國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

分散式異質性伺服器之多重資源動態負載平衡

Dynamic Load Balancing in Distributed Heterogeneous Multi-Resource Servers

研　究　生：楊智強

指導教授：陳　健 教授

中 華 民 國 九 十 七 年 七 月

分散式異質性伺服器之多重資源動態負載平衡

Dynamic Load Balancing in Distributed Heterogeneous Multi-Resource Servers

研 究 生：楊智強　　Student：Chih-Chiang Yang

指導教授：陳　健　　Advisor：Chien Chen

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

In partial Fulfillment of the Requirements

For the Degree of

Master

In

Computer Science

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

分散式異質性伺服器之多重資源動態負載平衡

研究生 ： 楊智強　　　　　　　　　　指導教授: 陳　健

國 立 交 通 大 學 資 訊 科 學 與 工 程 研 究 所

中文摘要

由於網際網路的蓬勃發展，使用網際網路服務的人口迅速地增加，因此必須同時使用更多的服務伺服器同時提供服務，來滿足大量的服務需求。同時使用多台伺服器若不能有效地分配負載，不僅造成系統利用度的低落，還會導致服務品質的不佳。為此已有很多研究被提出來解決這方面的問題。依據不同的需求，負載平衡的架構可大致被分為調度器為主、DNS 為主、使用者為主，以及伺服器為主的四大架構，且皆具有各自的優缺點。過去在地理分散的服務伺服器負載平衡方法上，服務伺服器為了降低狀態變化的頻率，通常會設定一負載緩衝區間，來降低因服務伺服器狀態變化而產生的網路封包，然而卻會造成各服務伺服器負載震盪的現象。此外傳統方法通常假設各伺服器皆具有相同的負載能力，且通常只考慮服務請求的一種資源消耗，然而事實上，各伺服器間往往不見得都具有相同的負載能力，而各服務請求也不只需要消耗一種伺服器的資源，因此只考慮一種資源消耗，將造成系統瓶頸發生在少數的資源上，導致整體的系統使用率不佳。為此，在地理分散的負載平衡架構上，我們採用 Random Early Detection (RED) 的機制，來更有效地決定伺服器的狀態，降低服務伺服器負載震盪的幅度。此外在地理集中的負載平衡架構上，我們提出一分散式的市場機制負載平衡方法，不僅考量各伺服器間的異質性，也考量各工作的不同資源需求。透過模擬，我們證明透過 RED 的概念，對於地理分散的伺服器而言，我們可以有效地降低各伺服器

負載震盪的幅度，而對於叢集內的伺服器而言，分散式的市場機制負載平衡方法，不僅可以有效地提高服務伺服器間，以及服務伺服器內部資源的負載平衡性，且即使伺服器的異質性提高，以我們的分散式市場機制負載平衡方法仍可以維持良好的負載平衡性，達到高使用率以及低回應時間等優點。

Dynamic Load Balancing in Distributed Heterogeneous Multi-Resource Servers

Student: Chih-Chiang Yang **Advisor: Dr. Chien Chen**

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

Due to the progress of the Internet, there are more and more people using Internet services nowadays. In order to satisfy the huge service requirements, using multiple servers to provide different services at the same time is necessary. However, if we can't effectively divide loads among servers, server utilization could decline and service quality could become uneven. Because of this reason, there are many researches have be presented to solve this problem. The load balancing methods can be roughly classified to four architectures: dispatcher-based, DNS-based, client-based and server-based, and different architectures have their own advantages and disadvantages. The conventional methods of load balancing always set a load buffer range to decrease the state change frequency of a service server in the geographic distributed load balancing architecture, and mostly assume that servers are homogeneous and just consider single resource consumption, such as CPU load. However, the load buffer range would result in load oscillation among servers. On the other hand, servers may not always have the same capacity, and jobs need many kinds of resource requirements. Only considering single resource consumption would cause the system bottleneck to derive from the lack of a small number of resources, and lead to low system utilization. For this reason, in the geographic distributed load balancing architecture, we use the concept of Random Early Detection (RED) to determine the server state probabilistically, and in the cluster load balancing architecture, we present a distributed market mechanism (MM) load balancing method which would consider the server heterogeneity and multiple-resource consumption simultaneously. In our simulation, we show that the oscillation of service server load can be reduced by using the concept of RED in the geographic distributed load balancing architecture. And distributed market mechanism load balancing method can improve the inter-server and intra-server load balancing at the same time, and keep the performance even if the server heterogeneity increasing, achieve high system utilization and low request response time.

Keywords: dispatch-based, DNS-based, client-based, server-based, load balancing;

誌謝

本篇論文的完成，我要感謝這兩年來給予我協助與勉勵的人。首先要感謝我的指導教授 陳健博士，即使這段期間遭遇了不少的挫折，陳老師對我的指導與教誨使我都能在遇到困難時另尋突破，在研究上處處碰壁時指引明路讓我得以順利完成本篇論文，在此表達最誠摯的感謝。同時也感謝我的論文口試委員，交大的曾煜棋教授、簡榮宏教授，及工業技術研究院的蔣村杰副經理，他們提出了許多的寶貴意見，讓我受益良多。

感謝與我一同努力的學長陳盈羽、張哲維，由於我們不斷互相討論及在論文上的協助，使我能突破瓶頸，研究也更為完善。另外我也要感謝實驗室的同學、學弟們，林政一、陳信帆、王振仰、張文馨、蔡赫維等人，感謝他們陪我度過這兩年辛苦的研究生活，在我需要協助時總是不吝伸出援手，陪我度過最煩躁與不順遂的日子。

特別感謝我的朋友，及許多大學時代的好友們，他們在精神上給我莫大的鼓勵，傾聽我內心 的聲音並帶給我無比的溫暖，指導我做出了許多正確的抉擇，而不致迷失了自我。這些好朋友們就像是明燈般照亮了昏暗的旅程，並陪伴我度過枯燥的研究生涯。

最後，我要感謝家人對我的關懷及支持，他們含辛茹苦的栽培，使我得以無後顧之憂的專心於研究所課業與研究，我要向他們致上最高的感謝。

Table of Content

List of Figure

List of Equation

Chapter 1: Introduction

In recent years, the number of people using Internet services is increasing due to the rapid development of Internet. Single service server that provides Internet service has become unable to cope with the growing Internet service demands. Constantly improving the performance of a single server not only increases the cost, it cannot really solve the problem of rapidly growing Internet service demands. Instead, it is necessary to use more service servers to provide services simultaneously.

However, even if there are multiple servers providing services at the same time, if we can not effectively assign users among servers, this would cause some service servers' load to be too high, while others' would be relatively low, resulting in bad and unstable quality of service. There are many researches that have been presented to solve this server uneven load problem. In [1], it classifies load balancing architecture into four classes: client-based, dispatcher-based, DNS-based, and server-based load balancing architecture, summarized as follows:

Firstly, in client-based load balancing architecture, client hosts need to modify their software or hardware or through the user manually depending on the service quality of service server to choose a better service server. Its shortcoming is not convenient to users.

Secondly, in dispatcher-based load balancing architecture, all service servers are usually placed in a geographically centralized area, and through a dispatcher to receive all user requests, and then depending on the states of service servers at each time, dispatcher can determine which service server is the best to provide service. Because of the service servers are placed in a geographical central area, the state information of servers can be more immediately get to help dispatcher to do load balancing. However, its first disadvantage is that services would be suspended due to the single dispatcher fault resulting in poor reliability. The second disadvantage is that only part of users who are closer to that system can get low propagation delay.

Thirdly, in DNS-based load balancing architecture, servers can be placed in geographical distributed areas. The users first send a domain name resolution request to the DNS server to obtain an IP address of a service server, and then send the service request to the service server with that IP address to get services. DSN servers usually do load balancing through assign users to different servers by random or round robin manners or according to the server states, such as load condition, network situation, which are periodically receiving from servers. As the master/slave DSN architecture has been widely used, that is, once the master DNS server failed, there is a slave DNS server can take over its works resulting in high reliability. But the difficulty is that because of servers are placed in geographical decentralized area. Hence the service server's states are not allowed to be obtained immediately to avoid congesting or wasting network bandwidth.

Finally, in server-based load balancing architecture, it first needs the assistance of a simple dispatcher or DNS server to simply distribute users among servers by random or round robin manner. At time goes by, the problem of load uneven between servers could begin to appear because of different execution time and resource requirements among jobs. At this time, servers must to determine if they need to exchange jobs or not by exchanging the states to each other to raising the load balancing degree. The advantage of this load balancing architecture is that the states of servers can be exchanged more immediately because of servers are placed in a geographic central area to achieve more good load balancing degree. On the other hand, load balancing is done by the coordination of all service servers in this architecture, small part of service servers failed would not dramatically affect the quality of services, so this architecture has high reliability.

In the load balancing policy, the conventional methods of load balancing always set a load buffer range to decrease the state change frequency of a service server in the geographic distributed load balancing architecture, and mostly assume that servers are homogeneous and just consider single resource consumption, such as CPU load; however, the load buffer range would result in load oscillation among servers. On the other hand, servers may not always have the same capacity and jobs almost needs many kinds of resource requirements, such as memory space, network bandwidth, etc. Only consider single resource consumption would cause the system bottleneck to derive from the lack of a small number of resources, and lead to low system utilization.

In this thesis, we would integrate the DNS-based and server-based load balancing architecture to implement our load balancing method. As shown in Fig. 1, all service servers would be first partitioned to multiple server clusters and placed in geographical distributed areas. The servers with the minimum ID in a server cluster should determine the states of its server cluster by Random Early Detection (RED) method. The idea of RED method is that the probability of the state of a server cluster becoming overloaded is directly proportional to the load of the server cluster at that time. After determine the state of the server cluster, the server with the minimum ID in that server cluster would periodically send that state to DNS server, then DNS server can assign client requests among server clusters according to the state of each server cluster.

Once the client requests arrived in a server cluster, our distributed market mechanism load balancing method would do the second phase load balancing inside the server cluster. The concept of our market mechanism load balancing method is that the cost of one job executed on a service server is related to the proportion of its different resource requirements, and the cost of each resource requirement is directly proportional to the load of that resource of the server. Hence, we would consider the different cost of a job executed on each server to determine the server with the best fit. On the other hand, each server must consider its multiple heterogeneous resource capacities and the multiple heterogeneous resource requirements of jobs, and through exchanging state information of each other to determine if they need to exchange jobs or not in order to raise the overall system load balancing degree and provide stable, reliable, and scalable high quality services further. In our simulation, we use four metrics: average standard deviation of service server loads, average standard deviation of resource loads, average server utilization, and average turn around time to analysis and compare our method with other conventional methods.

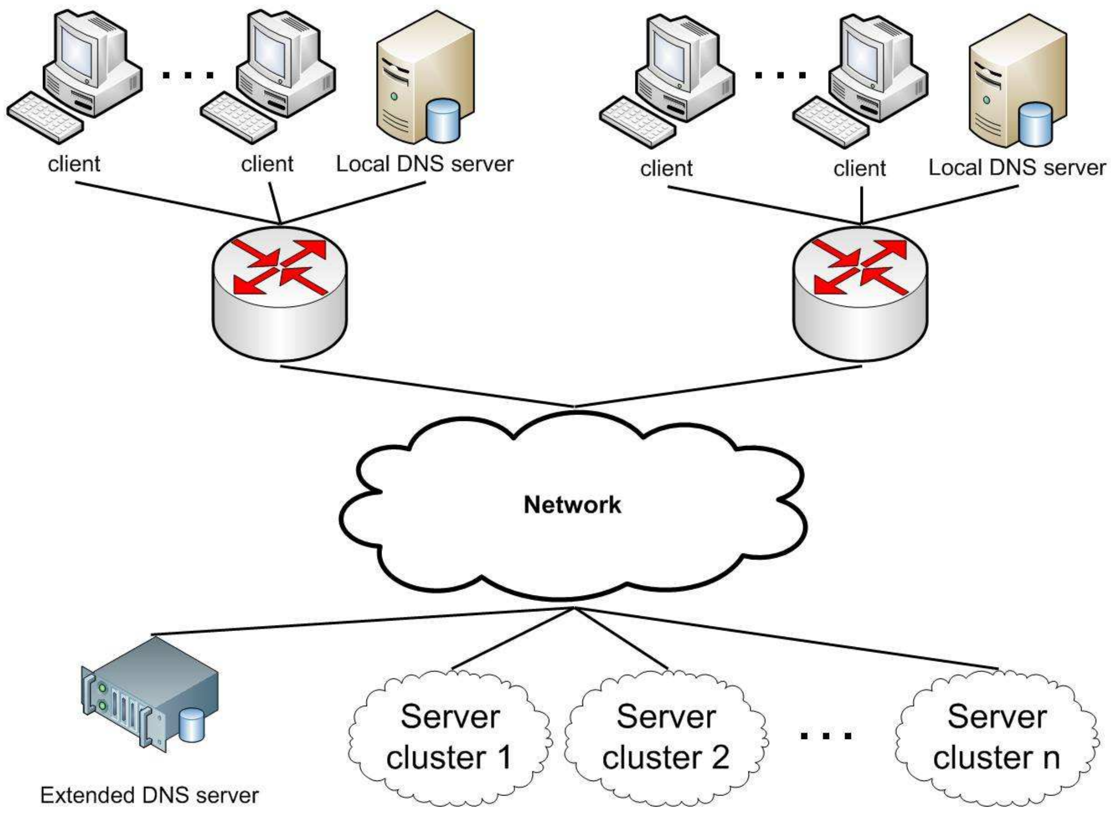


Fig. 1、DNS-based load balancing architecture

Chapter 2: Related work

[1][3] introduce four load balancing architectures: client-based, dispatcher-based[17], DNS-based, and server-based load balancing architecture, and some comparison of load balancing policies. [11] presents an adaptive TTL policy to do load balancing in DNS-based load balancing architecture, and [18] proposes that the TTL value must be carefully used. Also, in geographically distributed network architecture, [10] considers the load of servers in each zone and the cost of transferring a job across zones to determine if there is benefit of executing a job across zones or not. Similarly, [5] considers the cost of transferring a job from one server to another to determine whether to execute a job in a local or remote server.

[8][7] simply introduce four policies of distributed load balancing: information policy, transfer policy, location policy, and selection policy. [16] presents the most popular threshold-based transfer policy.

Recently, considering multiple-resource in load balancing has become more and more important. [9] presents a multiple-resource load balancing method to a single server, which defines a load balancing measure: (maximum load / average load), to determine executing which job in the server can obtain the highest load balancing degree, and then extend it to multiple servers in [6]. [2] also considers multiple resource requirements of a job, and uses backfill lowest approach to do job scheduling, on the other hand, it provides different QoS (Quality of Service) to different user classes.

[13] presents a LDMA method, which, adds a mobile agent into each server to help do a reliable and scalable load balancing. [12][14] introduce the characteristic of heavy-tailed workload in web service, and claim that different-sized jobs should be executed in different servers. [19] adds an in or out tag into a packet according to its

priority, and opportunely drops some packets with lower priority to achieve load balancing between clients.

[22] presents Random Early Detection (RED) gateways for congestion avoidance in packet-switched networks, avoiding the global synchronization that results from many connections decreasing their windows at the same time.

Chapter 3: DNS-based Load Balancing

In today’s Internet, each host should have its unique IP address in the network. Before sending a service request to a server, client must know the IP address of that server. According to the definition of IPv4 protocol, an IP address is composed of 32-bit digital number. However, such a 32-bit address is difficult to memorize for users. For this reason, domain name system has been invented to replace the 32-bit digital number with a domain name which is composed of a number of English words. Intuitively, a domain name is relatively easy to memorize for users. Following we will simply introduce the operation of DNS, and illustrate how to do load balancing among service servers by using DNS server.

3.1 DNS overview

DNS-based load balancing architecture is shown as Fig. 1. At first, clients would be partitioned into several groups according the local DNS (LDNS) server they used. Once a client wants to obtain a service from a server with a particular domain name, he would first send the domain name resolution query to his LDNS server. After receiving a domain name resolution query, the LDNS would first check to see whether there was a valid and unexpired IP address of that domain name or not. If yes, LDNS would directly send the IP address to the client. Otherwise, the LDNS server would need to ask the root DNS server for the IP address of a DNS server (the Extended DNS server in Fig. 1, also called EDNS server) which is responsible for resolving that domain name, and the LDNS server would then forward the domain name resolution query of the client to the EDNS server to obtain a new IP address of a service server and a TTL time. Finally, LDNS can response the new IP address to the client, and record the IP address for the TTL time. Before the TTL time expires, each domain name resolution query for the same domain name can be directly sent by the LDNS server without asking the EDNS server.

The characteristics of DNS-based load balancing architecture are as follows:

1. All service servers can be placed in geographic distributed area.
2. There is no directly geographic relationship between DNS server and service servers.

In DNS-based load balancing architecture, service servers can be allowed to be placed in geographic distributed areas. If we can consider the geographic relationship between service servers and clients, then we can provide the advantage of low propagation delay to clients. Moreover, because of the mature master/slave architecture of DNS, slave DNS servers may periodically backup the data of the master server, and assist in apportioning the domain name resolution queries of the master DNS server. If the master DNS server failed, one of the slave DNS servers would take over the subsequent work of the master DNS server, so it has high reliability.

However, there is usually no or little information exchange between the DNS server and service servers because of no direct geographic relationship between the DNS server and service servers. For this reason, conventional DNS-based load balancing methods usually use a random or round robin approach to do simple load balancing; however, this kind of method is most likely to cause an unbalanced load among service servers. Hence, what we need to consider is how to use infrequent server state information to achieve a high load balancing degree.

3.2 Conventional DNS-based load balancing method

In conventional DNS-based load balancing method, the DNS server uses a round robin manner to distribute the load among servers as the conventional method, and the service server periodically sends its overload or no overload state to the DNS server. Then, when doing round robin, the DNS server would skip the service servers with an overload state to raise the load balancing degree among servers. As previously mentioned, there is usually no direct geographic relationship between the DNS server and service servers, so the service server is not allowed to immediately send its state information to the DNS server to avoid too many state information packets congesting or wasting network bandwidth. For this reason, conventional method usually defines a load buffer range (LBR) in service servers. The finite state machine of LBR is shown in Fig. 2. Before the load of a service server is greater than 90%, the state of that server is no overload. That is, the DNS server can assign new client group requests to that service server. Once the load of that service server greater than 90%, its state would become overload, and it would keep this overload state until its load is less than 70%. The line chart of the service server state is shown in Fig. 3.

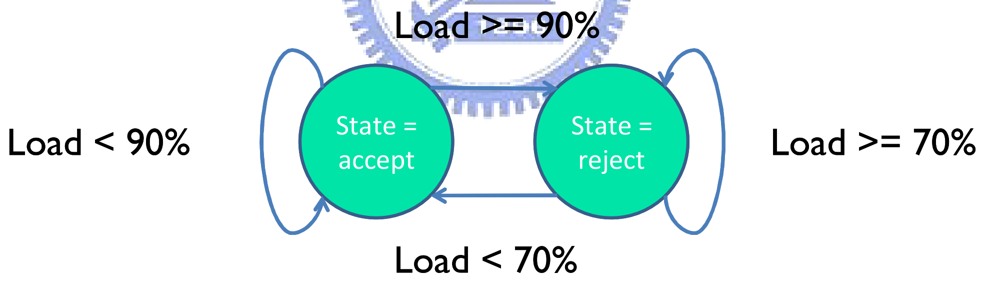


Fig. 2、Finite state machine of conventional load buffer range method

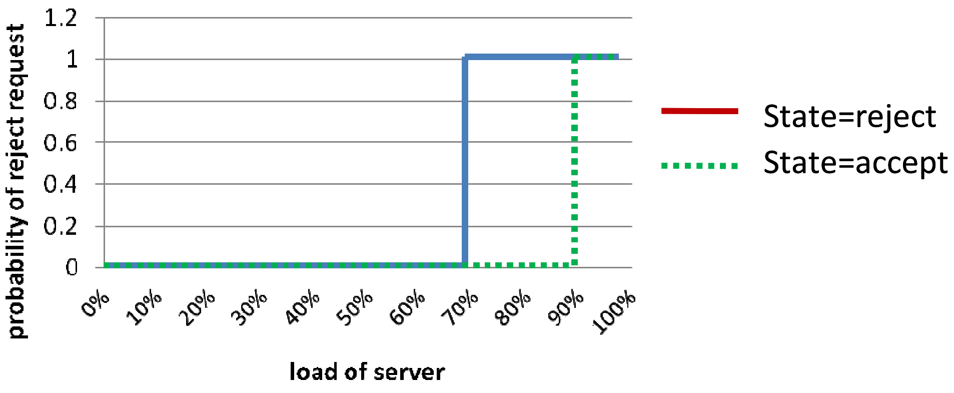


Fig. 3、Line chart of conventional load buffer range method

Noteworthy is that a service server with a state of overload does not mean it will not accept any client request, but instead notify the DNS server not to assign new client group requests to that service server. Before the TTL time expires, the client groups which are previously assigned to that service server could continually send their service requests to that service server, and that service server would also continually provide services to those client groups.

In this conventional method, when there are not many service servers and the amount of requests is high, once one of the service servers is overloaded, it must keep its overload state until its load is under 70% and then notify DNS server to assign new client group requests to that service server. During this period, the other service servers need to assist in sharing the additional 20% (90%-70%) load of that overload service server. This would cause the other service servers to overload and underload by turns, as shown in Fig. 4, resulting in unstable service quality.

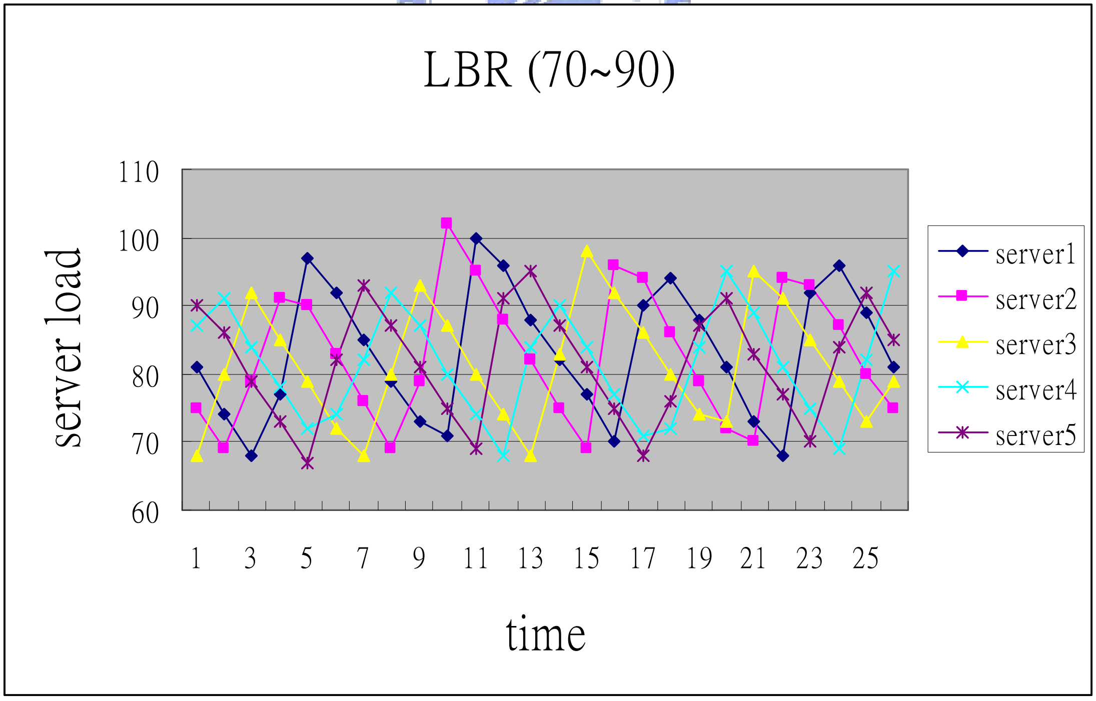


Fig. 4、Server load oscillation phenomenon of conventional method

3.3 Random Early Detection (RED) method

In order to solve the above load oscillation phenomenon of service servers, we think that the state of overload or no overload of a service server in the load buffer range should be a probability, but not absolute, in order to avoid burdening the other service servers with too many loads. Hence, we use the concept of random early detection (RED) method to determine the overload or no overload state of service servers probabilistically. That is, the probability of a service server state becoming overloaded is directly proportional to its load. The line chart of the probability of a service server state becoming overloaded is shown in Fig. 5, as the load of a service server becomes greater than 70%, its state starts to have a chance of becoming overload, as the load of the service server increases, the probability of a service server state becoming overloaded also increases. Once the load of a service server becomes greater than 90%, the probability of its state becoming overloaded should be 1.

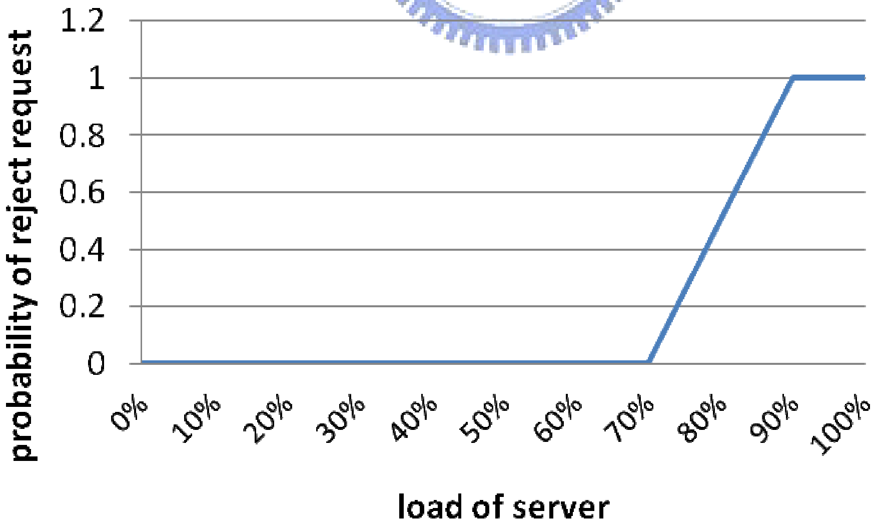


Fig. 5、Line char of state change of RED method

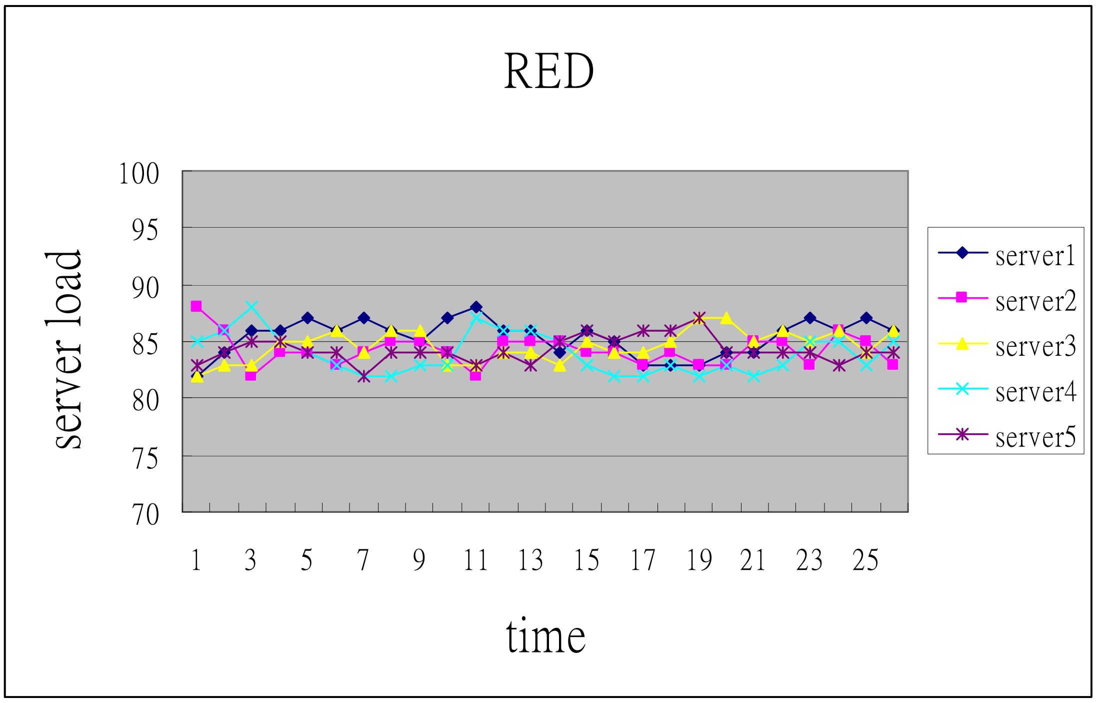


Fig. 6、Server load variation of RED method

As shown in Fig. 6, comparing to conventional method and RED method, using RED method to determine the state of service services can effectively raise the load balancing degree among service servers in the same traffic environment.

Then we try to decrease the load buffer range of conventional methods to observe the relationship between the range of load buffer and the standard deviation of service server load. Fig. 7 and Fig. 8 are load buffer ranges from 80% to 90% and from 84% to 86% respectively. As we can see, reducing the load buffer range of conventional method can diminish the degree of server load oscillation.

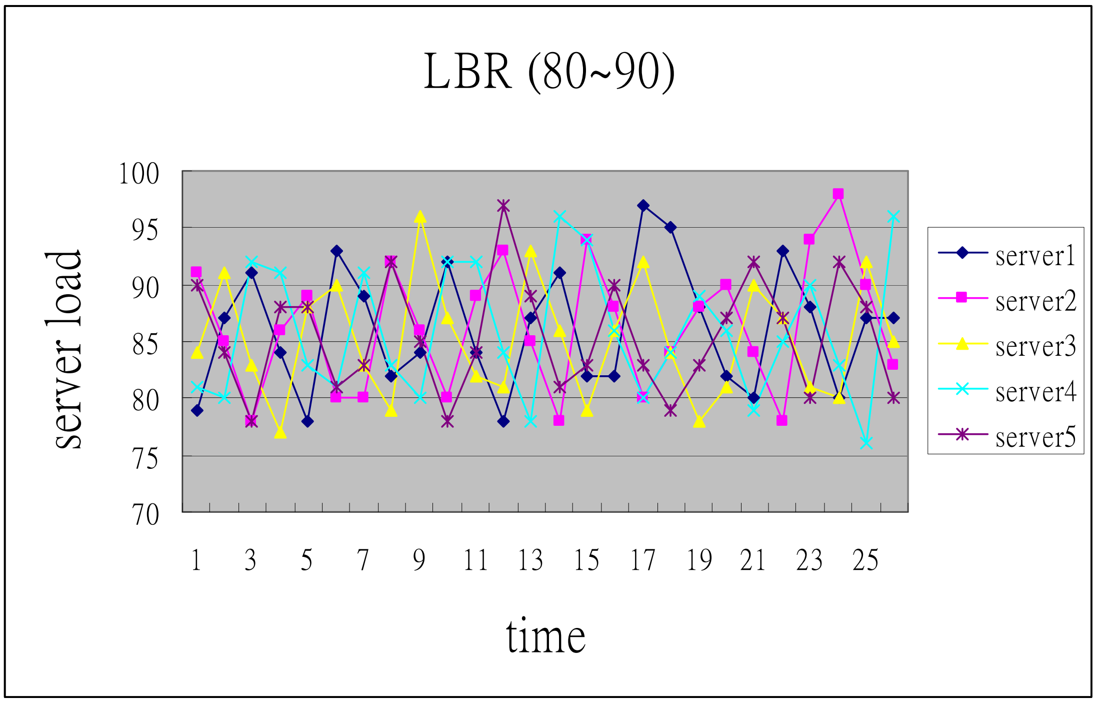


Fig. 7、Server load variation of load buffer range between 80% and 90%

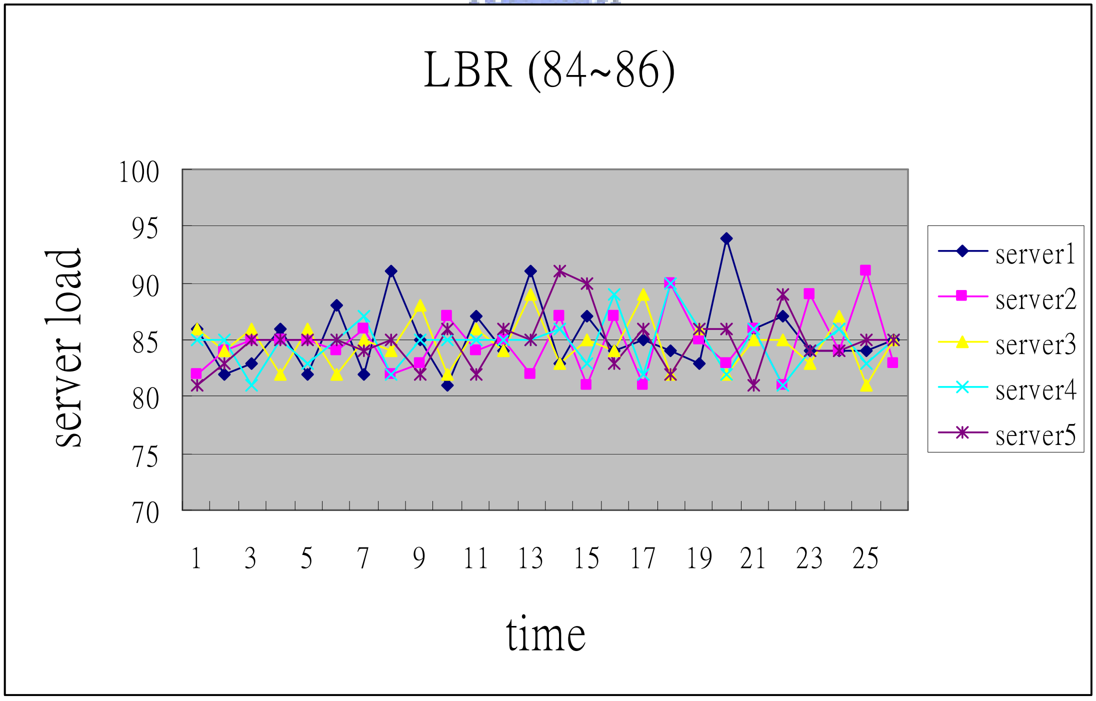


Fig. 8、Server load variation of load buffer range between 84% and 86%

Finally, we should make a summary comparison of server state change frequency and the average standard deviation of service server load for RED method and conventional method with different load buffer range setting. As shown in Fig. 9, even if we constantly reduce the load buffer range of conventional method until it is zero, its load balancing degree will become closer to but still be slightly higher than the RED method's, and its server state change frequency has became 1.5 times of the RED method's at this time. Moreover, if we use the default setting (70%~90%) of the conventional method, although its sever state change frequency is half of the RED method's, but its average standard deviation of server load is greater than five times of the RED method's at this time. Hence, compared with conventional method, RED method can use an acceptable server state change frequency to achieve excellent average standard deviation of service server, and no matter what load buffer range of conventional method is set to, its load balancing degree is still worse than RED method's.

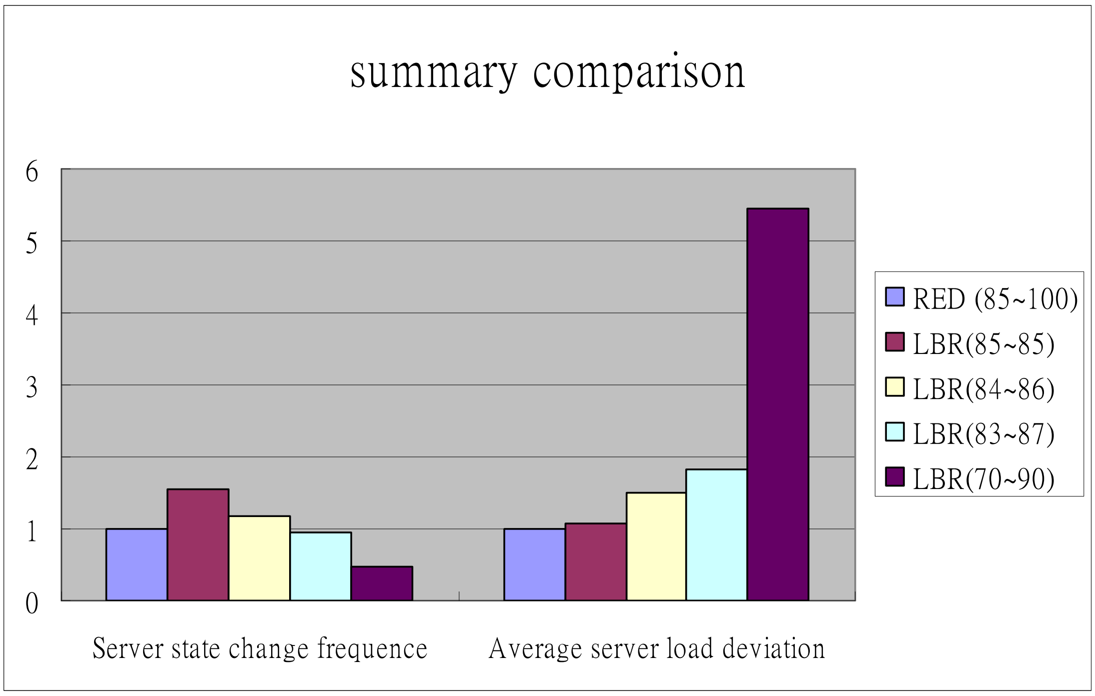


Fig. 9、Summary comparison of RED and LBR methods

Chapter 4: Server-based Load Balancing

For the convenience of management, service servers in most case are placed in geographical concentrated area. Now, we will aim at the load balancing in a server cluster.

4.1 Overview

The characteristics of load balancing in a server cluster are as follows:

1. Service servers are placed in geographical centralized area.
2. Information exchange between service servers can be more immediately.

The managements of server cluster can be classified to centralized (dispatcher-based) and distributed (server-based), and the comparisons are as follows:

4.1.1 Overview of centralized load balancing

Centralized load balancing means there is a particular dispatcher in a server cluster which is responsible for monitor the state of all service servers in the server cluster, its advantages and disadvantages are as follows:

1. Advantages:
    - High performance: dispatcher with knowledge of all service server state information and all user job requirements can make a more comprehensive decision-making to get better performance.
    - Simple: service servers only need to focus on the execution of client requests without doing any additional decision-making.
2. Disadvantages:
    - Worse response time and scalability: since the single dispatcher is responsible for the entire decision-making, hence whole system

bottleneck may occur in this single dispatcher results in response time increasing and system scalability decreasing.

- Low reliability: as the above reason, this single dispatcher failed would lead to whole system service outage. Hence its reliability is low.

4.1.2 Overview of distributed load balancing

Distributed load balancing means all service servers need to exchange their state information to each other and cooperate to do the decision-making of load balancing. Its advantages and disadvantages are as follows:

1. Advantages:
   - High scalability: compared with centralized load balancing architecture, distributed load balancing architecture can achieve higher system scalability through apportioning the works of decision-making among service servers.
   - Better response time: through apportioning the works of decision-making among servers can achieve relative better response time.
   - High reliability: load balancing is done by the cooperation of all service servers; a small number of service servers failing would not drastically affect the entire whole system's operation. Hence its reliability is relative higher.
2. Disadvantages:
   - Complicated: service servers need to spend some extra costs to carry out the decision-making of load balancing, and service servers are difficultly to obtain complete system state information, hence it needs more complicated design to achieve good load balancing degree.

- More information packets in LAN: because of the need of all service servers cooperate to do load balancing, the communication packets between service servers in server cluster would increase.

4.2 Overview of four of distributed load balancing policies

Because of the increasing demands for Internet services, service servers providing Internet services are growing as well. Hence distributed load balancing architecture with advantages of high reliability, high scalability, and a better turnaround time, etc, gradually replaces centralized load balancing architecture. In general, the load balancing processes of distributed load balancing architecture can be partitioned into the following four policies:

1. Information Policy: determine the kinds of state information exchanged among service servers.
2. Transfer Policy: determining the service servers which are needed to be adjust. For example, as a service server load greater than the maximum threshold, its state would be a sender and tends to find another service server with lower load to take over part of its works. On the other hand, if a service server load less than the minimum threshold, its state would be a receiver and tends to get part of works from another service server with higher load.
3. Location Policy: as a service server state becomes sender or receiver, it needs to find a matching service server at this phase to take over or to obtain part of works.
4. Selection Policy: after determining the matching service server, the service server with sender or receiver state would decide which work should be taken over in this policy.

4.3 Design of four distributed load balancing policies

After introducing the objective of the four policies of distributed load balancing architecture, we will illustrate the detail design of these four policies for load balancing. Before designing these four policies, we would first assume this distributed system is resource-aware and capacity-aware, that is we can know the resource requirements of each jobs and the capacity of each server for every resources. Then we would define what load balancing is. As the example in Fig. 10, there are two persons, and their weight-bearing capacities are 60kg and 40gk respectively. How should we allocate 50kg of goods to them? We think that if we do not consider their weight-bearing capacities and just allocate 50/2=25kg to each of them is not fair. On the contrary, they should have the same level of the load L after we allocate goods to them. Through the following formula:

$$60 * L + 40 * L = 50$$
$$L = 0.5$$
$$60 * 0.5 = 30, 40 * 0.5 = 20$$

We can get the load level of each of them should be 50%, and can further get they should carry 30kg and 20kg of goods respectively.

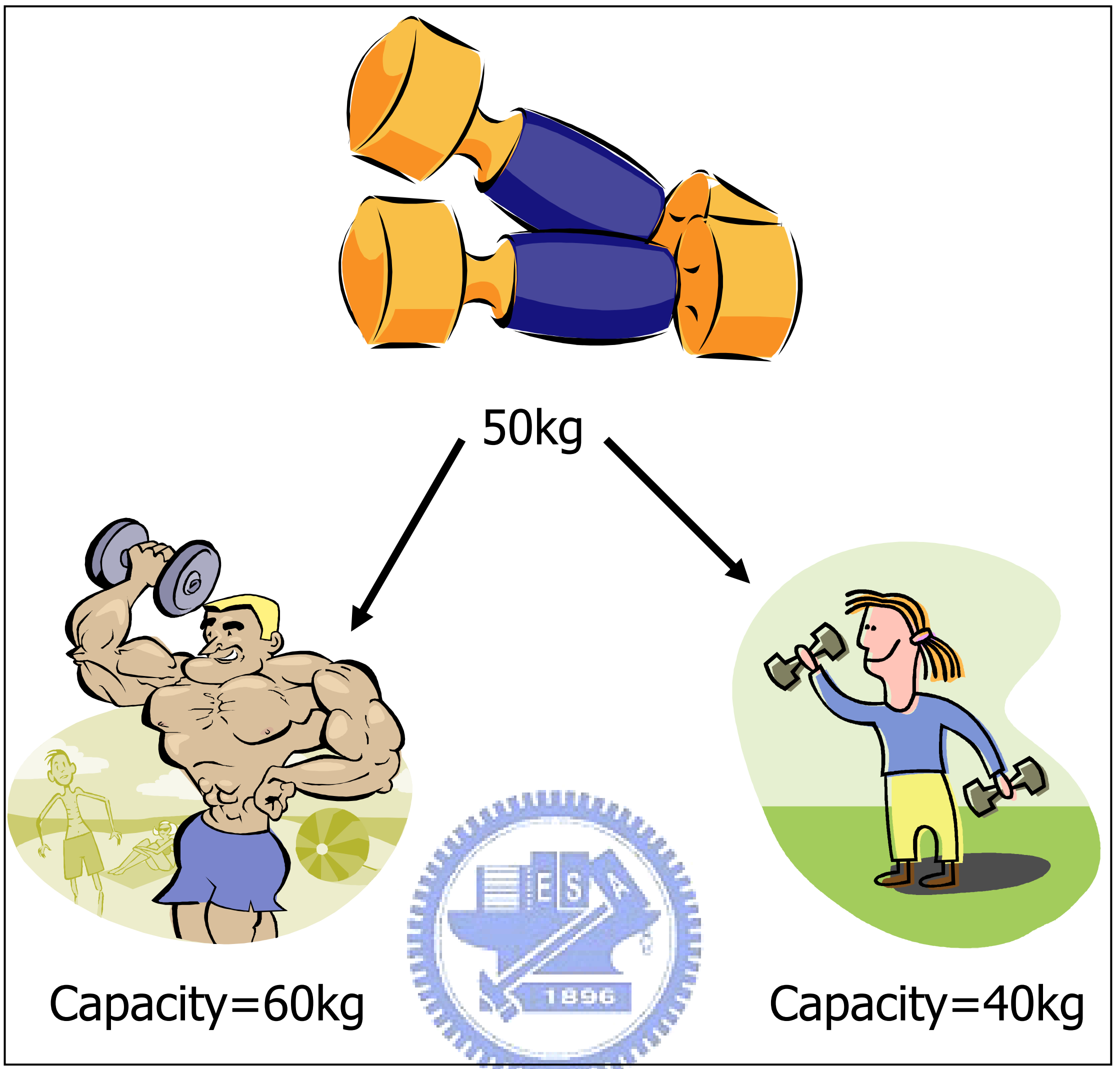


Fig. 10、Example of load balancing

4.3.1 Information policy

As we can see from above example, in order to achieve good load balancing degree, each service server should first know the capacity of all the other service servers, and the total amount of current requirements. Hence we define the information which service servers should exchange to each other as follows:

1. C_j^k: Capacity of server j for resource k.

2. L_j^k: Load of server j for resource k. (including waiting queue and execution queue)

Because we assume this system is resource-aware and capacity-aware, hence L_j^k can be obtained easily, and this value may greater than 100% due to it includes the jobs in the waiting queue. With knowledge of above information of the other service servers, we can get the requirements of jobs in each service server and the total requirements of jobs in this system. Assuming there are M service servers and K kinds of resources, there are some other symbols definition as follows:

- $R_j^k = C_j^k \times L_j^k$: Total requirements of the jobs in server j for resource k.
- $R_{Total}^k = \sum_{j=1}^{M} R_j^k$: Total requirements of the jobs in this server cluster for resource k.
- $L_{avg}^k = \frac{R_{Total}^k}{\sum_{j=1}^{M} C_j^k}$: Average load of whole server cluster for resource k.

Noteworthy the calculation of average load could be $\frac{\text{sum of service server load}}{\text{number of service servers}}$ in homogeneous environment, however, it should be $\frac{\text{sum of total requirements}}{\text{sum of service srver capacities}}$ in heterogeneous environment. As shown in Fig. 11, for example, there are two service servers with 100% and 50% of load respectively, if using old method would get the average load is $\frac{50\% + 100\%}{2} = 75\%$, but the correct average load is $\frac{50 + 50}{100 + 50} = 66.7\%$, hence these two service servers should use 66.7% of load as the goal to adjust, and finally execute 67 and 33 of the workloads respectively.

- $L_j^{avg} = \frac{\sum_{k=1}^{K} L_j^k}{K}$ ： Load of server j.

- $L_{avg}^{avg} = \frac{\sum_{k=1}^{K} L_{avg}^k}{K}$ ： Load of whole server cluster.

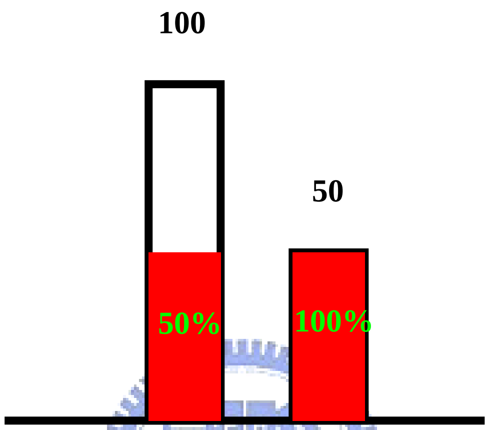


Fig. 11、Example of load calculation

4.3.2 Transfer policy

The common strategy is to use a threshold to determine the state of a service server. According to current load of a service server, the maximum threshold and the minimum threshold of whole server cluster, we can classify service server's state to sender, receiver and common. The maximum threshold Ts and the minimum threshold Tr are usually defined as follows:

- Ts: Average server cluster load plus a constant ($L_{avg}^{avg} + th$) or multiplied by a ratio which is greater than 1 ($L_{avg}^{avg} \times (1+r)$)。

- Tr: Average server cluster load minus a constant ($L_{avg}^{avg} - th$) or multiplied by a ratio which is less than 1 ($L_{avg}^{avg} \times (1-r)$) 。

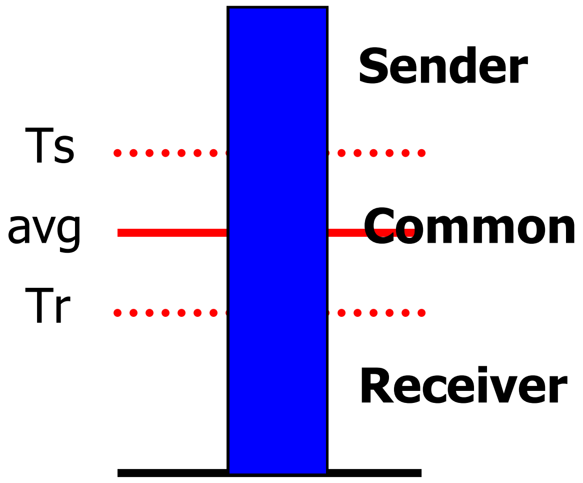


Fig. 12、Diagram of transfer policy

As shown in Fig. 12, we can define the state of a service server as follows:

- Sender: The service server with load greater than Ts. Tending to take over part of jobs to another service server with a lower load.
- Receiver: The service server with load less than Tr. Tending to take over part of jobs from another service server with a higher load.
- Common: The service server with load between Ts and Tr. Do not have to do any job take over.

4.3.3 Location policy

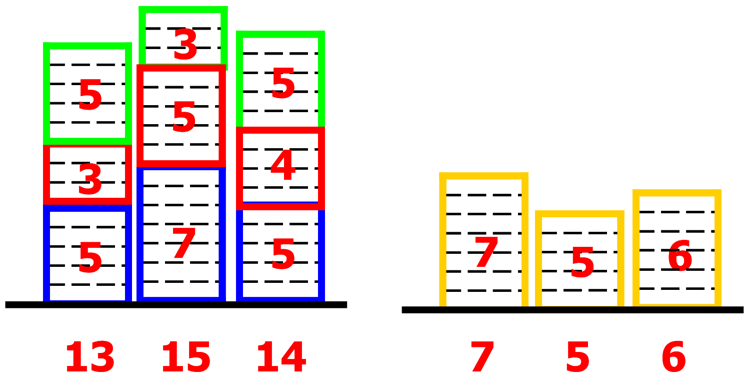
After determining the states of service servers, the service server with sender state needs to find an adaptable service server to take over part of its jobs, and the service server with receiver state needs to find an adaptable service server to get some jobs. The simplest transfer policy is that the service server with sender state sends part of its jobs to the service server with the lowest load in this server cluster, and the service server with receiver state gets some jobs from the service server with the highest load in this server cluster.

4.3.4 Selection policy

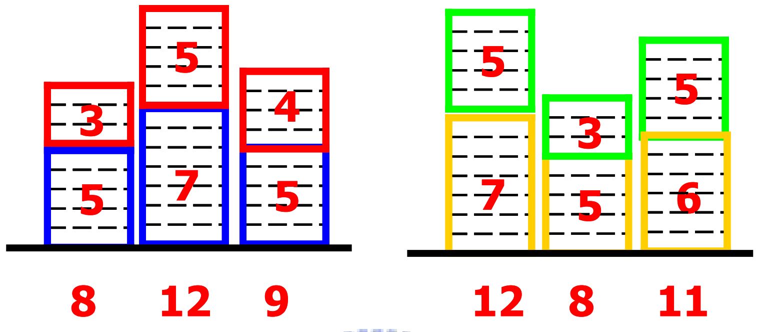
After finding the matching service server for the service server with sender or receiver state, then we need to determine which jobs should be taken over. Three of the most common methods are as follows:

1. Latest Arrival Job (LAJ): Sending the latest arrival job from the service server with a higher load to the service server with a lower load. This method just considers the load balancing degree between servers but not consider the load balancing degree of resources inside a server.
2. Backfill Lowest (BL): First finding the most available resource of the service server with lower load, and then take over the job which is most needed for that resource from the service server with higher load. This method would enhance the load balancing degree between servers and between resources inside a server.
3. Backfill balance (BB): Sending the job which can result in minimum $\frac{\text{maximum load}}{\text{average load}}$ value in the service server with a lower load from the service server with a higher load.

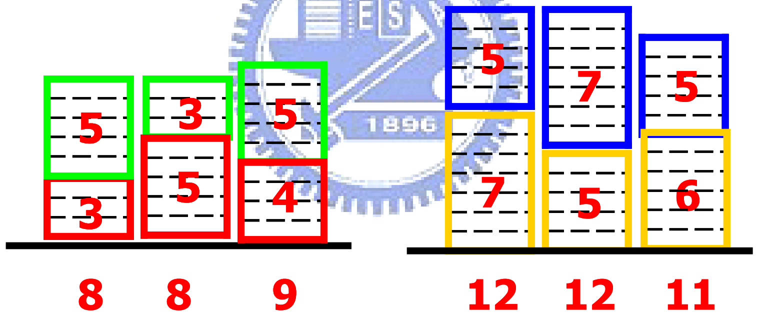
As shown in Fig. 13, there are two service servers with 3 jobs and 1 job respectively, and the load of their three resources are respectively 13, 15, 14 and 7, 5, 6 before doing job exchange as shown in Fig. 13 (a). At this time, the state of the left service server is sender, and tends to send a job to right service server. If using LAJ selection policy, the latest job left job with 5, 3, and 5 of the three resource requirements respectively would be sent to the right service server as shown in Fig. 13 (b). In BL selection policy, the most available resource of right service server is the second resource, and the job with 5, 7, and 5 of the three resource requirements of the left service server is most needed for the second resource, hence this job would be sent to right server as shown in Fig. 13 (c). In BB selection policy, the result of sending the three jobs of the left service server are $\frac{\max(7+5,5+3,6+5)}{\operatorname{avg}(7+5,5+3,6+5)} = \frac{12}{10.3} = 1.16$, $\frac{\max(7+3,5+5,6+4)}{\operatorname{avg}(7+3,5+5,6+4)} = \frac{10}{10} = 1$ and $\frac{\max(7+5,5+7,6+5)}{\operatorname{avg}(7+5,5+7,6+5)} = \frac{12}{11.7} = 1.03$ respectively. As we can see, the second job can get the minimum value, hence it would be sent to right service server as shown in Fig. 13 (d).



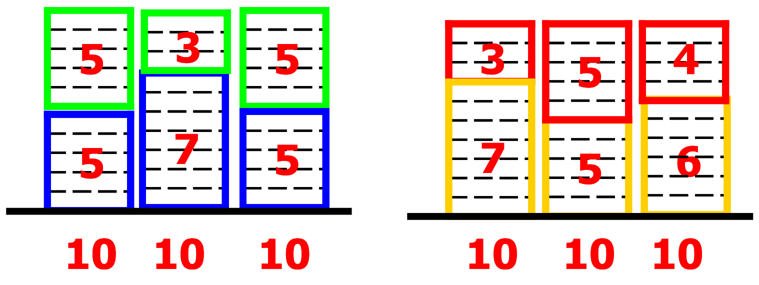
(a) Before job exchange



(b) Result of LAJ selection policy



(c) Result of BL selection policy



(d) Result of BB selection policy

Fig. 13、Example of selection policies

4.4 Market Mechanism (MM) load balancing method

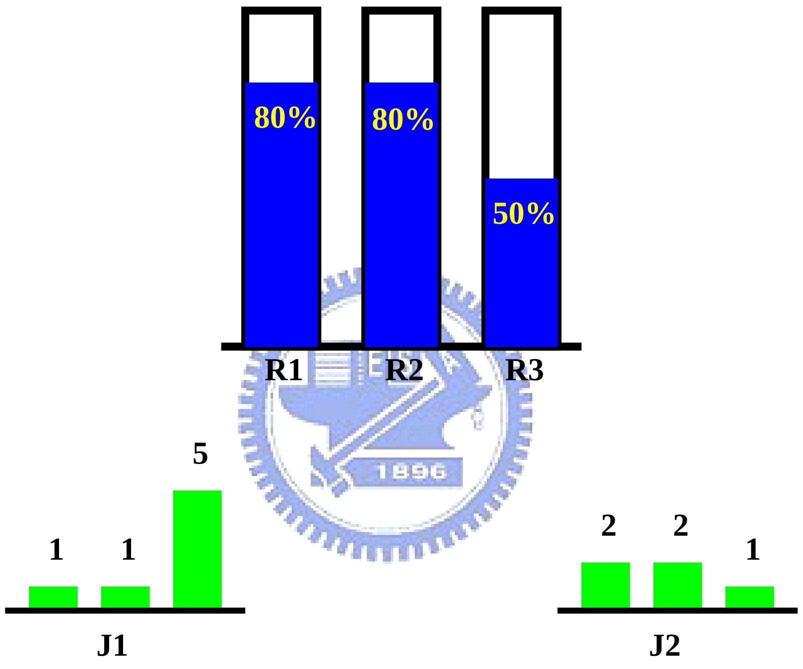
Conventional distributed load balancing method usually have not bad load balancing between service server, but the load balancing degree between resources inside a service server is poor. This is because when the load balancing degree between service servers is good enough, then even if the load balancing degree between resources inside a service server is poor, it does not have chance to address this condition because of its common state. For this reason, we present a distributed market mechanism load balancing method which would add a changer state for service servers to further improve the load balancing degree between resources inside a service server to raise the utilization of whole server cluster.

4.4.1 Definition of cost

At first, we think the cost of a job executed on a service server should be directly proportional to the load of that service server, that is, the higher the load of a service server is, the higher the cost of a job executed on that service server is. We define the cost per requirement of a job w executed on a service server j as:

$$Co_w^j = \frac{\sum_{k=1}^{K}\left(J_w^k \times L_j^k\right)}{\sum_{k=1}^{K} J_w^k} \qquad \text{Eq. 1}$$

J_w^k is the requirement of a job w for resource k. The reason of why we define the cost per requirement rather than the total cost of a job executed on a service server is that we hope to be more realistically aware of the adaptability of a job executed on a service server. As the example shown in Fig. 14, there is a service server with 80%, 80%, and 50% of its three kinds of resource load respectively and two jobs J1 and J2 whose requirements of these three kinds of resources are 1%, 1%, 5% and 2%, 2%, 1% respectively. If we just calculate the total cost of those two jobs we would get the J2 as 370 which is less than 410 of J2's, however, this is because the total requirement of J2 is less than J1's. In fact, it is not difficult to find that J1 is more adaptable than J2 to be executed on this service server. Therefore, if we calculate the cost per requirement of these two jobs executed on this service server, we would get the cost per requirement of J1 as 58.57 which is less then 74 of J2's. Hence J1 is more adaptable than J2 to be executed on this service server.



$$1\times80+1\times80+5\times50=410 \qquad 2\times80+2\times80+1\times50=370$$

$$\frac{1\times80+1\times80+5\times50}{7}=58.57 \qquad \frac{2\times80+2\times80+1\times50}{5}=74$$

Fig. 14、Example of cost calculation

After defining the cost per requirement of a job executed on a service server, we would do some modification for the four policies of distributed load balancing architecture to make our distributed market mechanism load balancing method work in a server cluster.

4.4.2 Information policy of distributed MM load balancing method

Conventional load balancing methods usually just consider the load of a single resource of a service server, such as the load of CPU L_j^1. Now, we would do a multiple resources load balancing, hence service servers should exchange the load of all their resources $L_j^1 \sim L_j^K$ to each other.

4.4.3 Transfer policy of distributed MM load balancing method

As mentioned in 4.3.1, L_{avg}^k represents the average load of whole server cluster for resource k, now we can define the load imbalancing degree of a service server as:

$$B_j = \sum_{k=1}^{K} \left(L_j^k - L_{avg}^k \right) \qquad \text{Eq. 2}$$

This would detect the load imbalancing between service servers. And the absolute load imbalancing degree of a service server is defined as:

$$Ba_j = \sum_{k=1}^{K} \left| L_j^k - L_{avg}^k \right| \qquad \text{Eq. 3}$$

This would detect the load imbalancing between resources inside a service server. Then we start to determine the state of a service server as follows:

- Sender: $Ts < B_j$
- Receiver: $B_j < Tr$
- Changer: $Tr \le B_j \le Ts$ and $T < Ba_j$, then its state would be a changer. Where T is a threshold.
- Common: service server with none of the above states would be common state.

This method is broadly similar to conventional method. The difference is that we add a changer state for service servers to further improve the load balancing degree of resources inside a service server.

4.4.4 Location policy of distributed MM load balancing method

The location policies for the sender and receiver are the same as for the conventional method.

It is different from the service server with sender and service server with receiver state, which would respectively send jobs to and receive jobs from their matching service servers, the service server with changer state should exchange jobs with its matching service server. We define the load imbalancing degree before a service server i with changer state and its matching service server j exchange jobs as:

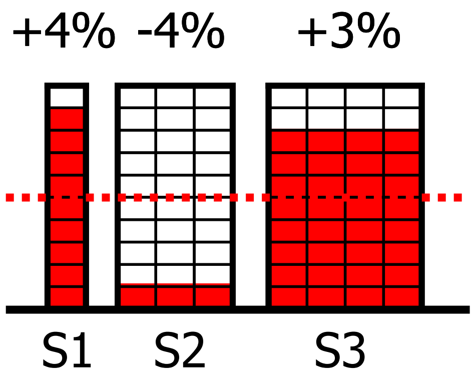
$$Ba_i + Ba_j$$

And the ideal load imbalancing degree after pairing service server i and service server j would be:

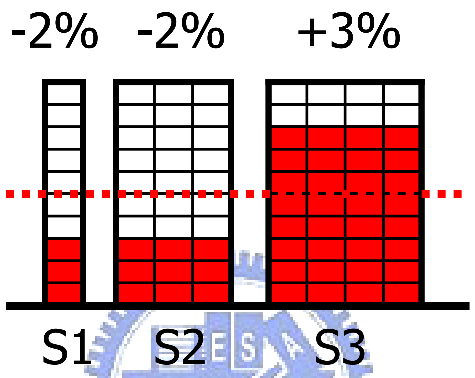
$$2 \times \sum_{k=1}^{K} \left| \frac{\left(L_i^k - L_{avg}^k\right) \times C_i^k + \left(L_j^k - L_{avg}^k\right) \times C_j^k}{C_i^k + C_j^k} \right| \qquad \text{Eq. 4}$$

The greater the load imbalancing degree before service server i and service server j exchange jobs minus the load imbalancing degree after they exchange jobs is, the better to pair these two service servers together is.

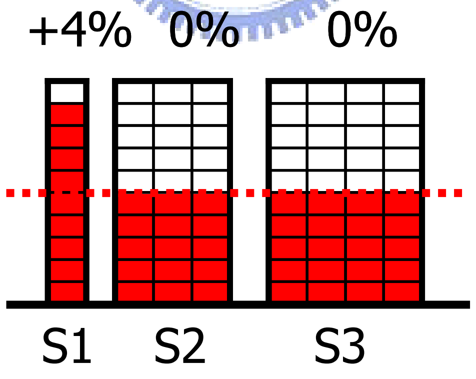
As an example shown in Fig. 15 (a), there are three service servers, the ratio of their capacities is 1:3:4 and their current loads compared to the whole system load are +4%, -4%, and +3% respectively. Before the job exchange, Ba_1, Ba_2, and Ba_3 are respectively 4, 4, and 3. Without considering the service server's capacity, we may tend to pair service server 1 and service server 2, however, the imbalancing degree before pairing service 1 and service server 2 is |4| + |-4| = 8, and the imbalancing degree after pairing service server 1 and service server 2 is $2\times\left|\frac{4\times1+(-4)\times3}{1+3}\right|=2\times2=4$ as shown in Fig. 15 (b), hence the improvement of pairing service server 1 and service server 2 is 8 – 4 = 4. If we pair service server 2 and service server 3, then before pairing, the imbalancing degree is |-4| + |3| = 7, and the imbalancing degree after pairing would be $2\times\left|\frac{(-4)\times3+(3)\times4}{3+4}\right|=2\times0=0$ as shown in Fig. 13 (c), hence the improvement of pairing service server 2 and service server 3 is 7 – 0 = 7 which is greater than the 4 gotten from pairing service server 1 and service server 2.



(a) Before job exchange



(b) Result of pairing service server 1 and service server 2



(c)Result of pairing service server 2 and service server 3.

Fig. 15、Example of location policy

4.4.5 Selection policy of distributed MM load balancing method

After pairing the service server with sender or receiver state with its matching service server, the service server i with a higher load would send its job w with a minimum value of $Co_{i,w}^{j}$ to the service server j with a lower load.

In the part of the service server with changer state and its matching service, we want the two service servers to respectively transfer a job which can give the most benefit to each other. We define the benefit of transferring a job w from service i to service server j is the cost of the job w be executed on service server i minus the cost of the job w be executed on server j, that is:

$$Co_{i,w}^{i} - Co_{i,w}^{j}$$

Assuming service server i and service server j would get the maximum benefit by transferring their job w1 and job w2 respectively, we must also provide that sum of the benefit of service server i and j respectively transfer its job w1 and w2 to each other should greater than a threshold T, that is:

$$\left(Co_{i,w1}^{i} - Co_{i,w1}^{j}\right) + \left(Co_{j,w2}^{j} - Co_{j,w2}^{i}\right) > T \qquad \text{Eq. 5}$$

Then the two service servers could be allowed to exchange jobs. This is because the job exchange should change the resource loads of both service servers, resulting in the cost of a job executed on these two service servers varying. If there is no threshold restriction, after service server i and service server j respectively transfer their job w1 and w2 to each other result in the resource loads of service server i and service server j varying, we may find that w1 and w2 become respectively appropriate to be executed on service server i and service server j again. This would lead to job w1 and w2 continuing to be exchanged between service server i and j without converging.

4.5 Simulation

In this thesis, we use C language to simulate the process of jobs with multiple resources requirements executed on distributed heterogeneous servers. We would compare the following five kinds of methods:

1. NO: Once a job is assigned to a service server, it sould not be transferred to another service server anymore.
2. LAJ (Latest Arrival Job): Using the four distributed load balancing policies of the conventional method as mentioned in section 4.3, and the LAJ selection policy as illustrated in section 4.3.4.
3. BL (Backfill Lowest): Using the four distributed load balancing policies of the conventional method as mentioned in section 4.3, and the BL selection policy as illustrated in section 4.3.4.
4. BB (Backfill Balance): Using the four distributed load balancing policies of the conventional method as mentioned in section 4.3, and the BB selection policy as illustrated in section 4.3.4.
5. MM (Market Mechanism): Our method as mentioned in 4.4.

4.5.1 Simulation environment

As illustrated in [2][6][9], we assume that a job would be executed on a service server only after all its requirements can be satisfied by that service server. Moreover, in order to emphasize the heterogeneous environment, we made the following setting for jobs and service servers respectively:

- Job: a job needs three kinds of resources and the requirements of each resource is independently random from 1 to 9. The execution time of a job is random from 5 to 25.

- Service server: there are three heterogeneous degree as follows:
  - Homogeneous: the capacities of a service server for each resource are 200.
  - Heterogeneous (150~250): the capacities of a service server for each resource are independently random number from 150 to 250.
  - Heterogeneous (100~300): the capacities of a service server for each resource are independently random number from 100 to 300.

We respectively simulate the situation of 2^1, 2^2 to 2^7 servicer servers, and get the result by averaging 20 times simulation results with different random seeds. Each job would be randomly assigned to any one of the service servers first. In our simulation, we control the request traffic between 70% and 80% of the capacity of this server cluster.

4.5.2 Metrics

We would use the following four metrics to compare the result of each method:

1. Average Standard Deviation of Server Load: the load balancing degree between service servers. Smaller is better.
2. Average Standard Deviation of Resource Load: the load balancing degree between resources of a service server. Smaller is better.
3. Server Utilization: system throughput. Larger is better.
4. Average Turn Around Time: the elapsed time from a job was generated to it finished, also called response time. Smaller is better.

4.5.3 Result

We first observe the average standard deviation of service server load. As shown in Fig. 16, when service servers are homogeneous, compared with the NO method, which doesn't detect load imbalancing between service servers, load balancing methods, such as LAJ, BL, BB, and MM have relatively better load balancing degrees, and the effect of using which selection policies is not obvious. And as shown in Fig. 17 and Fig. 18, as the heterogeneity of service servers increases, the variation of the results of the NO method is small, and the result of those methods (LAJ, BL, and BB) which just detect the load imbalancing among service servers would become worse as the heterogeneity of service servers increases. Compared with the above methods, as the heterogeneity of service servers increases, our MM method not only detects load imbalancing between service servers, but also detects load imbalancing between resources inside a service server could keep good average standard deviation of service server load. And as the heterogeneity of service servers increasing, the gap of our MM method and the other methods is more obvious.

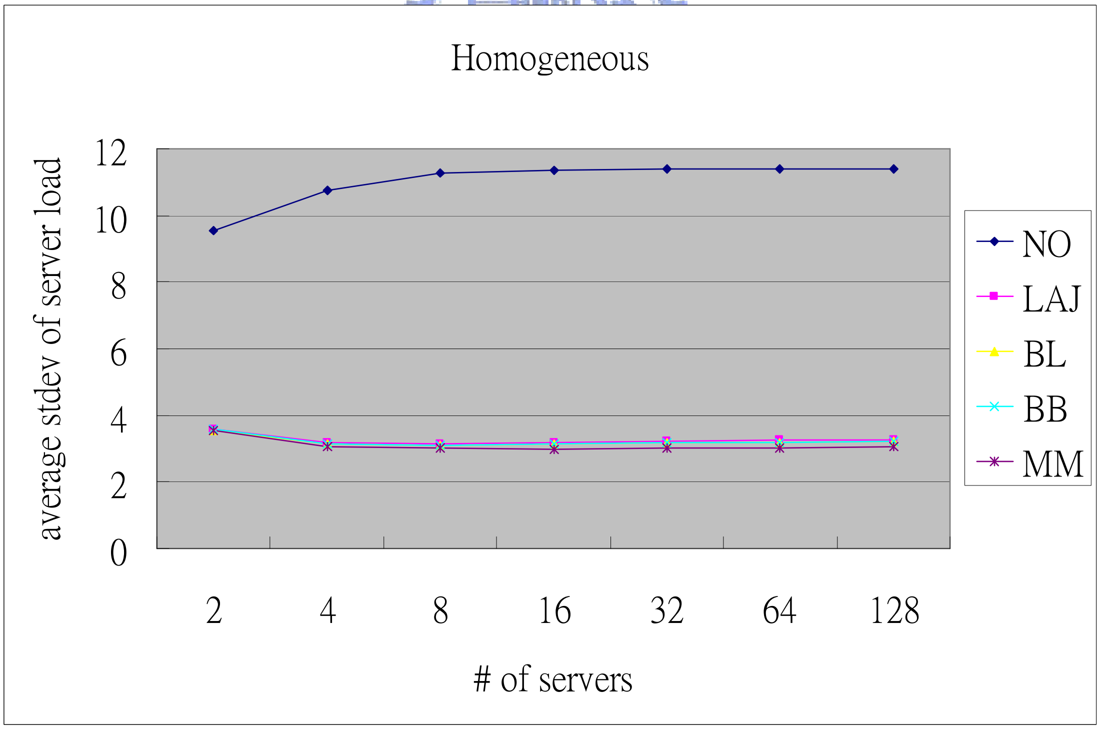


Fig. 16、Average standard deviation of server load in homogeneous environment

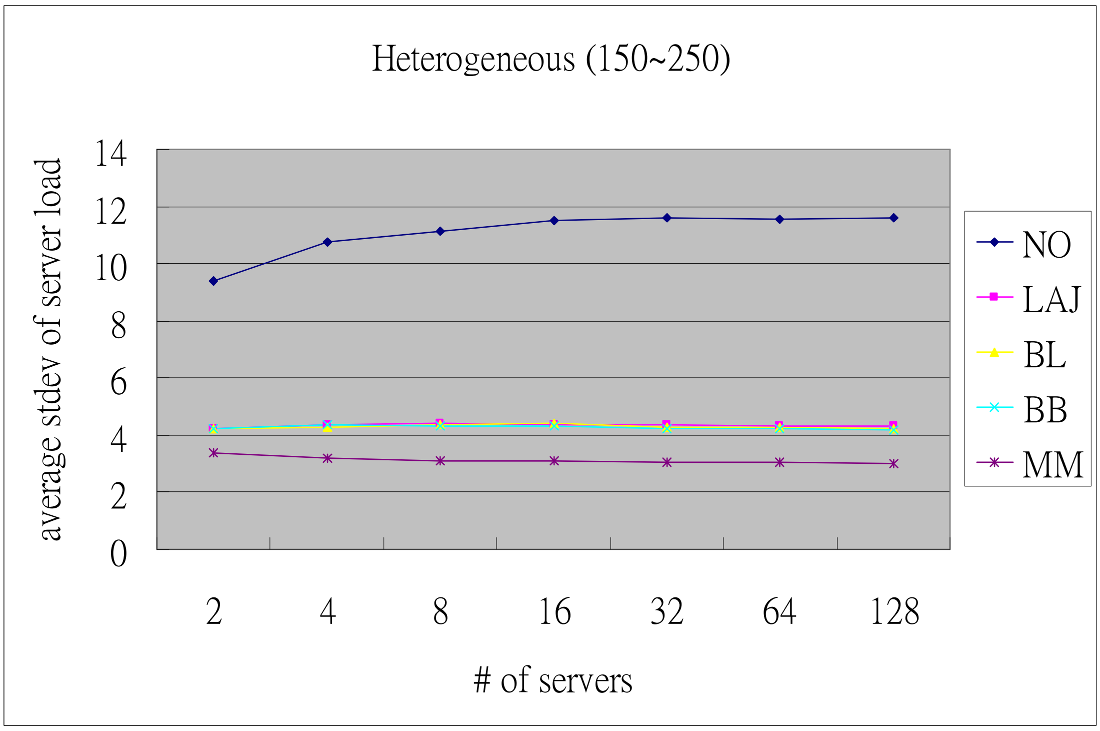


Fig. 17、Average standard deviation of server load in heterogeneous (150~250) environment

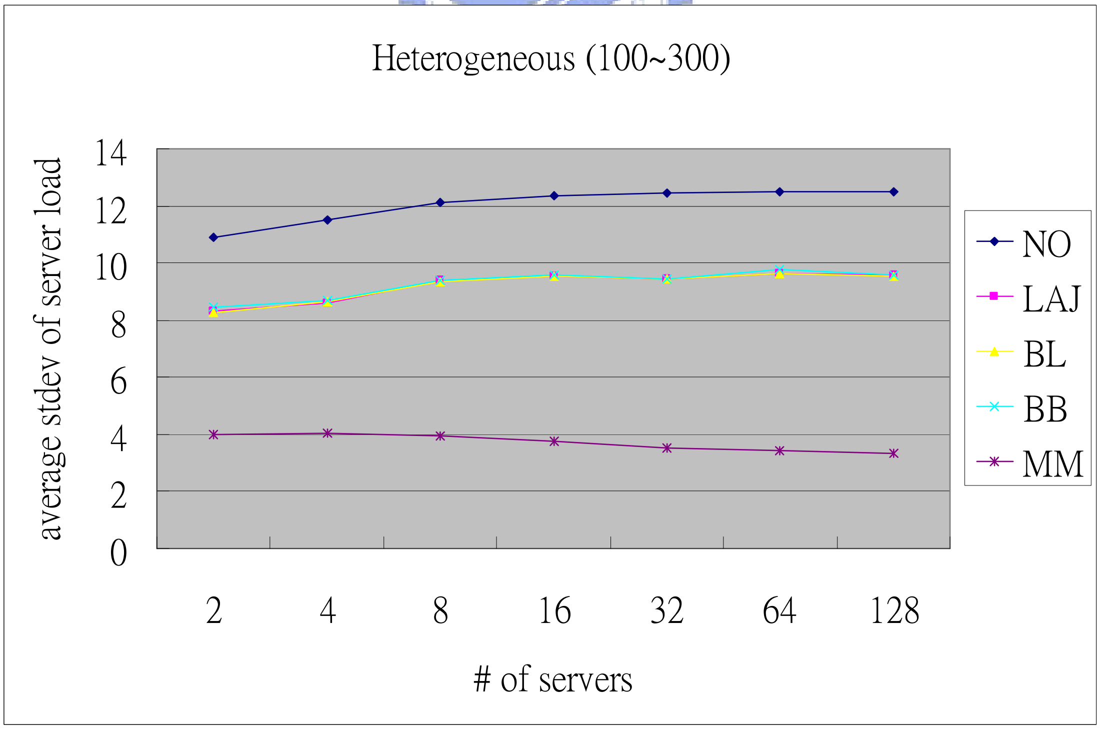


Fig. 18、Average standard deviation of server load in heterogeneous (100~300) environment

Then we observe the standard deviation of resource load. As shown in Fig. 19, in the homogeneous environment, the methods whose selection policy considers multiple resources, such as BL, BB, and MM, have got ahead of the LAJ and NO methods, and BB method is slightly ahead of BL method. However, our MM method is obviously leading the other methods. On the other hand, as shown in Fig. 20 and Fig. 21, as heterogeneity of service servers increasing, the result of NO, LAJ, BL, and BB method are rapidly deteriorating, but our MM method is still maintaining a good load balancing of resources.

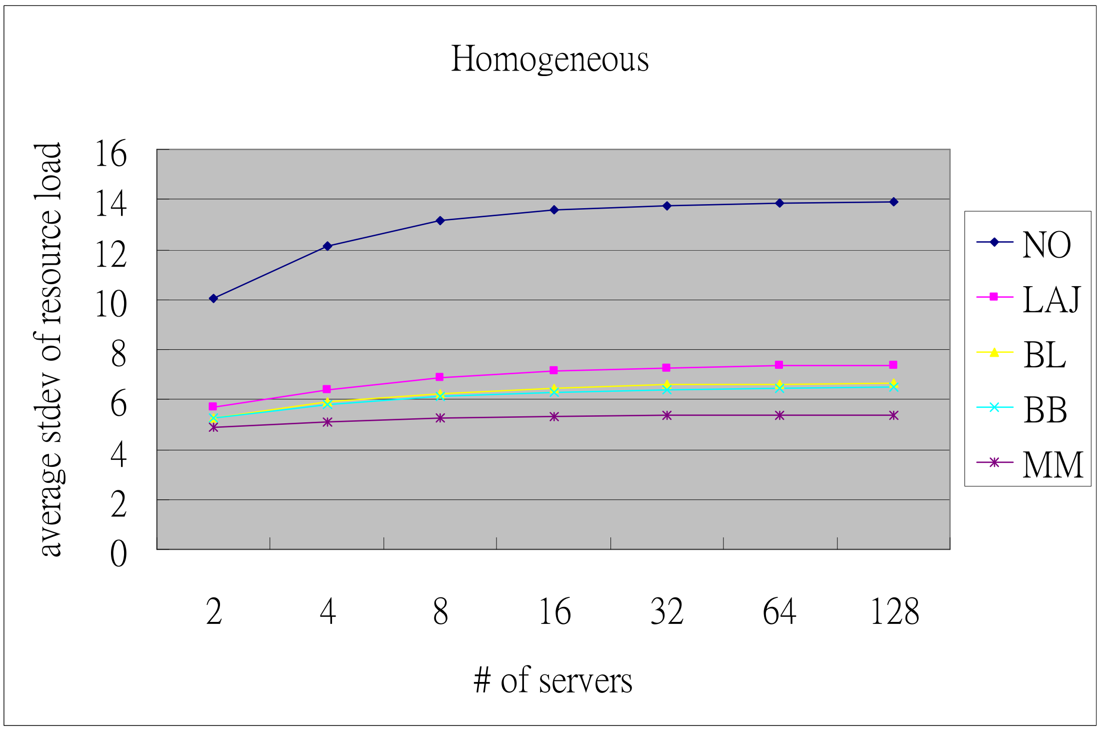


Fig. 19、Average standard deviation of resource load in homogeneous environment

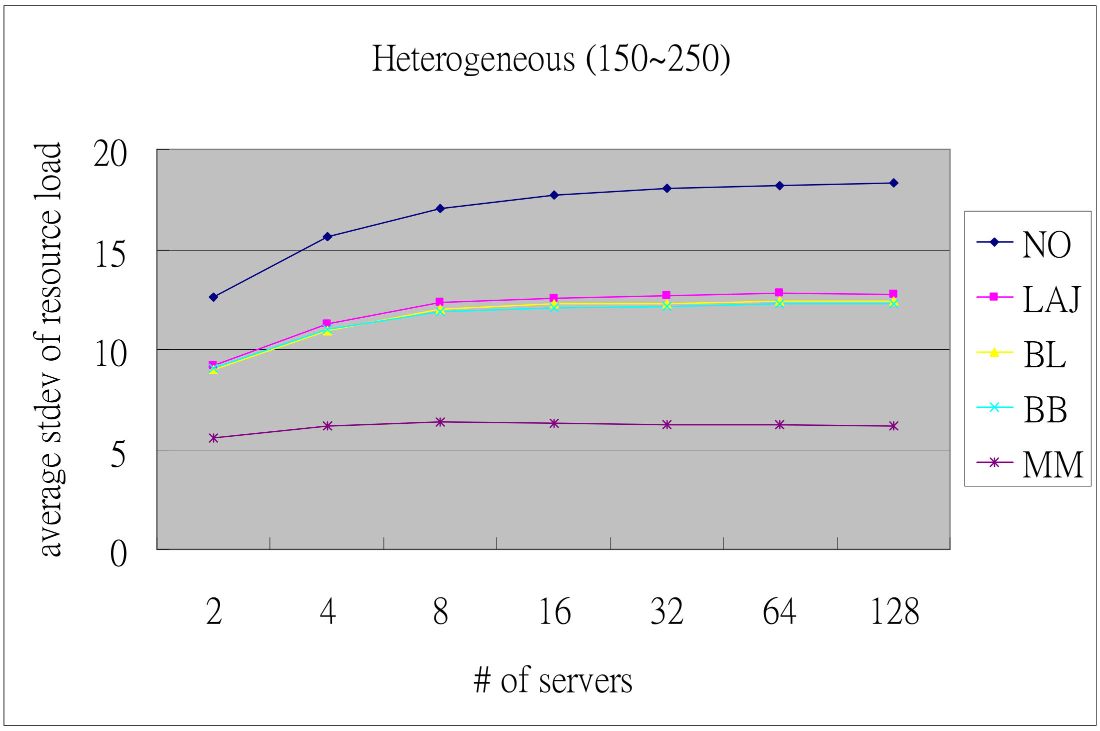


Fig. 20、Average standard deviation of resource load in heterogeneous (150~250) environment

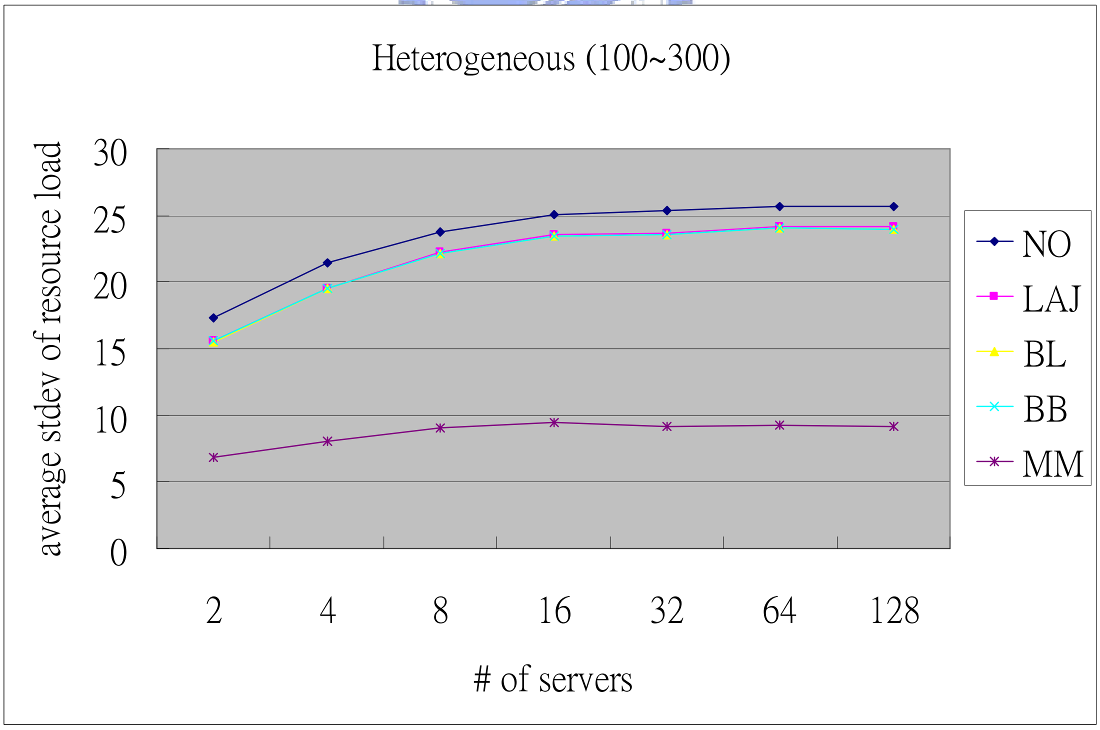


Fig. 21、Average standard deviation of resource load in heterogeneous (100~300) environment

In the part of average service server utilization, as shown in Fig. 22 and Fig. 23, in the homogeneous environment, the service server utilization of each method is similar, even the NO method without dynamic load balancing can get the similar average service server utilization to the other methods. However, as the heterogeneity of service servers increases, NO method starts to face the bottleneck condition derived from a small number of resources overloading. As shown in Fig. 24, when the heterogeneity of service server is higher, the average service server utilization of NO, LAJ, BL, and BB methods have started to decrease, especially the NO method. Nevertheless, the average service server utilization of our MM method is still similar to it in the homogeneous environment.

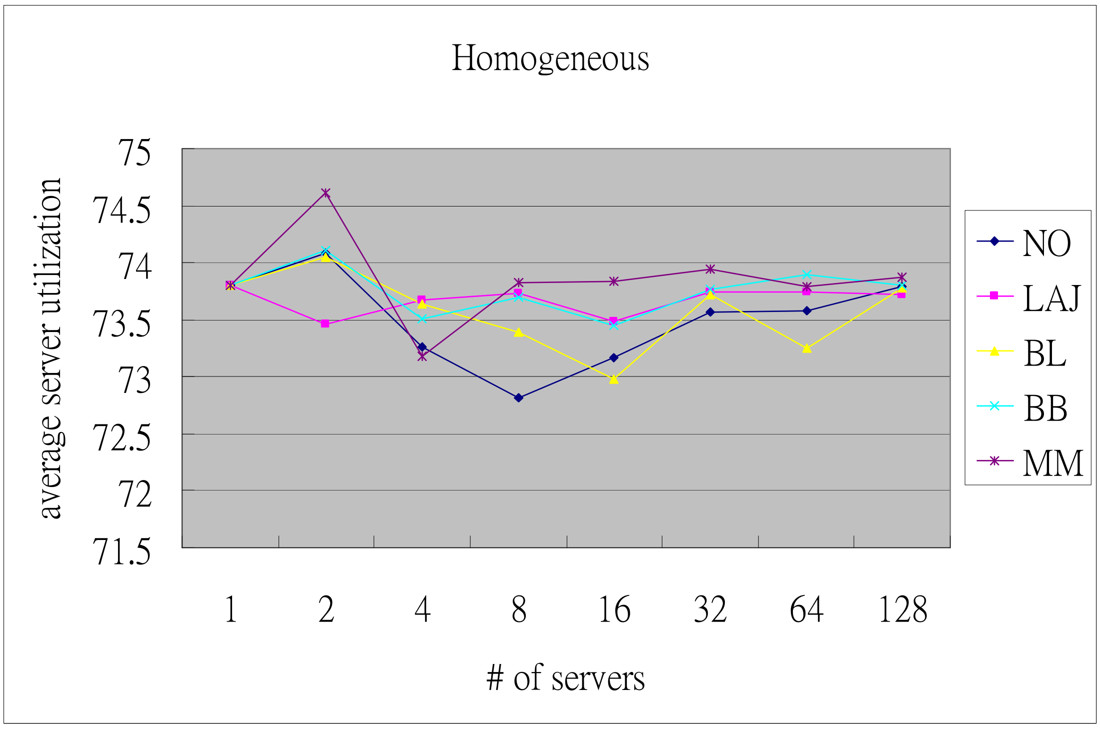


Fig. 22、Average server utilization in homogeneous environment

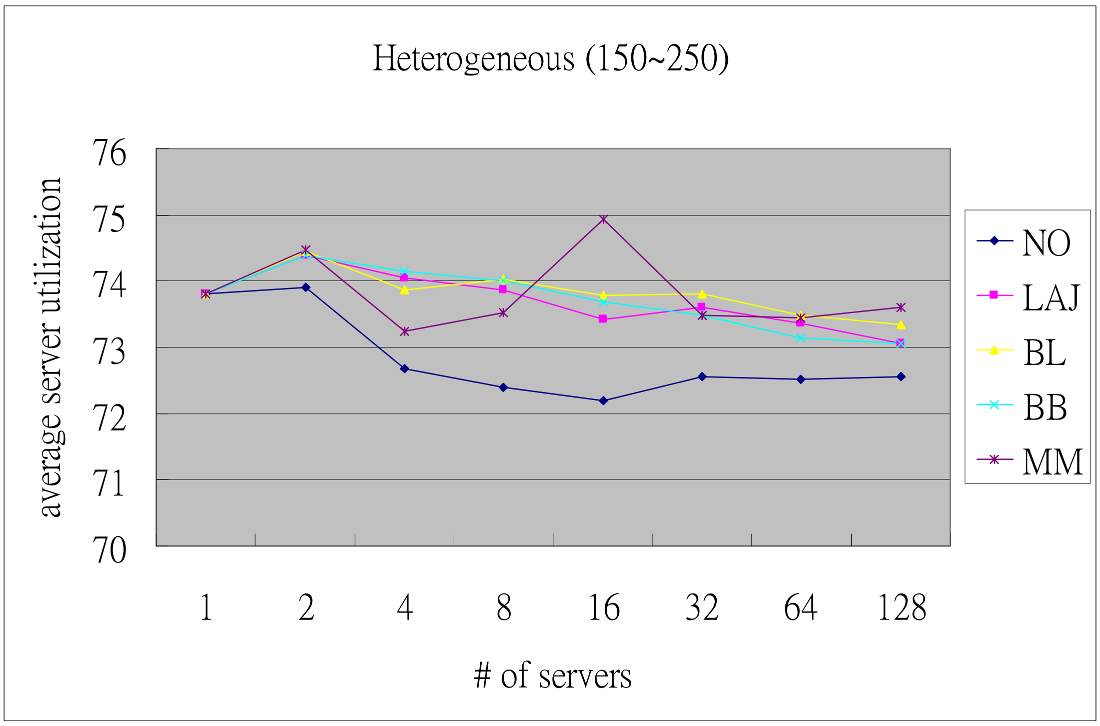


Fig. 23、Average server utilization in heterogeneous (150~250) environment

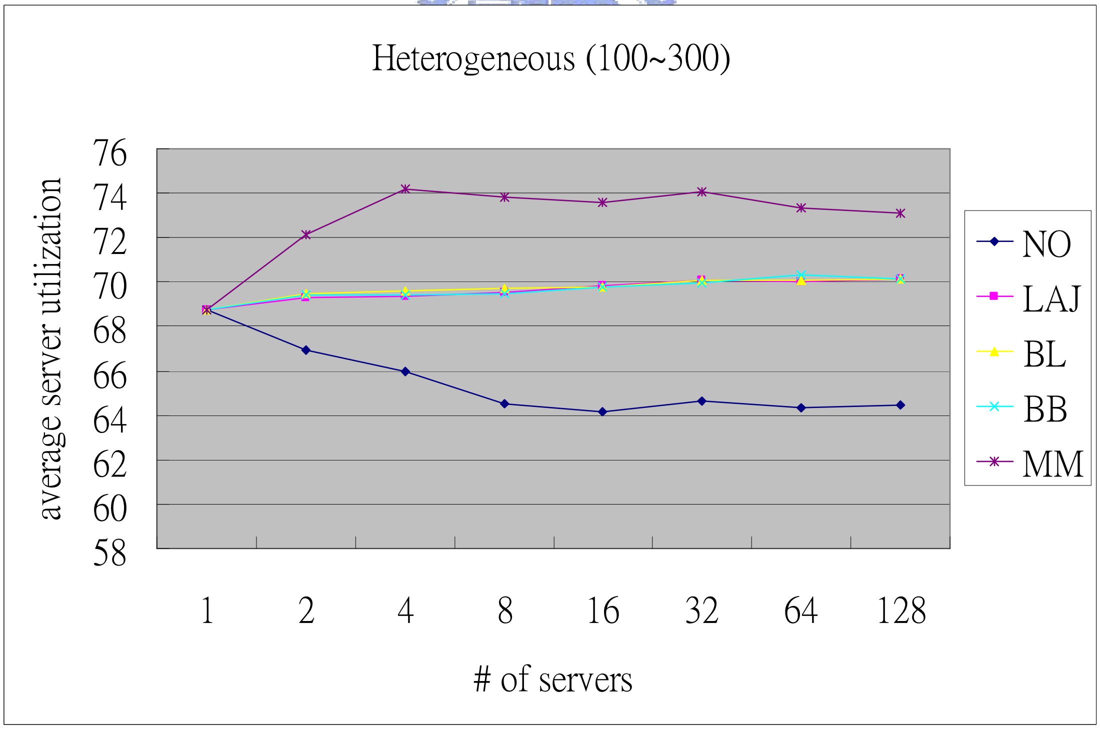


Fig. 24、Average server utilization in heterogeneous (100~300) environment

Finally, we will analyze the average turn around time of each method. Because of the execution time of a job is random from 5 to 25, the average execution time of a job is about 15, hence the more closer to 15 the average turn around time is, the better the load balancing method is. As shown in Fig. 25, in a homogeneous environment, although the NO method has similar service server utilization as the other methods, however, its average turnaround time is worse than the other methods. And as the heterogeneity of service servers increases, as shown in Fig. 26, each average turnaround time of the methods except MM method is worse and worse, and the gap between the NO method and the other methods also becomes larger and larger. On the other hand, as shown in Fig. 27, in the heterogeneous (100~300) environment, when the number of service servers is small, the heterogeneity of service servers is too big to get a good average turnaround time, but as the number of service server increases, our MM method can obtain a similar average turnaround time as in the homogeneous environment.

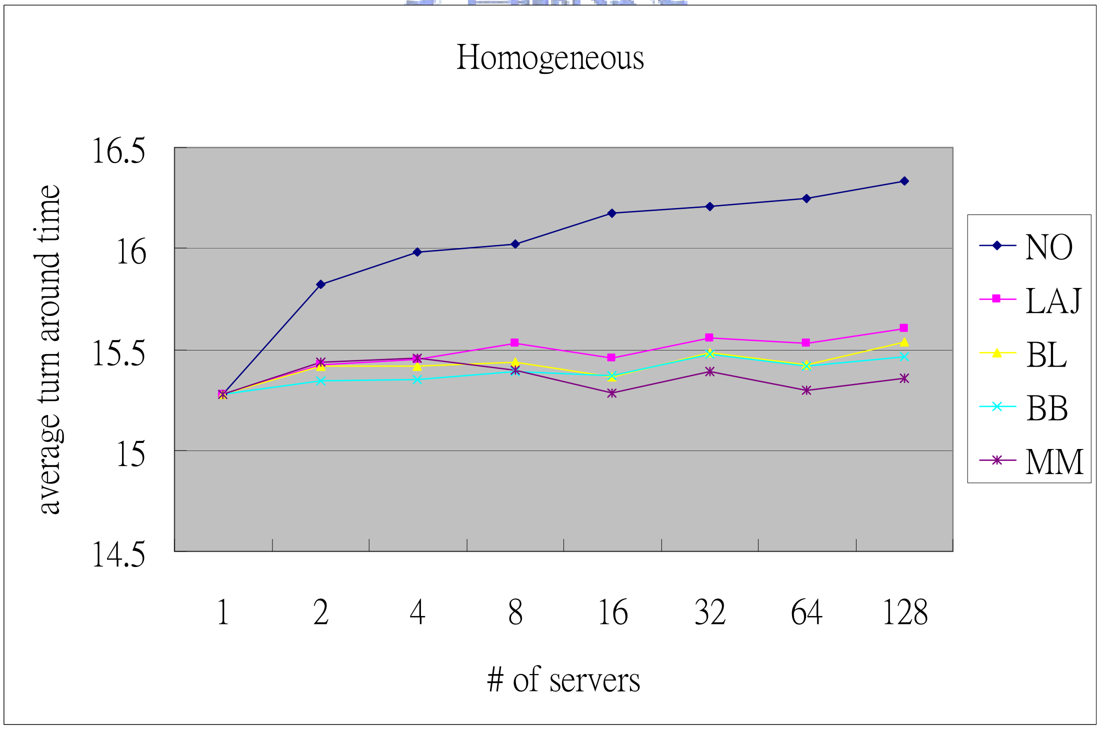


Fig. 25、Average turn around time in homogeneous environment

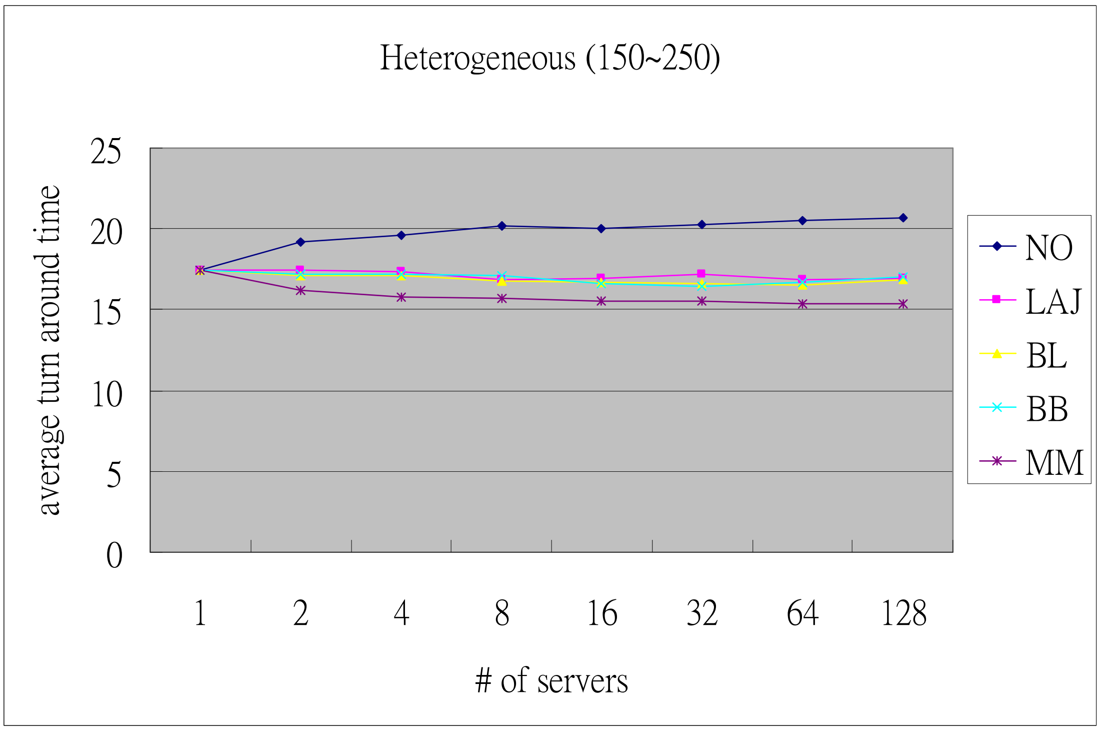


Fig. 26、Average turn around time in heterogeneous (150~250) environment

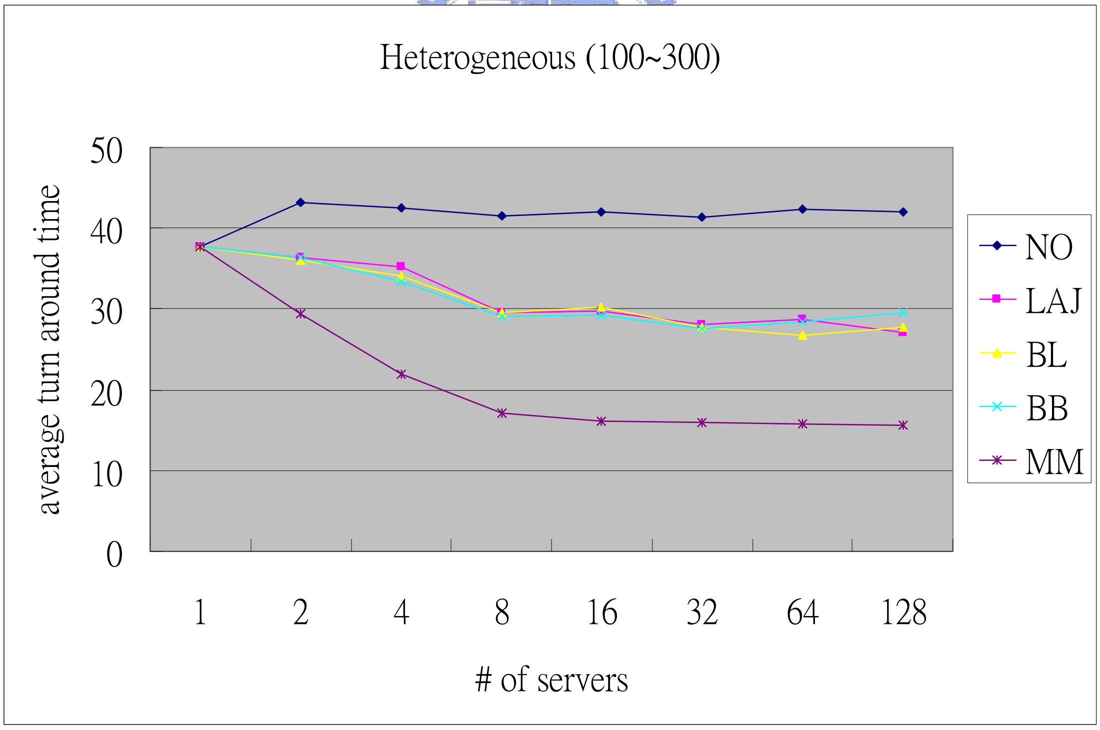


Fig. 27、Average turn around time in heterogeneous (100~300) environment

Chapter 5: Conclusions and future works

In the thesis, we integrate geographic distributed load and geographic centralized load balancing method. We first use DNS server to do a simple and efficient load balancing among geographic distributed server clusters, and then use a more complicated, heterogeneous, and multiple resources considering load balancing in a server cluster.

Compared with the conventional load buffer range method, our RED method can efficiently reduce the average standard deviation of service servers load to 1/5 of the conventional method’s, smooth the load variation of service servers, and provide more stable quality of services. Moreover, in our simulation, no matter what we set the load buffer range of the conventional method to, it could not get better load balancing degree than our RED method.

In a server cluster, we present a distributed market mechanism load balancing method which would consider the consumption of multiple resources, and heterogeneity between service servers at the same time. In the transfer policy, we add a changer state for service servers. Therefore, service servers can dynamically adjust the load imbalancing of their resources. And in selection policy, compared with other methods, our market mechanism method can more effectively select the job to be sent to achieve higher load balancing degree.

In our simulation, we show that our distributed MM load balancing method can efficiently avoid a system bottleneck derived from the lack of a small number of resources while other resources are idle. Compared with previous methods, our distributed MM load balancing method can raise the average service server utilization about 4%~9%, reduce 2 to 3 times of the average standard deviation of service servers and resources, keep the average turnaround time low compared with the other methods, and provide high system performance.

In future works, we will consider more realistic workloads to do load balancing. In that case, the resources of service servers would more likely to be consumed by a small number of jobs. Hence the four policies of distributed load balancing should be more carefully designed to determine the states of service servers and the jobs to be taken over.

References:

[1] V. Cardellini, M. Colajanni, and P. S. Yu, “Dynamic Load Balancing on Web-Server Systems,” IEEE Internet Computing, MAY/JUNE 1999.

[2] YD Lin, CM Tien, SC Tsao, RH Feng, and YC Lai, “Multiple-resource request scheduling for differentiated QoS at website gateway,” Computer Communication, JAN. 2008.

[3] H. Bryhni, E. Klovning, and O Kure, “A Comparison of Load Balancing Techniques for Scalable Web Servers,” IEEE Network, July/August 2000.

[4] K. Tumer, and J. Jawson, “Collectives for Multiple Resource Job Scheduling Across Heterogeneous Servers,” In Proc. of the second international joint conference on Autonomous agents and multiagent systems, July 2003.

[5] M. Arora, S. K. Das, and R. Biswas, “A De-centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments,“ In Proc. of the International Conference on Parallel Processing Workshops, 2002.

[6] W. Leinberger, G. Karypis, and V. Kumar, “Load Balancing Across Near-Homogeneous Multi-Resource Server,” Heterogeneous Computing Workshop, Aug. 2000.

[7] D. L. Eager, E. D. Lazowska, and J. Zahorjan, “A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing,” ACM, 1985.

[8] N. G. Shivaratri, P. Krueger, and M. Singhal, “Load Distributing for Locally Distributed Systems,” Computer, Dec. 1992.

[9] W. Leinberger, G. Karypis, and V. Kumar, “Job Scheduling in the presence of Multiple Resource Requirements,” ACM/IEEE, 1999.

[10] Z. Zhang, and W. Fan, “Web server load balancing: A queueing analysis,” European Journal of Operational Research, Feb. 2007.

[11] M. Colajanni, and P. S. Yu, “Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers,” Distributed Computing Systems, May 1998.

[12] M. E. Crovella, M. H. Balter, and C. D. Murta, “Task Assignment in a Distributed System: Improving Performance by Unbalancing Load,” Measurement and Modeling of Computer Systems, Oct. 1998.

[13] M. Aramudhan, and V. R. Uthariaraj, “LDMA: Load Balancing Using Decentralized Decision Making Mobile Agents,” Lecture note in computer science, 2006.

[14] G. Ciardo, A. Riska, and E. Smirni, “EquiLoad: a load balancing policy for clustered web servers,” Performance Evaluation, 2001.

[15] C. Lu, and S.M. LAU, “An Adaptive Load Balancing Algorithm for Heterogeneous Distributed System with Multiple Task Classes,” In Proc. of the 16th International Conference on Distributed Computing Systems, May 1996.

[16] T. Osogami, M. Harchol-Balter, A. Scheller-Wolf, and L. Zhang, “Exploring Threshold-based Policies for Load Sharing,” In Proc. of the 42th Annual Allerton Conference on Communication, Control, and Computing, Sep. 2004.

[17] M. Harchol-Balter, “Task Assignment with Unknown Duration,” ACM, 2002.

[18] A. Shaikh, R. Tewari, and M. Agrawal, “On the Effectiveness of DNS-based Server Selection,” In Proc. of IEEE Infocom, 2001.

[19] D. D. Clark, and W. Fang, “Explicit Allocation of Best-Effort Packet Delivery Service,” IEEE/ACM Transactions on Networking, 1998.

[20] E. Rahm, and R. Marek, “Dynamic Multi-Resource Load Balancing in Parallel Database Systems,” In Proc. of the 21th VLDB Conference, 1995.

[21] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, “A Scalable Solution to the Multi-Resource QoS Problem,” IEEE Real-Time Systems Symposium, 1999.

[22] S. Floyd, and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” IEEE/ACM Transactions on Networking (TON), 1993.