Offering different Services by Server Clusters[1]

Jing LIU, Mingyang ZHENG and Jiubin JU

Computer Science Department, Jilin University, Changchun 130012,P.R.china

(E-mail:jjb@mail.jlu.edu.cn)

Abstract Nowadays operating systems on single servers have encountered a lot of trouble; server clusters are not able to offer different services for different clients. Under this circumstance, we put forward a new idea, that is, different clients are given different priority domains, thus can gain different services. We improve current resource management mechanism, and use it on server nodes in clusters. Experiment results show that the method is effective, and a cluster can provide different services to different clients, therefore the performance of server clusters are improved.

Keywords server cluster, priority, resource management, different service

1 Introduction

As the Internet develops, people's requirements for quality of service become higher and higher, so Web servers must grow stronger and stronger accordingly. Requests from people are very different. A general manager may want to use urgent messages. Employees may need to read news about world cup. So Web servers should also be able to provide different services for different users.

Now general-purpose operating systems are most popular for their good performance and cheap price, but they provide insufficient support for the resource management in large-scale servers. So researchers put much stress on server clusters, which consist of some common PCs to get high performance. Here we wish to improve web servers' performance, and provide different services to different clients, too.

Current general-purpose operating systems and clusters are not perfect in these aspects:

- On single machines: General-purpose operating systems are lack of efficient resource management.

General-purpose operating systems, such as UNIX, are originally designed to support large-scale timesharing systems efficiently. Applications running on them are independent of each other and run in user mode most time. But in modern servers, applications usually consist of a number of cooperating processes; they spend so much time inside the I/O subsystems of the OS kernel. In current operating systems' resource management mechanisms [1], processes are resource principals, which resources of the system are allocated to and resource consumptions are counted for, and they can't fit modern server applications. Firstly, in server applications mainly providing network services, a lot of procedures are processed in the kernel. But in general-purpose operating systems, kernels don't control or count resource consumed by applications on network communications, which lead to resource miscounting and reallocation incorrectly. Secondly, some

[1] Project 60073040 supported by NSFC(China)

applications consist of more than one process, which cooperate together to perform one independent activity and should be regarded as one resource principal. But in general-purpose operating systems, one process is one resource principal; these applications will use more resource than they should. Thirdly, some applications consist of only one process, which perform more than one independent activity. General-purpose operating systems only control resource allocation and consumption of the whole process, and can't calculate the internal resource consumption of different activities. Under this circumstance, we should improve the resource management mechanism of general-purpose operating systems imminently.

- On server clusters: Server clusters should provide different services to different clients.

Now researches are mainly on high availability[2], load balancing[3], etc on server clusters. But the shortcoming on resource management of operating systems in server nodes are not noticed, and researches on this aspect are not perfect. It's difficult to provide different services on a distributed and cluster-based server. If different clients are regarded differently so that requests from high level users (such as payers) will receive high quality services (less response time and low latency), then users will be satisfied.

We improve the resource management mechanism of current operating systems, and use it in server nodes of clusters. Then bring out a scheme to provide different services on server clusters, which will improve the performance of large-scale servers.

2 Improving current resource management mechanism

To overcome the shortcoming of resource consumption miscounting on network process in general-purpose operating systems, Gaurav, etc bring out the resource container abstract [4,5] which set up a new resource management mechanism. It's a new and effective resource management facility that binds a resource container to an activity; the resource container is a resource principal. It controls resource consumption clearly and carefully. We improve it further.

In this new resource management mechanism, a resource principal is responsible for an activity, including all the resource needed by the activity. That is, scheduling is related to an activity, not a thread or process. The system scheduler allocates resource to an activity, regardless of how the activity is mapped to threads. The kernel carefully counts resource consumption of each principal, which later system scheduler is able to access, and then control how to schedule the resource requests of correlative resource principals. Sometimes a resource principal will be related to several protect domains, for example, the request for CGI resource by a client. At this moment, all the resource consumed in these protect domains relevant to the request are calculated to the only resource principal. On the other side, a process contains several resource principals, such as an event-driven server application. This time, the server still creates a resource principal for each relation, and each resource principal records the consumption of its client connection. All resource principals in the system make up a hierarchy structure, in which resource use of a child container is constrained by its parent container.

From above we can see that this resource management mechanism can correctly calculate the resource consumption of each activity. Then we give a priority to each resource principal, according to which the system does its scheduling, and principals with higher priorities will gain more resource than those with lower

priorities. Here the priority of a principal is limited to a domain, called a priority domain. The priority of a principal can be changed for justice when the principal consumes too much resource, but the priority must not out of its priority domain, because we provide different services to different clients, and we must ensure service quality of high-level clients. Although high-level clients' priority may be depressed if they consume too much resource, with the existence of the priority domain, they will obtain better services, too.

The effective use of this resource management mechanism will need LRP[6] -Lazy Receiver Processing. LRP processes kernel network I/O in the context of the concerned application. It integrates the resources consumed in network processing into the system's global resource management framework. LRP replaces the system's global IP queue with a per-socket queue. The network interrupt handler de-multiplexes incoming packets according to their destination sockets, and puts them to the right receive queue. When the receive queue is full, discards them by default. With the receiver protocol processing, the received packets are processed at the priority of the receiving principals, not like a general-purpose operating system, which is eager processing, disturbing the execution of the current resource principal. All the sender protocol processing is also processed at the priority of sending principals. With the above method, LPR can count the resource consumption of network processing to correct resource principals.

3 Scheme of offering different services in clusters with priority

After extending the above resource management mechanism to server clusters, different services will be obtained among plenty of clients. Requests of high-level clients will be assured of quick response. Now let's analyze lottery scheduling algorithm[7], and then discuss the offering of different services in server clusters with priority.

Lottery scheduling algorithm

The basic thought of Lottery scheduling is distributing lotteries for all sorts of system resources (such as CPU cycles). The system scheduler chooses a lottery at random in the time of making decisions, and the process owning the lottery will gain system resource. In the realization of the algorithm, the more important processes are given more lotteries than others in order to gain more chances. For example, if there are 100 lotteries in all, and a process has 30 ones, then the process has the possibility of 30 percent to be scheduled. After a long time running, it will approximately gain 30 percent of CPU time. And because each client with more than zero lotteries will gain a lottery at last, starvation will be avoided.

Lottery scheduling is very suitable for the internal scheduling in the new resource management mechanism. A resource principal will gain resource according to how many lotteries it possesses. This character is fit for providing different services for different clients.

Server clusters providing different services according to priority

First, let's make some definitions:

Definition 1: resource

This refers to all the shared system entities needed when serving, and is multiplexed by the system among several services, such as CPU cycles, memories, disks, etc. In this article it mainly refers to CPU resource.

Definition 2: service quality

This refers to how well a server can serve clients, mainly refers to whether the response is correct, how much time is the response time, and whether the response is simple, etc. Good service quality is the premise of winning clients.

Definition 3: client level

All clients for service are classified into some levels. Different services are provided to clients of different levels. Web servers probably will provide different service quality to different clients, such as users paying for service or not, leaguers or not. Different users should get different server resources (CPU, memory, disk space). Providing different services for different clients is in prevailing these days. Classification of client levels can by needed, such as contents requested by clients, or IP addresses of clients, or how much fee a client pays, etc.

Definition 4: priority domain

It is the range between the upper limit and the lower limit of the domain. Each client level has a priority domain, and the priority of the resource principal varies in it. When a resource principal consumes too much system resource, the system will lower its priority, but the new priority is still in the priority domain.

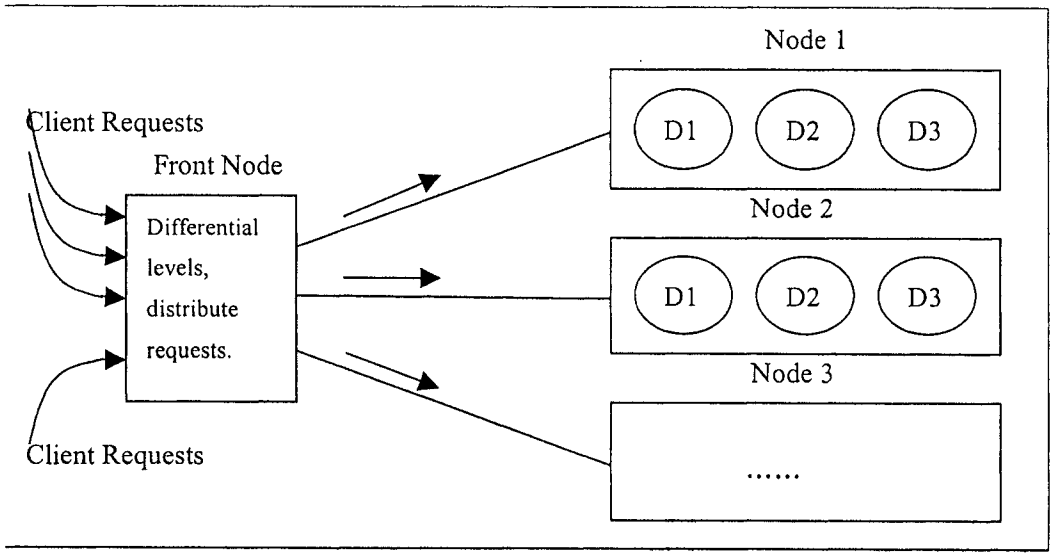

Figure 1: A server cluster provides different services for clients

Most structures of current server clusters are a front end with several back end servers[8,9], the front end is responsible for receiving requests and distributing them to back end servers, and back end servers are responsible for resolving, processing and responding requests. The front end is a distributor for clients' requests.

The server cluster divides client levels by some rules, and processes requests of different client levels differently, expecting better services for high-level clients. When a client request arrives at the front-end machine of the server cluster, the front end recognizes its client level, chooses an idler back end server, and sends the request to it, in company with the client level. Then the back end server decides the priority domain

according to the client level, that is to say, the proportion of CPU resource is determined, which the resource principal serving the request can use. Surely resource principals relevant to high-level client requests will gain high proportional resources and gain high quality services due to the lottery-scheduling algorithm in the new resource management mechanism. In the whole server cluster, if a back end server is too busy to handle more requests, then it will inform the front end to distribute follow requests to other back end servers. And then the server cluster will satisfy client's needs of different services.

After giving a priority domain to each client level, servers will process clients' requests differently on receiving, processing and responding, and then higher-level clients will gain a better service. Also, clients' requests can be classified into many levels.

Figure 1 shows a server cluster can provide different services for clients. The main task of the front end is deciding which level a client request belongs to, and distributing the arriving request to an idler back end server. The back end server is responsible for receiving requests from the front end, creating new resource principals, and converting client levels into priority domains, as parameters of resource principals.

4 Implementation

The working process is as follows:

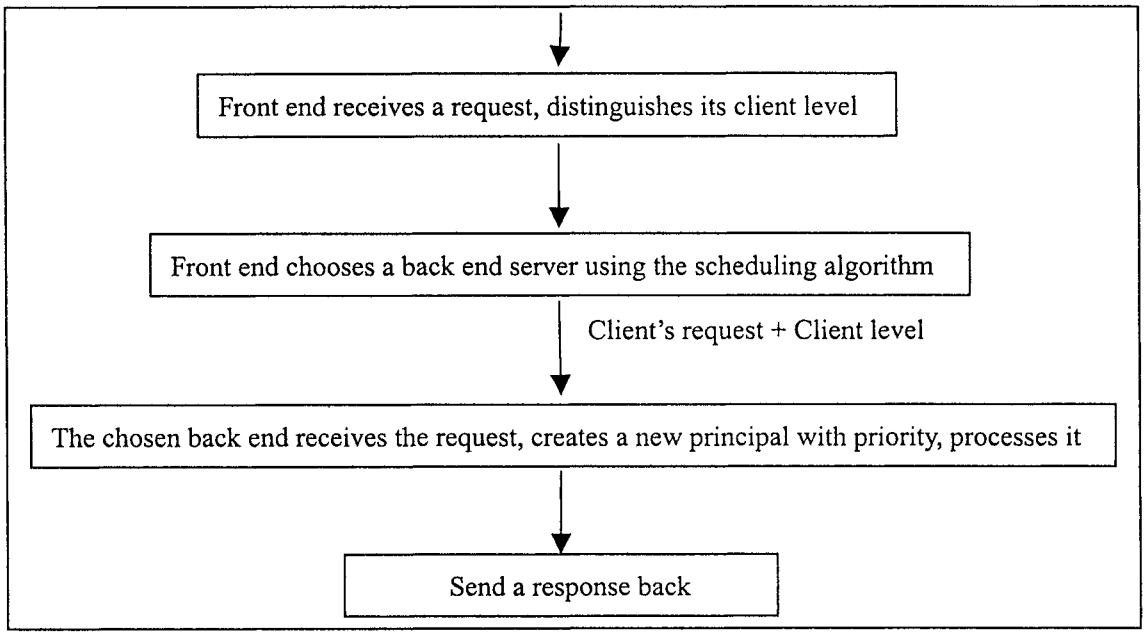

Figure 2: working process of a server cluster

Firstly, when a client's request arrives, the front end decides which client level--C the request belongs to. There are many means to do it, such as according to contents requested, or client's IP addresses, or fees paid by clients.

Secondly, the front end chooses a back end server with a lighter load to process the request, using a load

balancing scheduling algorithm. Then the client's request with its client level is sent to the chosen end. There are many scheduling algorithms can be used here, such as round-robin scheduling, weighted round-robin scheduling, least-connection scheduling, weighted least-connection scheduling, etc. We can also use many IP load-balancing techniques. One is NAT (network address translation), which relies on the fact that the headers of packets can be adjusted appropriately so that clients believe they are contacting one IP address, but servers at different IP addresses believe they are contacted directly by the clients. Another is IP tunneling, which encapsulate IP datagram within IP datagram, allowing datagrams destined for one IP address to be wrapped and redirected to another IP address.

Thirdly, the chosen back end server creates a new resource principal for the request, and decides corresponding priority domain to be a parameter of the principal according to the client level, waiting for scheduling. As the priority value of each principal varies only in its priority domain, the scheduling time and resource a principal can hold vary according to its different client level. On the whole, requests from higher-level clients will win higher quality service, vice versa.

Lastly, after the chosen back end server processes the request and gets a response, it either sends the response back to the front end sending response to the client, or sends it to the client directly. They can be used neatly.

If the server cluster is expected to strengthen fault tolerance ability, a back-up front end can be added to the cluster. When the main front end fails, the back-up front end can work as nothing wrong happens.

After the priority domain of a resource principal is decided, the priority is set to the upper limit at first time. It is lowed down as the whole system resource the principal consumes increases. Then the percentage of resource this principal can use lows accordingly, and other request's winning chances will increase. But the priority will not be under the lower limit of its priority domain, thus the priority of higher-level requests is still higher than lower ones. A principal's resource is used by processes that bound to it. As a result, requests are processed at different rates. This resource management is able to control the resource consumption of a respective activity in a fine grain, even if the activity spans over more than one protect domain, so the priority policy works well and clients of different level will gain different quality of service.

5 Performance Evaluation

In our experiment, the server cluster consists of a front end and two back end server nodes. The front end is responsible for receiving clients' requests outside and filtering client level of requests (here client requests are divided by source IP addresses), and then allocating requests to a chosen back end server. One of the back end server is a Celeron 500 MHz machine configured with 128M RAM. The other one is a Celeron 650MHz machine configured with 128M RAM. They both run the FreeBSD 4.0 operating system with the improved resource management mechanism. The modified Thttpd [10] Web server is used at the server nodes. Requests are generated by a client program based on the Sclient [11] architecture. A Pentium II 650 machine and a Pentium II 997 machine are clients. The client machines and all cluster nodes are connected via 10Mbps HUB.

Thttpd is an asynchronous Web server, with a process polling. In our experiment, Thttpd works on the

new resource management mechanism, serving the outside requests. Sclient is a tool measuring Web servers' ability. It can generate heavy concurrent traffic that has a temporal behavior similar to that of real Web traffic, evaluating server performance under overload. Two client machines run Sclient application, starting together to send requests.

Each time a client requests a 1K document. We find that if the request rate is over 120/s, the server cluster's serving ability descends. So we choose this rate as clients' request rate.

Here Web services are divided into 9 levels, and the higher, the better (can be adjusted at will). In the experiment, client 2 always belongs to level 1, but client 1 belongs to level 1, 3, 5, 7, 9 separately, then we deserve changes of average response time. Two client machines send requests to the server cluster together, and stop after 10 seconds. We record the time from connect establishing to service ending of each request, sum it up and compute the average value to be the average response time. Repeat several times, the results are showed in Fig.3.

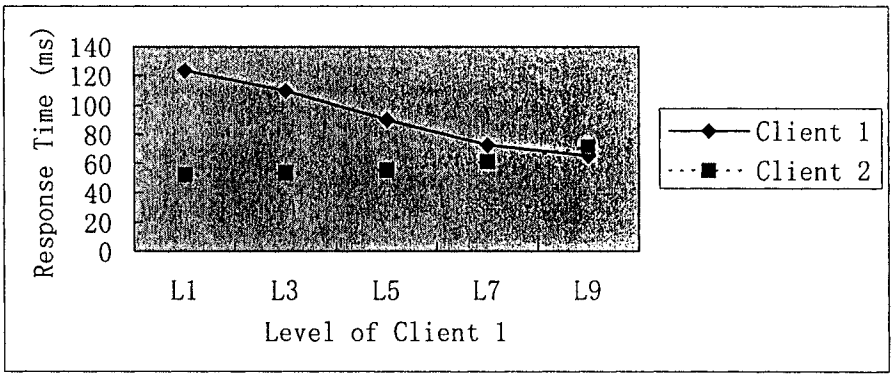

Figure 3: response time of clients

Client 2 belongs to level 1. In the drawing, the X-coordinate shows the average response time of client 1 whose levels range from level 1 to level 9; the Y-coordinate shows the average response time. The real curve denotes client 1's response time range. The broken line denotes client 2's response time range. The drawing proves that as the levels of client 1 ranges from 1 to 9, its average response time changes from 123.3783 ms down to 65.2789 ms, which means it gains service better and better; while the average response time of client 2 becomes longer and longer.

6 Related Work

Mohit etc. proposed cluster reserves[12] to manage system resources in cluster-based network servers. In the implementation, resource containers are used to achieve performance isolation on individual cluster nodes, and cluster reserves extend performance isolation into performance isolation for the cluster. This mechanism is better than others that provide separate server nodes for each service class. It can achieve performance isolation in several service classes; each service class defines a group of requests. Resources consumed by clients of each service class are counted and rescheduled respectively to ensure the service class will get resource it wants. But it must count the resource each service class consumed, and calculate how much

resource should reallocate to the service class next time, which waste server time. In contrast to it, in our scheme, requests are divided into different client levels, and clusters serve differently due to it. So each resource principal is allocated different system resources used by processes bound to it. In this kind of server clusters, the scheduling algorithm is simple and effective, and the response time is reduced. It efficiently realizes different service, and is easy to complete.

Almeida et al's work [13] wants to provide differentiated Quality-of-Service in Web hosting services. The work is on a modified Apache Web server, running on a general-purpose monolithic operating system. They use numeric process priorities to indicate differentiated QoS, experimenting both with a fully user-level implementation, and with a slightly modified Linux kernel scheduler. They are able to provide differentiated HTTP service to different QoS classes. However, the effectiveness of this technique is limited by their inability to control kernel-mode resource consumption. Also, this approach does not extend to event-driven servers. But our scheme is able to calculate resource consumption of each resource principal more precisely, including the consumption in the kernel, and gain different service in server clusters.

7 Conclusion

We have introduced the priority domain, a method to provide different services in server clusters. We divide priorities into several domains in server clusters, and resource principals that process different level clients' relations are given priorities belong to different priority domains. It effectively meets the needs of different quality of services in clusters. In the back-end server nodes, improved resource containers are used to process client requests, which ensure carefully counting and scheduling of resource consumed by each principal, including network-processing extending to the kernel. This effectively and simply realizes different services in server clusters.

Experiment results showed that priority domains are efficient, and improve the performance of server clusters.

Reference

[1] Maurice J.Bach. The Design of the UNIX Operating System. Beijing: China Machine Press, 2000.

[2] High-Availability Linux Project. http://linux-ha.org/.

[3] Corradi A. Diffusive Load-balancing Policies for Dynamic Applications. IEEE Concurrency, 1999,7(1).

[4] G. Banga, P. Druschel, and J. C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation, pages 45–58, New Orleans, LA, February 1999.

[5] G. Banga. Operating system support for server applications. PhD thesis, Rice University. May 1999.

[6] P. Druschel and G. Banga. Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems. In Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation, Seattle, WA, October 1996.

[7] C. A. Waldspurger and W. E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource

Management. In Proceedings of Symp. on Operating Systems Design and Implementation, Nov. 1994.

[8] Wensong Zhang. Linux Virtual Server for Scalable Network Services. Ottawa Linux Symposium 2000.

[9] Zeus load balancer. http://www.zeus.com/products/zlb/.

[10] ACME Labs. thttpd. http://www.acme.com/software/thttpd/.

[11] G. Banga and P. Druschel. Measuring the Capacity of a Web Server. In Proceedings of the 1997 USENIX Symp on Internet Technologies and Systems, Dec. 1997.

[12] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers. In Proceedings of the 2000 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems, pages 90–101, Santa Clara, CA, June 2000.

[13] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing Differentiated Quality-of-Service in Web Hosting Services. In Proceedings of Workshop on Internet Server Performance, June 1998.