3) The traditional algorithms in the multilevel redundancy allocation field take reliability and resource costs into consideration. However, these two features are not sufficient to describe service oriented systems. The execution time of the services needs to be considered in the algorithm, when a solution is generated. Therefore, the algorithm in this work expands the traditional algorithms to add the execution time feature.

4) The new encoding mechanism: This is proposed in this work in order to improve the performance of the genetic algorithm. The genetic algorithm is

the core of the migration and deployment framework and the encoding mechanism is the central component of the genetic algorithm. Therefore, the encoding mechanism of the genetic algorithm can influence the framework in terms of the performance. The traditional encoding mechanism proposed in (He, Wu, Xu, Wen, & Jiang, 2013) is based on a two dimensional array, which is exploited to store all the solutions. The genetic algorithm attempts to generate new solutions according to predefined principles. The operations of generating new solutions are based on the array. During the period of the generation of solutions, the number of the redundant units is increased and decreased. Since all the units can be allocated with the redundant units in the multilevel redundancy allocation model, the units at the higher level can be arranged with the redundancies. In the model, the modification of the redundant units at the higher level can lead to a greater change of the units at the lower levels. For example, if a redundant unit is added for the one at the higher level, the structure comprising its children units would be assigned with the redundancies. In this case, the elements of the two dimensional array may be moved. Additionally, sometimes the movement would be performed from the first element of the one dimensional array to the end. The genetic algorithm mainly generates new solutions via the modification of the redundant units of the fundamental solutions.

To handle the performance problem of the genetic algorithm, a new encoding mechanism is proposed in the work. The encoding mechanism is based on a linked list data structure. A one-dimensional array is utilised to represent the structure of the solutions. In each element of the array, a linked list is created in order to denote the redundant units of each unit at different levels. Compared with the traditional mechanism, the linked list does not need to move the elements when an element is removed or a new element is added, for the only links need to be changed. Therefore, for the new encoding mechanism, there are only two steps to perform the increase and decrease operations. One is to locate the position of the element required to be changed. The other one is to change the links or the value of an element. Therefore, the mechanism can reduce the movement of other elements in order to improve the performance of the genetic algorithm.

In summary, a migration and deployment framework integrating a genetic algorithm with the new encoding mechanism is proposed in the work. The framework takes the execution time and the cost constraints into consideration. The theory of the genetic algorithm in multilevel redundancy allocation models is firstly exploited for service oriented systems. Based on the traditional genetic algorithm in the multilevel redundancy allocation model, a new encoding mechanism based on a linked list based array is proposed to improve the performance of the genetic algorithm.

7.2 Future work

In this section, possible future work is described based on the current work. Detailed descriptions of the future work are as follows.

1) More non-functional requirements: For service oriented systems, the requirements include two parts: one is functional requirements and the other one is the non-functional requirements. In this work, the non-functional requirements reliability, cost and execution time are taken into consideration. However, in the service field, there are more non-functional requirements which need to be considered, such as the response time, the reputation, etc. In future work, more features may be utilised in the framework in order to measure the quality of the entire application.

2) Deployment of the applications in different cloud computing platforms: In the real deployment of the applications, more cloud computing platforms may be exploited, which are not close to each other. In this case, more factors influence the performance of applications, especially the network situation. For example, a request from the end users is sent to a specific service and it may invoke another one in a different cloud computing platform. The worst situation is that these two services are very far apart and the rules of these cloud computing platforms may be totally different. In this case, the response time of these services may be so long that the end users could not tolerate them. Due to this phenomenon, the locations and the transfer time between the services in different cloud computing platforms may need to be considered when the service oriented systems are migrated and deployed in some cloud computing platforms.

Reference

1. Agarwal, M., Aggarwal, S., & Sharma, V. K. (2010). Optimal redundancy allocation in complex systems. Journal of Quality in Maintenance Engineering, 16(4), 413-424.
2. Agrawal, B., Chakravorty, A., Rong, C., & Wlodarczyk, T. W. (2014). R2Time: a framework to analyse open TSDB time-series data in HBase. IEEE 6th International Conference, 970-975.
3. Ahmed, W., Wu, Y., & Zheng, W. (2015). Response time based optimal Web service selection. IEEE Transactions on Parallel and Distributed Systems, 26(2), 551-561.
4. Aihkisalo, T., & Paaso, T. (2012). Latencies of service invocation and processing of the REST and SOAP Web service interfaces. IEEE Eighth World Congress on Services, 100-107.
5. Altmann, J., & Kashef, M. M. (2014). Cost model based service placement in federated hybrid clouds. Future Generation Computer Systems, 41, 79-90.
6. AlZain, M. A., Soh, B., & Pardede, E. (2013). A Byzantine fault tolerance model for a multi-cloud computing. IEEE 16th International Conference on Computational Science and Engineering (CSE), 130-137.
7. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, g., Patterson, D. A., Rabkin, A., Stoica, I., & Zaharia, M. (2009). Above the clouds: a berkeley view of cloud computing. Electrical engineering and computer sciences university of California at Berkeley, 1-25.
8. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Konwinski, R. K. A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53, 50-58.
9. Assuncao, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A. S., & Buyya, R. (2013). Big data computing and clouds: challenges, solutions, and future directions. Distributed, Parallel, and Cluster Computing, 79-80, 1-39.
10. Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A. S., & Buyya, R. (2015). Big data computing and clouds: trends and future directions. Journal of Parallel and Distributed Computing, 79-80, 3-15.
11. Avizienis, A. (1995). The methodology of N-version programming. Software fault tolerance, 23-46.
12. Baek, J., Vu, Q. H., Liu, J. K., Huang, X., & Xiang, Y. (2015). A secure cloud computing based framework for big data information management of smart grid. IEEE Transactions on Cloud Computing, 3(2), 233-244.
13. Balasubramanian, B., & Garg, V. K. (2013). Fault tolerance in distributed systems using fused data structures. IEEE Transactions on Parallel and Distributed Systems, 24(4), 701-715.

14. Baroncelli, F., Martini, B., & Castoldi, P. (2010). Network virtualization for cloud computing. Annals of telecommunications, 65(11), 713-721.

15. Beji, N., Jarboui, B., Eddaly, M., & Chabchoub, H. (2010). A hybrid particle swarm optimization algorithm for the redundancy allocation problem. Journal of Computational Science, 1(3), 159-167.

16. Beji, N., Jarboui, B., Siarry, P., & Chabchoub, H. (2012). A differential evolution algorithm to solve redundancy allocation problems. International Transactions in Operational Research, 19(6), 809-824.

17. Bendjeghaba, O., & Ouahdi, D. (2008). Multi-agent ant system for redundancy allocation problem of multi states power system. 1270-1274.

18. Bonvin, N., Papaioannou, T., & Aberer, K. (2010). A self-organized, fault-tolerant and scalable replication scheme for cloud storage. SoCC '10 Proceedings of the 1st ACM symposium on Cloud computing, 205-216.

19. Borsato, D. (2015). Cloud. Canadian Theatre Review, 162, 74-75.

20. Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, 1-10.

21. Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 30(1), 107-117.

22. Brun, Y., Bang, J. y., Edwards, G., & Medvidovic, N. (2015). Self-adapting reliability in distributed software systems. IEEE Transactions on Software Engineering, 41(8), 764-780.

23. Cai, Z., Zhao, L., Wang, X., Yang, X., Qin, J., & Yin, K. (2015). A pattern-based code transformation approach for cloud application migration. 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), 33-40.

24. Cała, J., & Watson, P. (2010). Automatic software deployment in the azure cloud. DAIS'10 Proceedings of the 10th IFIP WG 6.1 international conference on Distributed Applications and Interoperable Systems, 155-168

25. Campos, I., Fernández-del-Castillo, E., Heinemeyer, S., Lopez-Garcia, A., Pahlen, F., & Borges, G. (2013). Phenomenology tools on cloud infrastructures using OpenStack. The European Physical Journal C, 1-25.

26. Caserta, M., & Voß, S. (2014). A corridor method based hybrid algorithm for redundancy allocation. Journal of Heuristics, 22, 405-429.

27. Caserta, M., & Voß, S. (2015). A discrete-binary transformation of the reliability redundancy allocation problem. Mathematical Problems in Engineering, 2015(2015), 1-6.

28. Cha, S., & Yoo, J. (2011). A safety-focused verification using software fault trees. Future Generation Computer Systems, 28(8), 1272.

29. Chai, H., & Zhao, W. (2014). Byzantine fault tolerance for services with commutative operations. SCC '14 Proceedings of the 2014 IEEE International Conference on Services Computing, 219-226.

30. Chang, B. R., Tsai, H. F., Chen, C. Y., Huang, C. F., & Hsu, H. T. (2015). Implementation of secondary index on cloud computing NoSQL database in big data environment. Scientific Programming, 2015, 1-10.

31. Chang, B. R., Tsai, H. F., Guo, C. L., & Chen, C. Y. (2015). Remote cloud data center backup using HBase and Cassandra with user-friendly GUI. 2015 IEEE International Conference on Consumer Electronics, 420-421.

32. Chao, L., Li, C., Liang, F., Lu, X., & Xu, Z. (2015). Accelerating Apache Hive with MPI for Data Warehouse Systems. IEEE 35th International Conference on Distributed Computing Systems, 664-673.

33. Chen, X., Zheng, Z., Liu, X., Huang, Z., & Sun, H. (2013). Personalized QoS-aware Web service recommendation and visualization. IEEE Transactions on Services Computing, 6(1), 35-47.

34. Cheung, L., Roshandel, R., Medvidovic, N., & Golubchik, L. (2008). Early prediction of software component reliability. ICSE '08 Proceedings of the 30th international conference on Software engineering, 111-120.

35. Choy, S., Wong, B., Simon, G., & Rosenberg, C. (2014). A hybrid edge-cloud architecture for reducing on-demand gaming latency. Multimedia Systems, 20(5), 503-519.

36. Coelho, L. d. S. (2009). Reliability–redundancy optimization by means of a chaotic differential evolution approach. Chaos, Solitons and Fractals, 41(2), 594-602.

37. Coppolino, L., Romano, L., & Vianello, V. (2011). Security engineering of SOA applications via reliability patterns. Journal of Software Engineering and Applications, 4(1), 1-8.

38. Corradi, A., Fanelli, M., & Foschini, L. (2014). VM consolidation: a real case based on OpenStack Cloud. Future Generation Computer Systems, 32, 118–127.

39. Costa, P., Pasin, M., Bessani, A. N., & Correia, M. P. (2013). On the performance of Byzantine fault-tolerant MapReduce. IEEE Transactions on Dependable and Secure Computing, 10(5), 301 - 313

40. Dave, A., Lu, W., Jackson, J., & Barga, R. (2011). CloudClustering: toward an iterative data processing pattern on the cloud. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 1132 - 1137

41. Delac, G., Silic, M., & Srbljic, S. (2015). A reliability improvement method for SOA-based applications. IEEE Transactions on Dependable and Secure Computing, 12(2), 136-149.

42. Duan, S., Peisert, S., & Levitt, K. (2014). hBFT: speculative Byzantine fault tolerance with minimum cost. IEEE Transactions on Dependable and Secure Computing(99), 1-14.

43. Dunn, D. (2007). Citrix Systems: versatile tools. CITRIX Systems Inc. COMPUTER software(75), 62-62.

44. El Kharboutly, R., & Gokhale, S. S. (2014). Efficient reliability analysis of concurrent software applications considering software architecture. International Journal of Software Engineering and Knowledge Engineering, 24(1), 43-60.

45. Fang, C., Liang, D., Lin, F., & Lin, C. (2007). Fault tolerant Web services. Journal of Systems Architecture, 53(1), 21-38.

46. Feng, X., Luo, X., & Jin, Y. (2012). An acceleration system for long distance live migration of virtual machine. 10th International Conference on Optical Internet (COIN), 10 - 11

47. Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., & Govindan, R. (2013). Reducing web latency: the virtue of gentle aggression. Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, 43, 159-170.

48. Forster, M., & Schneider, D. (2010). Flexible, any-time fault tree analysis with component logic models. IEEE 21 st International Symposium on Software Reliability Engineering, 51-60.

49. Fyffe, D. E., Hines, W. W., & Lee, N. K. (1968). System reliability allocation and a computational algorithm. IEEE Transactions on Reliability, R-17(2), 64-69.

50. Garraghan, P., Moreno, I. S., Townend, P., & Xu, J. (2014). An analysis of failure-related energy waste in a large-scale cloud environment. IEEE Transactions on Emerging Topics in Computing Journal Article, 2(2), 166-180.

51. Garraghan, P., Townend, P., & Xu, J. (2011). Byzantine fault-tolerance in federated cloud computing. IEEE 6th International Symposium on Service Oriented System Engineering (SOSE), 280 - 285

52. Gen, M., & Cheng, R. (1996). A survey of penalty techniques in genetic algorithms. Proceedings of IEEE Inernational Conference on Evolutionary of Computation, 804-809.

53. Ha, C., & Kuo, W. (2005). Multi-path approach for reliability-redundancy allocation using a scaling method. Journal of Heuristics, 11(3), 201-217.

54. Ha, C., & Kuo, W. (2006a). Multi-path heuristic for redundancy allocation: the tree heuristic. IEEE Transactions on Reliability, 55(1), 37-43.

55. Ha, C., & Kuo, W. (2006b). Reliability redundancy allocation: an improved realization for nonconvex nonlinear programming problems. European Journal of Operational Research, 171(1), 24-38.

56. Hasanzadeh Mofrad, M., Jalilian, O., Rezvanian, A., & Meybodi, M. R. (2016). Service level agreement based adaptive grid superscheduling. Future Generation Computer Systems, 55, 62-73.

57. Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Ullah Khan, S. (2015). The rise of big data on cloud computing: review and open research issues. Information Systems, 47, 98-115.

58. He, P., Wu, K., Wen, J., & al, e. (2010). Improved memetic algorithm for multilevel redundancy allocation IEEE transactions on Reliability, 59, 754-765.

59. He, P., Wu, K., Xu, J., Wen, J., & Jiang, Z. (2013). Multilevel redundancy allocation using two dimensional arrays encoding and hybrid genetic algorithm. Computers and Industrial Engineering, 64(1), 69-83.

60. Hsieh, H. C., & Chiang, M. L. (2011). A new solution for the Byzantine agreement problem. Journal of Parallel and Distributed Computing, 71(10), 1261-1277.

61. Hsieh, T. (2014). Hierarchical redundancy allocation for multi-level reliability systems employing a bacterial-inspired evolutionary algorithm. Information Sciences, 288, 174-193.

62. Hsieh, T., & Yeh, W. (2012). Penalty guided bees search for redundancy allocation problems with a mix of components in series-parallel systems. Computers and Operations Research, 39(11), 2688-2704.

63. Hu, P. (2015). The cooperative study between the Hadoop big data platform and the traditional data warehouse. The Open Automation and Control Systems Journal, 7(1), 1144-1152.

64. Huang, D., Yi, L., Song, F., Yang, D., & Zhang, H. (2014). A secure cost-effective migration of enterprise applications to the cloud. International Journal of Communication Systems, 27(12), 3996-4013.

65. Huang, G., Wang, W., Liu, T., & Mei, H. (2011). Simulation-based analysis of middleware service impact on system reliability: Experiment on Java application server. The Journal of Systems & Software, 84(7), 1160-1170.

66. Huang, S., Wang, B., Zhu, J., Wang, G., & Yu, G. (2014). R-HBase: a multi-dimensional indexing framework for cloud computing environment. IEEE International Conference on Data Mining Workshop, 569-574.

67. Huynh, T., & Miller, J. (2009). Another viewpoint on evaluating web software reliability based on workload and failure data extracted from server logs. Empirical Software Engineering, 14(4), 371-396.

68. Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., & Kusumoto, S. (2005). Ranking significance of software components based on use relations. IEEE Transactions on Software Engineering, 31(3), 213 - 225.

69. Jais, M. K. (2015). Redefining reliability evaluations for software-intensive systems. Annual Reliability and Maintainability Symposium, 1-4.

70. Jung, G., & Sim, K. M. (2011). Agent-based adaptive resource allocation on the cloud computing environment. The 40th International Conference on Parallel Processing Workshops (ICPPW), 345 - 351

71. Jung, J., Bae, C., & Lee, J. (2011). Instant virtual desktop system with dynamic I/O client device. International Conference on Engineering and Industries (ICEI), , 1-3

72. Karunamurthy, R., Khendek, F., & Glitho, R. H. (2012). A novel architecture for Web service composition. Journal of Network and Computer Applications, 35(2), 787-802.

73. Kim, & Mook, J. (2013). A secure smart-work service model based OpenStack for Cloud computing. Cluster computing, 17(3), 691-702.

74. Kim, H., Bae, C., & Park, D. (2006). Reliability-redundancy optimization using simulated annealing algorithms. Journal of Quality in Maintenance Engineering, 12(4), 354-363.

75. Kim, H., Wong, W. E., Debroy, V., & Bae, D. (2010). Bridging the gap between fault trees and UML state machine diagrams for safety analysis. 2010 Asia Pacific Software Engineering Conference, 196-205.

76. Kosar, T., Akturk, I., Balman, M., & Wang, X. (2011). PetaShare: A reliable, efficient and transparent distributed storage management system. Scientific Programming, 19(1), 27-43.

77. Kostantos, K., Kapsalis, A., Kyriazis, D., Themistocleous, M., & da Cunha, P. R. (2013). Open-source IaaS fit for purpose: a comparison between opennebula and OpenStack. International Journal of Electronic Business Management, 11(3), 191-201.

78. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., & Wong, E. (2009). Zyzzyva: speculative byzantine fault tolerance. ACM Transactions on Computer Systems (TOCS), 27(4), 1-39.

79. Kulturel-Konak, S., Smith, A. E., & Coit, D. W. (2003). Efficiently solving the redundancy allocation problem using Tabu search. IIE Transactions, 35(6), 515-526.

80. Kumar, P., Chaturvedi, D. K., & Pahuja, G. L. (2009). Heuristic algorithm for constrained redundancy reliability optimization and performance evaluation. Proceedings of the Institution of Mechanical Engineers, 223(O4), 381-386.

81. Kumar, R., Izui, K., Masataka, Y., & Nishiwaki, S. (2008). Multilevel redundancy allocation optimization using hierarchical genetic algorithm. IEEE Transactions on Reliability, 57(4), 650-661.

82. Kumar, R., Izui, K., Yoshimura, M., & Nishiwaki, S. (2009). Optimal multilevel redundancy allocation in series and series–parallel systems. Computers & Industrial Engineering, 57(1), 169-180.

83. Kumar, R., Khatter, K., & Kalia, A. (2011). Measuring software reliability: a fuzzy model. ACM SIGSOFT Software Engineering Notes, 36(6), 1-6.

84. Kumar, V., & Vidhyalakshmi, P. (2012). Cloud computing for business sustainability. Asia Pacific Journal of Management Research and Innovation, 8(4), 461-474.

85. Kuo, W., & Prasad, V. R. (2000). An annotated overview of system-reliability optimization. IEEE Transactions on Reliability, 49(2), 176-187.

86. Kureshi, I., Pulley, C., Brennan, J., Holmes, V., Bonner, S., & James, Y. (2013). Advancing research infrastructure using OpenStack. International Journal of Advanced Computer Science and Applications, 3(4), 64-70.

87. Laprie, J. C., Arlat, J., Beounes, C., & Kanoun, K. (1990). Definition and analysis of hardware- and software-fault-tolerant architectures. Computer, 23(7), 39-51.

88. Lee, C., & Zheng, Y. (2015). Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases. 2015 International Conference on Consumer Electronics-Taiwai (ICCE-TW), 426-427.

89. Liang, Y., & Smith, A. E. (2004). An ant colony optimization algorithm for the redundancy allocation problem (RAP). IEEE Transactions on Reliability, 53(3), 417-423.

90. Liu, H. (2013). Big data drives cloud adoption in enterprise. IEEE Internet Computing, 17(4), 68-71

91. Liu, H., & He, B. (2015). VMbuddies: coordinating live migration of multi-tier applications in cloud environments. IEEE Transactions on Parallel and Distributed Systems, 26(4), 1192-1205.

92. Lo, W., Yin, J., Deng, S., Li, Y., & Wu, Z. (2012). Collaborative Web service QoS prediction with location-based regularization. 2013 IEEE 20th International Conference on Web Services, 464-471.

93. Lombardi, F., & Di Pietro, R. (2011). Secure virtualization for cloud computing. Journal of Network and Computer Applications, 34(4), 1113-1122.

94. Luo, W., Liu, B., & Watfa, A. K. (2014). An open schema for XML data in Hive. 2014 IEEE International Conference on Big Data, 25-31.

95. Lyu, M. R. (2007). Software reliability engineering: A roadmap. Future of Software Engineering, 153-170.

96. Maciel, L. A. H. d. S., & Hirata, C. M. (2013). Fault-tolerant timestamp-based two-phase commit protocol for RESTful services. Software: Practice and Experience, 43(12), 1459-1488.

97. Magott, J., & Skrobanek, P. (2011). Timing analysis of safety properties using fault trees with time dependencies and timed state-charts. Reliability Engineering & System Safety, 97(1), 14-26.

98. Manen, S., Brandt, E., Ekris, J., & Geurts, W. (2015). TOPAAS: an alternative approach to software reliability quantification. Quality and Reliability Engineering International, 31(2), 183-191.

99. Manno, G., & Chiaccho, F. (2011). MatCarloRe: an integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree. Expert Systems with Applications, 39(12), 10334.

100. Meiappane, A., Murugan, S. S., Murugan, S. S., Arun, A., & Ramachandran, A. (2010). Latency of Web service in health care system using GSM networks. 2010 Second International Conference on Machine Learning and Computing, 22-26.

101. Merideth, M. G., Iyengar, A., Mikalsen, T., Tai, S., Rouvellou, I., & Narasimhan, P. (2005). Thema: Byzantine fault tolerant middleware for Web-service applications. Proceedings of the 2005 24th IEEE Symposium on Reliable Distributed Systems, 131-140.

102. Mirandola, R., Potena, P., Riccobene, E., & Scandurra, P. (2014). A reliability model for service component architectures. Journal of Systems and Software, 89, 109-127.

103. Mohanty, R., Ravi, V., & Patra, M. R. (2013). Hybrid intelligent systems for predicting software reliability. Applied Soft Computing, 13(1), 189-200.

104. Nahas, N., Nourelfath, M., & Ait-Kadi, D. (2007). Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series–parallel systems. Reliability Engineering and System Safety, 92(2), 211-222.

105. Ng, K. Y. K., & Sancho, N. G. F. (2001). A hybrid dynamic programming/depth-first search algorithm, with an application to redundancy allocation. IIE Transactions, 33(12), 1047-1058.

106. Okutan, A., & Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. Empirical Software Engineering, 19(1), 154-181.

107. Onishi, J., Kimura, S., James, R. J. W., & Nakagawa, Y. (2007). Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method. IEEE Transactions on Reliability, 56(1), 94-101.

108. Pedersen, J. M., Tahir Riaz, M., Dubalski, B., Ledzinski, D., Júnior, J. C., & Patel, A. (2013). Using latency as a QoS indicator for global cloud computing services. Concurrency and Computation: Practice and Experience, 25(18), 2488-2500.

109. Phyu, M. P., & Thein, N. L. (2011). Efficient storage management for distributed storage system. Fourth International Conference on Machine Vision, 8350, 1-12.

110. Portinale, L., & Codetta Raiteri, D. (2011). Using dynamic decision networks and extended fault trees for autonomous FDIR. 2011 23rd IEEE International Conference on Tools with Artificial Intelligence, 480-484.

111. Qian, M., Hardjawana, W., Shi, J., & Vucetic, B. (2015). Baseband processing units virtualization for cloud radio access networks. IEEE Wireless Communications Letters, 4(2), 189-192.

112. Qiu, W., Zheng, Z., Wang, X., Yang, X., & Lyu, M. R. (2014). Reliability based design optimization for cloud migration. IEEE Transactions on Services Computing, 7(2), 223-236.

113. Rahmani, M., Azadmanesh, A., & Siy, H. (2014). Architectural reliability analysis of framework-intensive applications: A web service case study. Journal of Systems and Software, 94, 186-201.

114. Rajaraman, V. (2014). Cloud computing. Resonance, 19(3), 242-258.

115. Ramirez Marquez, J. E., & Coit, D. W. (2004). A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems. Reliability Engineering and System Safety, 83(3), 341-349.

116. Ramirez Marquez, J. E., Coit, D. W., & Konak, A. (2004). Redundancy allocation for series-parallel systems using a max-min approach. IIE Transactions, 36(9), 891-898.

117. Randell, B., & Xu, J. (1995). The evolution of the recovery block concept. Fault Tolerance, 1-21.

118. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens: an open architecture for collaborative filtering of netnews. Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, 175-186.

119. Ro, C. (2015). Modeling and analysis of memory virtualization in cloud computing. Cluster computing, 18(1), 177-185.

120. Salas, J., Sorrosal, F. P., Martínez, M. P., & Peris, R. J. (2006). WS-replication: a framework for highly available web services. Proceedings of the 15th International Conference on World Wide Web 357-366

121. Santos, G. T., Lung, L. C., & Montez, C. (2005). FTWeb: a fault tolerant infrastructure for web services. Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference, 95-105.

122. Saraladevi, B., Pazhaniraja, N., Paul, P. V., Basha, M. S. S., & Dhavachelvan, P. (2015). Big Data and Hadoop: a study in security perspective. Procedia Computer Science, 50, 596-601.

123. Schlosser, D., Staehle, B., Binzenhöfer, A., & Boder, B. (2010). Improving the QoE of Citrix thin client users. 2010 IEEE International Conference on Communications, 1-6.

124. Serrano, D., Han, D., & Stroulia, E. (2015). From relations to multi-dimensional maps: towards an SQL-to-HBase transformation methodology. IEEE 8th International Conference on Cloud Computing, 81-89.

125. Seung, H. S., & Lee, D. D. (1999). Learning the parts of objects by non-negative matrix factorization. Nature, 401(6755), 788-791.

126. Sharma, L. K., Saket, R. K., & Sagar, B. B. (2015). Software reliability growth models and tools - a review. 2015 2nd International Conference on Computing for Sustainable Global Development, 2057-2061.

127. Shen, Y., Chen, H., Shen, L., Mei, C., & Pu, X. (2014). Cost optimized resource provision for cloud applications. 2014 IEEE International Conference on High Performance Computing and Communications, 1060-1067.

128. Shirey, J., Charng, A., & Nguyen, Q. (2013). Researching and communicating the complexity of IT image management. Newsletter Communication Design Quarterly Review, 1(3), 28-33

129. Singh, L. K., Tripathi, A. K., & Vinod, G. (2015). Approach for parameter estimation in Markov model of software reliability for early prediction: a case study. IET Software, 9(3), 65-75.

130. Song, J., Guo, C., Wang, Z., Zhang, Y., Yu, G., & Pierson, J.-M. (2015). HaoLap: A Hadoop based OLAP system for big data. Journal of Systems and Software, 102, 167-181.

131. Staalinprasannah, N., & Suriya, S. (2013). Implementation of Xenserver to ensuring business continuity through power of virtualization for cloud computing. 2013 Fourth

International Conference on Computing, Communications and Networking Technologies, 1-6.

132. Thanakornworakij, T., Nassar, R., Leangsuksun, C. B., & Paun, M. (2013). Reliability model of a system of k nodes with simultaneous failures for high-performance computing applications. International Journal of High Performance Computing Applications, 27(4), 474-482.

133. Tian, G., & Meng, D. (2010). Failure rules based node resource provision policy for cloud computing. 2010 International Symposium on Parallel and Distributed Processing with Applications, 397-404.

134. Tian, Z., Zuo, M. J., & Huang, H. (2008). Reliability redundancy allocation for multi-State series-parallel systems. IEEE Transactions on Reliability, 57(2), 303-310.

135. Tziritas, N., Khan, S. U., Xu, C. Z., Loukopoulos, T., & Lalis, S. (2013). On minimizing the resource consumption of cloud applications using process migrations. Journal of Parallel and Distributed Computing, 73(12), 1690-1704.

136. Wang, K., Bian, Z., Chen, Q., Wang, R., & Xu, G. (2014). Simulating Hive cluster for deployment planning, evaluation and optimization. 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, 475-482.

137. Wang, L., Cheng, C., Wu, S., Wu, F., & Teng, W. (2015). Massive remote sensing image data management based on HBase and GeoSOT. 2015 IEEE International Geoscience and Remote Sensing Symposium, 4558-4561.

138. Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Kawaba, M., & Pu, C. (2012). Response time reliability in cloud environments: an empirical study of n-tier applications at high resource utilization. 2012 IEEE 31st Symposium on Reliable Distributed Systems, 378-383.

139. Wang, S., & Watada, J. (2009). Modelling redundancy allocation for a fuzzy random parallel–series system. Journal of Computational and Applied Mathematics, 232(2), 539-557.

140. Wang, W., Loman, J., & Vassiliou, P. (2004). Reliability importance of components in a complex system. Proceedings of the Annual Reliability and Maintainability Symposium, 6-11.

141. Wang, W., Wang, H., Yang, B., Liu, L., Liu, P., & Zeng, G. (2013). A Bayesian network based knowledge engineering framework for IT service management. IEEE Transactions on Services Computing, 6(1), 76-88.

142. Wang, Y., & Li, L. (2012). Heterogeneous redundancy allocation for series-parallel multi-state systems using hybrid particle swarm optimization and local search. IEEE Transactions on Systems, Man, and Cybernetics, 42(2), 464-474.

143. Wang, Z., Tang, K., & Yao, X. (2010). A memetic algorithm for multi-level redundancy allocation. IEEE Transactions on Reliability, 59(4), 754-765.

144. Wei, D. S. L., Pearson, S., Matsuura, K., Lee, P. P. C., & Naik, K. (2014). Guest editorial: cloud security. IEEE Transactions on Cloud Computing, 2(4), 377-379.

145. Wu, J., Chen, L., Zheng, Z., Lyu, M. R., & Wu, Z. (2014). Clustering Web services to facilitate service discovery. Knowledge and Information Systems, 38(1), 207-229.

146. Wu, Z., Xiong, N., Han, W., Huang, Y. N., Hu, C. Y., Gu, Q., & Hang, B. (2013). A fault tolerant method for enhancing reliability of services composition application in WSNs based on BPEL. International Journal of Distributed Sensor Networks, 2013, 1-11.

147. Xiang, J., Yanoo, K., Maeno, Y., Tadano, K., Machida, F., Kobayashi, A., & Osaki, T. (2011). Efficient analysis of fault trees with voting gates. IEEE International Symposium on Software Reliability Engineering, 230-239.

148. Xu, G., Lin, S., Wang, G., Liu, X., Shi, K., & Zhang, H. (2012). HERO: heterogeneity aware erasure coded redundancy optimal allocation for reliable storage in distributed networks. IEEE 31st International Performance Computing and Communications Conference, 246-255.

149. Yalaoui, A., Chatelet, E., & Chu, C. (2005). A new dynamic programming method for reliability & redundancy allocation in a parallel-series system. IEEE Transactions on Reliability, 54(2), 254-261.

150. Yalaoui, A., Chu, C., & Châtelet, E. (2005). Reliability allocation problem in a series–parallel system. Reliability Engineering and System Safety, 90(1), 55-61.

151. Yangui, S., Ben Nasrallah, M., & Tata, S. (2013). PaaS-independent approach to provision appropriate cloud resources for SCA based applications deployment. 2013 Ninth International Conference on Semantics, Knowledge and Grids, 14-21.

152. Yao, Q., Tian, Y., Li, P. F., Tian, L. L., Qian, Y. M., & Li, J. S. (2015). Design and development of a medical big data processing system based on Hadoop. Journal of Medical Systems, 39(3), 1-11.

153. Yeh, W. C. (2009). A two stage discrete particle swarm optimization for the problem of multiple multi-level redundancy allocation in series systems. Expert Systems with Applications, 36(5), 9192-9200.

154. Yeh, W. C., & Hsieh, T. J. (2011). Solving reliability redundancy allocation problems using an artificial bee colony algorithm. Computers and Operations Research, 38(11), 1465-1473.

155. Yeo, H., & Crawford, C. H. (2015). Big data: cloud computing in genomics applications. IEEE International Conference on Big Data, 2904-2906.

156. Yigitbasi, N., Gallet, M., Kondo, D., Iosup, A., & Epema, D. (2010). Analysis and modeling of time correlated failures in large-scale distributed systems. 2010 11th IEEE/ACM International Conference on Grid Computing, 65-72.

157. Yu, C., & Huang, L. (2014). A Web service QoS prediction approach based on time and location aware collaborative filtering. Service Oriented Computing and Applications, 10(2), 135-149.

158. Yu, T., Qiu, J., Reinwald, B., Zhi, L., Wang, Q., & Wang, N. (2012). Intelligent database placement in cloud environment. 2012 IEEE 19th International Conference on Web Services, 544-551.

159. Yu, Y., Zhao, J., Wang, X., Wang, Q., & Zhang, Y. (2015). Cludoop: an efficient distributed density based clustering for big data using Hadoop. International Journal of Distributed Sensor Networks, 2015.

160. Yun, W. Y., & Kim, J. W. (2004). Multi-level redundancy optimization in series systems. Computers & Industrial Engineering, 46(2), 337-346.

161. Yun, W. Y., Song, Y. M., & Kim, H.-G. (2007). Multiple multi-level redundancy allocation in series systems. Reliability Engineering and System Safety, 92(3), 308-313.

162. Zhang, Y., Zheng, Z., & Lyu, M. R. (2011). BFTCloud: a byzantine fault tolerance framework for voluntary resource cloud computing. 2011 IEEE International Conference on Cloud Computing (CLOUD), 444-451

163. Zhang, Y., Zheng, Z., & Lyu, M. R. (2014). An online performance prediction framework for service oriented systems. IEEE Transactions on Systems, Man, and Cybernetics, 44(9), 1169-1181.

164. Zhao, L., Ren, Y., Li, M., & Sakurai, K. (2012). Flexible service selection with user-specific QoS support in service-oriented architecture. Journal of Network and Computer Applications, 35(3), 962–973.

165. Zhao, W. (2007). Byzantine fault tolerance for nondeterministic applications. Third IEEE International Symposium on Dependable, Autonomic and Secure Computing, 108-115.

166. Zhao, W. (2009). Design and implementation of a Byzantine fault tolerance framework for Web services Journal of Systems and Software, 82(6), 1004-1015.

167. Zheng, Z., & Lyu, M. R. (2008). A QoS-aware fault tolerant middleware for dependable service composition. IEEE 19th International Symposium on Software Reliability Engineering, 239-248.

168. Zheng, Z., & Lyu, M. R. (2010). An adaptive QoS-aware fault tolerance strategy for web services. Empirical Software Engineering 15(4), 323-345

169. Zheng, Z., & Lyu, M. R. (2013). Personalized reliability prediction of Web services. ACM Transactions on Software Engineering and Methodology (TOSEM), 22(2), 1-25.

170. Zheng, Z., & Lyu, M. R. (2015). Selecting an optimal fault tolerance strategy for reliable service oriented systems with local and global constraints. IEEE Transactions on Computers, 64(1), 219-232.

171. Zheng, Z., Ma, H., Lyu, M. R., & King, I. (2011). QoS aware Web service recommendation by collaborative filtering. IEEE Transactions on Services Computing, 4(2), 140-152.

172. Zheng, Z., Ma, H., Lyu, M. R., & King, I. (2013). Collaborative Web service QoS prediction via neighborhood integrated matrix factorization. IEEE Transactions on Services Computing, 6(3), 289-299.

173. Zheng, Z., Zhang, Y., & Lyu, M. R. (2010). CloudRank: a QoS-driven component ranking framework for cloud computing. 2010 29th IEEE International Symposium on Reliable Distributed Systems, 184-193.

174. Zheng, Z., Zhang, Y., & Lyu, M. R. (2014). Investigating QoS of real world Web services. IEEE Transactions on Services Computing, 7(1), 32-39.

175. Zheng, Z., Zhou, T. C., Lyu, M. R., & King, I. (2012). Component ranking for fault tolerant cloud applications. IEEE Transactions on Services Computing, 5(4), 540-550.

176. Zheng, Z., Zhu, J., & Lyu, M. R. (2013). Service generated big data and big data-as-a-service: an overview. 2013 IEEE International Congress on Big Data, 403-410.

177. Zheng, Z. B., Zhou, T. C., Lyu, M. R., & King, I. (2010). FTCloud: a component ranking framework for fault tolerant cloud applications. 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE), 398-407

178. Zhu, J., Kang, Y., Zheng, Z., & Lyu, M. R. (2012). WSP: a network coordinate based Web service positioning framework for response time prediction. 2012 IEEE 19th International Conference on Web services, 90-97.

179. Zhu, J., Zheng, Z., & Lyu, M. R. (2013). DR2: dynamic request routing for tolerating latency variability in online cloud applications. 2013 IEEE Sixth International Conference on Cloud Computing, 589-596.

180. Zhu, M., Zhang, X., & Pham, H. (2015). A comparison analysis of environmental factors affecting software reliability. Journal of Systems and Software, 109, 150-160.

181. Zhu, X., Qin, X., & Qiu, M. (2011). QoS aware fault tolerant scheduling for real-time tasks on heterogeneous clusters. IEEE Transactions on Computers, 60(6), 800-812.