웹 서버 클러스터에서 Cyber Foraging 응용을 위한 비주기적 부하 갱신을 통한 부하 분산 기법

XiaoYi Lu[†] · Zhen Fu[†] · 최 원 일[†] · 강 정 훈[††] · 옥 민 환[†††] · 박 명 순[††††]

요 약

본 논문에서 웹 서버 클러스터 환경에서 웹 요청들의 분산을 위한 부하 분산 기법을 제안한다. 전통적인 주기적 부하 정보 갱신 기반의 부하 분산 기법에서는 동기화된 부하 정보와 동적 페이지에 포함된 스크립트들의 갱신 정보 때문에 Cyber Foraging 서비스 같은 동적 웹 응용 프로그램에서는 적합하지 않다. 이를 해결하기 위해 Update-on-Finish 부하 분산 알고리즘은 비주기적인 부하 갱신 방법을 사용하고 있다. 웹 스위치는 비주기적인 부하 리포팅 후, 서버들의 실제 부하 정보를 알 수 있다. 그런후 실제 정보에 따라 부하 분산 스케줄을 재배열 한다. 하지만 Update-on-Finish 알고리즘의 경우 부하 정보를 유지하기 위한 통신 부하가 크다는 문제점을 가진다. 본 논문에서는 각 서버가 K%의 작업을 마친 후 비주기적 부하 정보 보고를 통하여부하감소시킨 방법을 제안한다. 또한 서버의 처리능력이 다른 환경을 고려하여 서로 다른 threshold Ti값을 적용함으로써, 다양한 처리 능력을 가진 서버들을 위한 로드 밸런싱 알고리즘으로 확장하여 제안하고 있다. 시뮬레이션 결과에서 제안된 K-Percent-Finish Reporting 방법은 Update-on-Finish 방법보다 최소 50% 이상의 통신 부하를 감소시키면서, 기존 주기적 부하 정보 갱신 기반의 관련 연구들보다 향상된 시스템 처리 능력을 보여주고 있다.

키워드 : 웹서버 클러스터, 부하 분산, 동적 웹페이지, 서버 능력

Request Distribution for Fairness with a Non-Periodic Load-Update Mechanism for Cyber Foraging Dynamic Applications in Web Server Cluster

XiaoYi Lu[†] · Zhen Fu[†] · Wonil Choi[†] · Jung Hun Kang[††] · MinHwan Ok[†††] · Myong-Soon Park[††††]

ABSTRACT

This paper introduces a load-balancing algorithm focusing on distributing web requests evenly into the web cluster servers. The load-balancing algorithms based on conventional periodic load-information update mechanism are not suitable for dynamic page applications, which are common in Cyber Foraging services, due to the problems caused by periodic synchronized load-information updating and the difficulties of work load estimation caused by embedded executing scripts of dynamic pages. Update-on-Finish algorithm solves this problem by using non-periodic load-update mechanism, and the web switch knows the servers' real load information only after their reporting and then distributes new loads according to the new load-information table, however it results in much communication overhead. Our proposed mechanism improve update-on-finish algorithm by using K-Percents-Finish mechanism and thus largely reduce the communication overhead. Furthermore, we consider the different capabilities of servers with a threshold Ti value and propose a load-balancing algorithm for servers with various capabilities. Simulation results show that the proposed K-Percents-Finish Reporting mechanism can at least reduce 50% communication overhead than update-on-finish approach while sustaining better load balancing performance than periodic mechanisms in related work.

Key Words : Web Server Cluster, Load Balancing, Dynamic Web Pages, Server Capability

1. Introduction

Cyber foraging [1] is a mechanism to augment the computational and storage capabilities of mobile devices by exploiting nearby compute and data staging servers. Cyber foraging uses opportunistically discovered servers in the environment to improve the performance of interactive applications and distributed file systems on mobile clients. And the nearby discovered servers, which are called surrogates in cyber foraging, work together to handle incoming request, and this working mechanism is

※ This work was supported by the Korea Research Foundation Grant and funded by the Korea Government (MOEHRD) (KRF2003-041-D00469).
† 준 회 원 : 고려대학교 대학원 컴퓨터학과 석사과정
†† 준 회 원 : 고려대학교 대학원 컴퓨터학과 석사학위취득
††† 정 회 원 : 한국철도기술연구원 선임연구원
†††† 정 회 원 : 고려대학교 컴퓨터학과 교수(교신저자)
논문접수 : 2006년 12월 28일, 심사완료 : 2007년 2월 1일

similar to web server cluster. In web clusters, all the web servers work with each other handling incoming requests, and there is usually a central web switch taking charge of distributing incoming requests among the web servers. The distribution of requests has a great affection over the performance of the web cluster system, and thus inducing an important research topic for web clusters: load-balancing among web servers, the purpose of which is to balance the load among servers and avoid any server over-loaded, and thus increases the system stability and improves the communication between different servers in the system. Similarly, load balancing for surrogates in cyber foraging is important too, and we can arrange all the surrogates in a server cluster manner, in which one surrogate works as web switch and distribute requests to other surrogates. In this way, we can use the same load balancing mechanism of web server clusters to distribute request fairly in cyber foraging.

There have been many researches on load-balancing in request distribution for web cluster [3], [4], [5]; however most of them only focus on static web pages rather than dynamic web pages which are embedded with executing scripts. Dynamic web Scripting applications are increasingly popular in recent years and they are common used in cyber Foraging, e.g. remote execution [1]. Generally, it is very difficult to assess the workload of the servers running dynamic applications. The reason is the scripts execution is usually embedded inside the web pages. Thus, the difficulty makes it complicated to design the workload assessment algorithm for network switch which has a requirement of the high efficiency. Even if the network switch's capability can achieve the requirement, most of the load-balancing algorithms still have an inherent problem on scalability in the load-information exchanging mechanism.

In the next section, some recent related works are introduced, and our non-periodic load-update mechanism for load-balancing is described in section 3. In the first part of section 3, we assume all the servers have the same capability, while in the second part of section 3, we further extent our idea by considering the different capabilities of web servers. Performance evaluation and comparison with related works is shown in Section 4, and we will conclude our work in Section 5.

2. Related Works

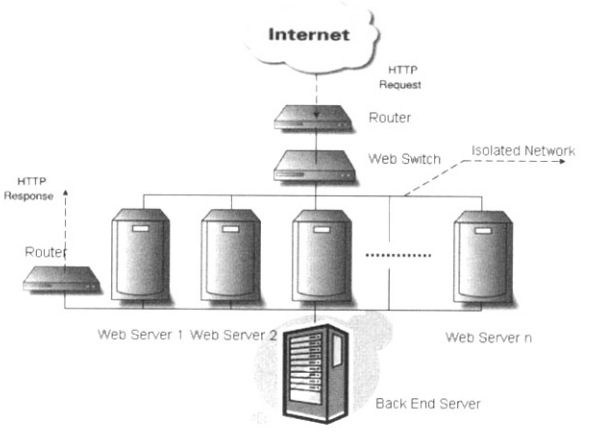
Many load-balancing mechanisms have been proposed which can be mainly categorized into two kinds, static and dynamic load-balancing. Static load-balancing does not consider current load on web servers while dynamic load-balancing mechanism cares about the server's current load before distributing requests. Random and Round-Robin are two representative algorithms for static load-balancing, and their purpose is to reduce the burst in the request arrival stream to every single web server [5], [6]. In a random allocation, the requests are assigned to any server picked randomly among web server clusters. In a Round-Robin algorithm, the first request is allocated to a server picked randomly from all the web servers, and for the subsequent requests, the web switch follows the circular order to redirect the request. Once a server is assigned a request, the server is moved to the end of the list. This keeps the servers equally assigned.

Weighted Round-Robin (WRR) improves Round-Robin by considering the current load of web server [2]. WRR uses periodic load information update. The web switch collects each server's load information periodically, and after it realizes each server's load, it sends requests to a less loaded server with a higher rate while sends requests to a more loaded server with comparative lower rate until each server reach equal loads before the next load-information updates. With the knowledge of dynamically changing load information of web servers, WRR can get better performance than Round-Robin.

Dahlin's algorithm [8] outperforms WRR by exploiting the advanced update mechanism of load-information [9]. Dahlin's algorithm also uses periodic load-information update. The web switch realizes the differences in loads between servers by load-information update. It sends requests to servers with least loads. After this duration, all the server's loads are equalized to the most loaded server.

Both WRR and Dahlin's algorithm require knowing the current load of web Server to distribute the requests. However nowadays, there are dynamic web pages created in Java, PHP, ASP and so on, thus it is difficult to estimate the load of these web pages, even with the information from the application layer, due to executing scripts embedded in the pages. Although such an estimation algorithm might exist, the web switch cannot use a highly sophisticated algorithm in distributing requests, since it has to take immediate decisions for hundreds of thousands of requests per second. For the weighing dynamic web pages is far different from that of static web pages, it is not an appropriate respect in applying WRR or Dahlin.

Moreover, both WRR and Dahlin's algorithm periodically updates web server's load information, which means the web switch should check out the load information of



(Fig. 1) Web server cluster with isolated system network

all the web servers simultaneously and periodically. From Figure 1 we can see that requests are distributed from the web switch through the isolated network. The web switch should update the load-information of all web servers in prior in order to distribute the requests evenly. Conventionally, all the servers report their load-information to the web switch at any given rate and the load-information of all the servers should be updated synchronized. If the load-information updates are not synchronized in this kind of periodic load-balancing mechanism, the load-information of some servers should be obsolete, and thus the load-balancing algorithm will not work, synchronicity is strongly required here. At the same time, due to synchronized reporting, a number of report packets are concentrated at the web switch from all the web servers and the web switch requires a term to receive all the report packets. If there are many servers, the number of report packets would be large and the term is not neglectable, and for the web switch can not distribute any requests during this term, the load-balancing mechanism can not get expected performance. What is more, in the traditional periodic load information update mechanism, the decision of reporting time interval is only is depending on a tradeoff between load-balancing performance and communication overhead, and can not reflect any current working status of servers, which means no matter how many requests are queued on a server and whether the server has the ability to process more requests, it send a report to the switch at any predefined time interval. Non-periodical load information update mechanism improved the idea of periodical reporting by allowing all the servers not reporting simultaneously and the reporting time of each server is depending on its own working status, e.g. Update-on-Finish[9] , in which each server reports when the number of executing requests decreases that n more requests are needed.

Update-on-Finish [9] reporting is based on non-periodic load information update, and since each server reports when one of its queuing requests is finished, all the report packets are not concentrated at any instant, and thus the bottleneck at the web switch caused by periodical update can be avoided. However since in [9], whenever a server finish executing a request, it should send a report to the web switch, the communication over head between servers and web switch is high. In this paper, we adopt non-periodic updating mechanism, and in the first part of the proposed mechanism, we suppose all the servers having the same capability and each server sends a report to web switch when K percents of its queuing requests have been finished. By properly setting the K value, the communication overhead can be largely reduced comparing with Update-on-Finish [9] mechanism. Furthermore, since servers are usually different from each other in their performance since they may have different CPU capabilities, different memories, and different stability of network connection status etc. Especially for cyber foraging, since the communication between mobile clients and a surrogates is via short-range wireless peer-to-peer technology [10], the different wireless connectivity statuses strongly affect the overall performance. So in this paper, we further consider the different capabilities of servers in the second part of proposed mechanism to decide reporting time.

In the following part, our proposed load-balancing mechanism for cyber foraging which is based on non-periodic load information update is described in detail.

3. Non-Periodic Load-Update Algorithm for Load-Balancing

Depending on whether consider the requests content type, web switches are broadly classified into Layber-4 and Layer-7 web switches [11]. Layer-7 load balancing, also known as application-level load balancing, is to parse requests in application layer and distribute requests to servers based on different types of request contents. However, because many servers may have same contents, we can apply load-balancing algorithms to those servers after applying the contents-based algorithms first. Our purpose in this paper is to get better service by selecting proper servers among the servers which have same contents. Therefore we assume that the all servers have same contents.

Also, we mainly focus on the processing of dynamic

web pages. Since the executing scripts of dynamic web pages require additional computing cost and execution time, and the execution time of scripts is usually different among dynamic web pages, we call this execution time Execution Length, which is not deterministic and hard to know before finishing execution. So it is obvious that execution length is not suitable for evaluating load-balancing. In the first part of the proposed idea, we suggest the run-queue length best describes a server's load which is the same as the previous work [7] and use it as the performance evaluation metric. And in the extended idea part, where we further consider the different capabilities of servers, we use the occupation of a server's resources as the evaluation metric, and the goal of load balancing under this metric is to let all the servers contribute the same percentage of their abilities into requests execution.

3.1 K-Percents-Finish Reporting Mechanism

In this section, we are going to describe our first proposed load-balancing mechanism, in which each server reports to web switch when K percents of their current loads finished, and K here is a threshold value to decide reporting occasion. There are some assumptions supporting this idea:

① In this part, there are enough incoming requests, which means the arrival rate of requests is big enough that before a web server finishes a request, a new request will be distributed to it, thus the situation of server halt without any request will not happen and load-balancing is necessary at anytime.
② After the web switch compensates each web server, no reporting from each server will happen until a server finishes K percent of its current requests.
③ Using run-queue length describes a server's load
④ All the web servers have the same processing capability, as well as the same network connection quality.

Based on these assumptions, our load-balancing algorithm with the same capability web servers can be described as following steps:

① At the initial step, the web switch simply distributes the incoming requests to all the servers in "Round-Robin" manner. And the data structure design of distribution list here is the same as Round-Robin. As a result, equal numbers of requests are executed in the servers.
② Each server has a table recording how many requests have been finished. When a web server finishes executing a request, it adds one to the finish-request table and checks its loads whether the Number of finished requests/Number of current load is bigger than k, which indicates K percentage of current loads has been finished. If so, it sends a report to the web switch and sets the finish-request table to zero, otherwise, it continues executing queued requests.
③ If new requests come to web switch when no report from any web server is made, the switch would continually use "Round Robin" to distribute requests among web servers.
④ When the web switch receives a report from a server, it grants a priority to the server according to the order of arriving reports, the server which sends the report firstly gets the highest priority, and the servers which have not reported yet have lowest priority.
⑤ New incoming requests are distributed to servers like this: the server with the highest priority will be put to the head of distribution list and be the first one to get the new distributed requests, and the lower the priority is, the later the server gets distributed requests.
⑥ Procedure continues from the second step.

In our proposed mechanism, each web server makes no report until K percent of their current load is done. And the operation of web switch is quite easy for it does not need to know load information before server reports and only need to re-arrange "Round-Robin" schedule according to reported information. In the simulation part, we will show how different K values affect the system performance. By properly setting up the K value, our proposed mechanism can well outperform previous approaches.

3.2 Load Balancing Considering Different Server Capabilities

In section 3.1, we suppose that all the web servers have the same processing capability and no difference between them when handling incoming requests. But in real use, we can not expect all the servers have exactly the same processing capability, servers are usually different from each other in terms of different storages, CPU capabilities, and different amount of RAM and ROM, and all these appended with the different stability of network connection status make the performance of servers vary a lot.

In this section, our proposed algorithm is based on the common accepted truth that with some optimization techniques, performance gain can be achieved by sending disproportionately high fraction of workload to the server of higher capability.

If we consider the different capabilities of web servers, only using run-queue length as the evaluation metric is

not suitable. Servers which have better processing capability can finish executing requests faster, thus if we simply grant all the web servers with the same number of requests to balance the load, the more capable servers can finish their tasks early and have halt time.

In this section, we use the usage percentage of a web server's resources as the performance evaluation metric, and the load balance goal here is to let all the web servers use the same percentage of their resources, which means all the web servers should contribute the same percentage of their capabilities into request execution.

First, the parameters used are listed as below:

- n: the number of web servers
- i: $1 \le i \le n$, indicate the serial number of web servers
- w_i: the weight of web server i when the web switch distribute requests
- l_d^i: the current real load of server i (not only consider run-queue length but also consider the occupation of a server's capability)
- r_j: the using percentage of resource j
- a_j^i: the weight of resource j of server i
- u_i: the resource using status of server i
- R_i: the run-queue length of server i
- r_{cpu}^i: the using percentage of CPU of server i
- r_{mem}^i: the using percentage of memory of server i
- r_{net}^i: the using percentage of network resources of server i

In this part, different servers have different weight values which can indicate their processing capability. These weight values are obtained from history requests executing experience, and servers with better performance history have higher w_i value. When the web switch distributes requests, the web servers with higher weight value w_i will be given more requests, and each web server can be indicated as a vector $s_i=\{ i, w_i \}$.

Further more, we use the equation (1) to indicate the resource using status of server i, in which $\sum_{j=1} a_i = 1$

$$u_i = \sum_{j=1} a_j r_j \tag{1}$$

For different using purpose, different kinds of resources will be the most important factors affecting the performance of web server, but among them, the most common factors are the usage of CPU, memory and network resources. So we can simply use equation (2) to describe the resource using status of each web server:

$$u_i = \alpha r_{cpu}^i + \beta r_{mem}^i + \gamma r_{net}^i \tag{2}$$

In equation (2), α , β and γ stands for the weighting value of CPU, memory, and network resources of each server separately. And these values are decided according to different kinds of requests, for some complicated computation requests, α requires heavier weight because CPU is the most important factor to execution while for some I/O requests, network status is more important and thus γ requires bigger weight value. But in our paper, we do not consider the request type, so we simply use α=0.25, β =0.25 and γ=0.5 as weight value.

In this part, the decision of request distribution depends on the real load l_d^i of each server, which considers not only the server's run-queue length R_i but also its resource using status u_i and its weight value w_i. We use equation (3) here to denote l_d^i, and the incoming request is firstly sent to the server with the minimum l_d^i value: $i = \min(l_d^i)$.

$$l_d^i = \frac{u_i R_i}{w_i} \tag{3}$$

The report mechanism here is similar as the first part: we set up a threshold value T_i for deciding the report time.

At the initial step, the web switch still simply distributes incoming requests to all the servers in "Round-Robin". After this step, each server has the same number of requests waiting to be executed. Whenever a server finishes executing a request, it checks out its l_d^i value, if l_d^i is still above the threshold value T_i, the server i continues executing queued requests without any other operation, else if l_d^i goes below the threshold value T_i, the server i sends a report to the web switch. When the web switch receives a report, it records the server's vector $s_i=\{ i, w_i \}$ and its l_d^i value, and puts recorded value into its load information table. And here the web server with the minimum l_d^i value, which $i = \min(l_d^i)$ will be granted with the highest priority, and among the reported servers, the bigger the l_d^i value is, the lower priority it will be granted, and the servers that have not report yet still have the lowest priority. And the following step is the same as step 5 and 6 in section 3.1.

This report on threshold value algorithm is used to decide report time according to each server's real load l_d^i. In this way, when a web server finds its real load l_d^i goes below the threshold value T_i, which means it finishes several requests and have spare resources and capability to handle more requests, it reports to the web switch, and new requests will be sent to it, and the more resources it spares, the sooner it gets new incoming

requests. As a result, no server will be overloaded or idle, and the contribution percentage of servers' capabilities is the same after load balancing.

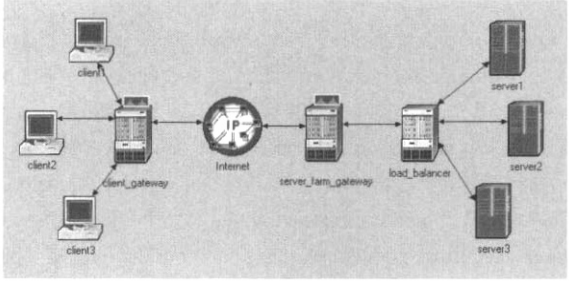
The proper value of T_i is decided through real simulation, and to simplify the discussion, we suppose all the web servers adopt the same T_i value.

4. Performance Evaluation

The main goal of the experimentation described here is to evaluate the performance of the proposed algorithm and compare it with other related works. We will firstly evaluate its load balancing performance in evenly request distribution, and then see its affection to communication overhead.

4.1 Experimental Methodology

Figure 2 is the topology we used in the experiments. The web server cluster contains one gateway, one load balancer and 3 web servers. Detailed descriptions of the nodes in the testing bed are showed in the Table 1.



(Fig. 2) Network Topology

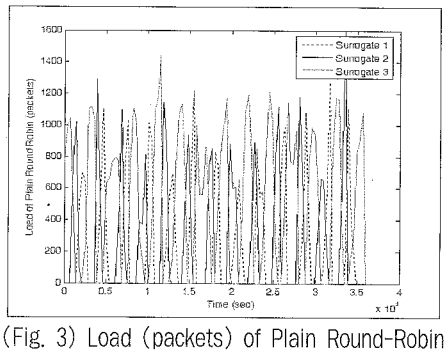
4.2 Result and Analysis

In the simulation, to evaluate the performance under heavy workload, the client gateway was set to send heavy HTTP requests to the load balancer to distribute them to web servers.

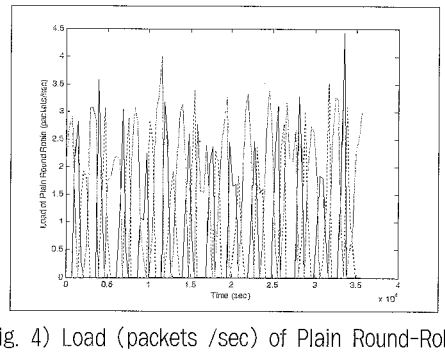
4.2.1 Evaluation of plain Round Robin and Dahlin

All of the evaluations lasted for about 10 hours. And the Average traffics under Round-Robin algorithm that the nodes received are as showed in table 2 and table 3:

Figure 3 and Figure 4 states the statistics under plain round robin, whereas Figure 5 and Figure 6 show the result under dahlin:



(Fig. 3) Load (packets) of Plain Round-Robin



(Fig. 4) Load (packets /sec) of Plain Round-Robin

〈Table 1〉 Network nodes description

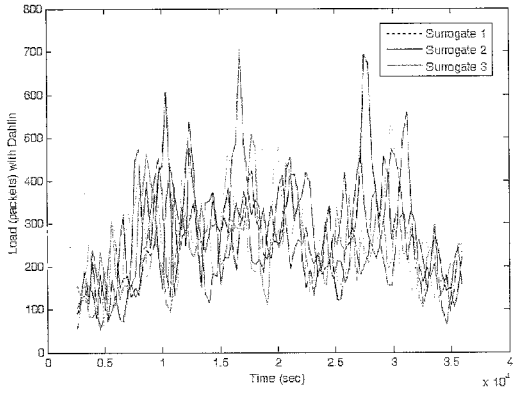
Server	Supported Protocols: UDP, IP, Ethernet, RIP, TCP, OSPF Port Interface Description:Ethernet connection at 10 Mbps, 100 Mbps, or 1000 Mbps
Load Balancer	Device Name: load_balancer_e16 The load_balancer_e16 models a device which intercepts internet application traffic and sends it to one of a list of preconfigured servers
Server_fam_gateway	Vendor: Cisco Systems Product: CISCO4000 Device Class: Router

〈Table 2〉 Traffic of load balancer under plain Round-Robin

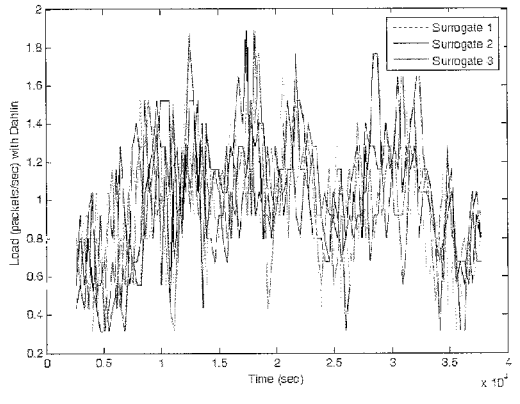
Node	Load Balancer Traffic Received (bytes/sec)	Load Balancer Traffic Received (packets/sec)	Load Balancer Traffic Received (packets/sec)	Load Balancer Traffic Sent (packets/sec)
Load balancer	1,693/815/782	1.39/0.67/0.64	151/73/70	1.06/0.51/0.49

〈Table 3〉 Traffic of web servers under plain Round-Robin.

Node	Ethernet Load (bits)	Ethernet Load (bits/sec)	Ethernet Traffic Received (bits)	Ethernet Traffic Received (bits/sec)	Ethernet Traffic Received (packets)	Ethernet Traffic Received (packets/sec)
server1	6,253,930	6,254	543,090	558	477	0.79
server2	6,706,146	6,520	582,431	582	512	0.77
server3	7,387,458	13,544	658,152	773	579	1.06



(Fig 5) Load (packets) of Dahlin



(Fig. 6) Load (packets /sec) of Dahlin

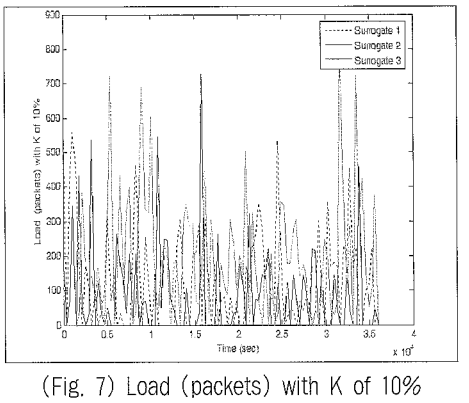
4.2.2 Evaluation of K-Percents-Finish Reporting Mechanism with K value of 10%

The Average traffics under proposed algorithm of the nodes received are showed in table 4 and table5:

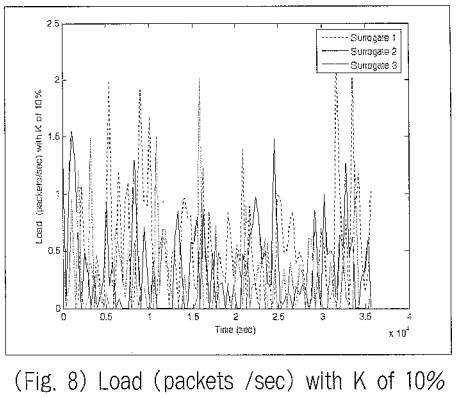
The proposed algorithm was simulated with a strategy of on-demand load-information update. That is, only when the current load achieves the preset threshold, the load information will be updated. As showed in the traffic information tables, plain Round-Robin sent 6,253,930 bits data to the server 1 and 6,706,146 bits data to server 2 and 7,387,458 bits data to server 3. Under the same network settings, proposed algorithm sent 3,290,061 bits data to server 1 and 3,257,203 bits data to server 2 and 3,155,138 data to server 3. It is obvious that incoming requests are distributed more evenly by proposed algorithm than plain Round-Robin algorithm. We will analyze the result more detailedly.

Figure 7 and 8 state the statistics under proposed algorithm with the K threshold value of 10%.

By analyzing this evaluation result with that of plain round robin, we can see that the packets transmitted to the server under the proposed algorithm are more evenly than plain Round-Robin algorithm, and thus outperform plain round robin approach.



(Fig. 7) Load (packets) with K of 10%



(Fig. 8) Load (packets /sec) with K of 10%

〈Table 4〉 Traffic of load balancer with K value of 10%

.Node	Load Balancer Traffic Received (bytes/sec)	Load Balancer Traffic Received (packets/sec)	Load Balancer Traffic Received (packets/sec)	Load Balancer Traffic Sent (packets/sec)
Load balancer	1,142/1,131/1,096	0.937/0.928/0.899	101/101/97	0.715/0.709/0.686

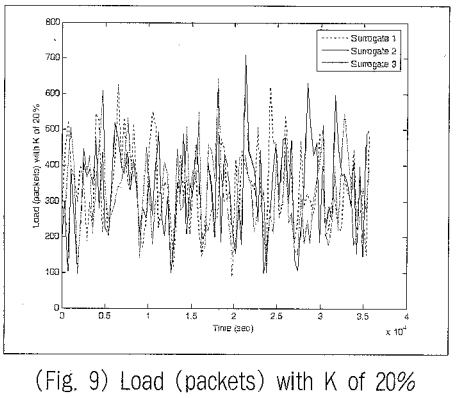
〈Table 5〉 Traffic of web servers with K value of 10%.

Node	Ethernet Load (bits)	Ethernet Load (bits/sec)	Ethernet Traffic Received (bits)	Ethernet Traffic Received (bits/sec)	Ethernet Traffic Received (packets)	Ethernet Traffic Received (packets/sec)
server1	3,290,061	9,139	292,112	811	258	0.716
server2	3,257,203	9,048	290,224	806	255	0.709
server3	3,155,138	8,764	280,640	780	247	0.687

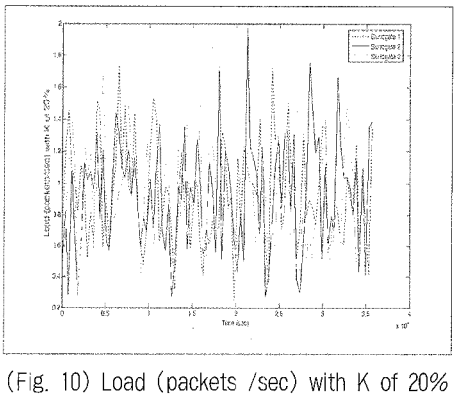
4.2.3 Evaluation of K-Percents-Finish Reporting Mechanism with K value of 20%

We also evaluate the algorithm performance with the K threshold value of 20%. The traffic under the proposed algorithm with 20% K value is showed as the table 9 and 10. We can see K of 20% can achieve better performance than 10%.

We also use other K values ranging from 10%~40%, but find the load balancing performance does not change proportionally with K value. And we find that the proper K value also depends on different characteristics of each system. So it is hard for us to say which K value is the best, but we recommend a range between 20%~30% in which can achieve better performance comparatively.



(Fig. 9) Load (packets) with K of 20%



(Fig. 10) Load (packets /sec) with K of 20%

4.2.4 Report on Threshold Value Algorithm

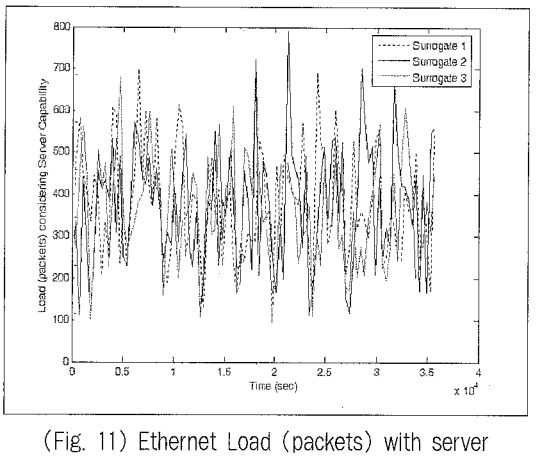
The parameters of the evaluation of the server capability concerned algorithm are set as the table 8 shows.

Here we suppose all the servers report when 50 percents of their resource is available. And we adopt W=2 and R=10 for setting threshold value Ti, thus Ti=50%* 10/2=2.5

Figure 11 and 12 state the statistics under server capability concerned algorithm. Compared with the K percent reporting mechanism, report on threshold value algorithm can balance the server load in the web cluster more efficiently by assigning comparatively more requests to lighter loaded servers. The system performance changes with different Ti values. In the future work we will focus on how to find the proper Ti value.

〈Table 8〉 Parameter Setting.

Parameter	Value
u	50%
R	10
W	2



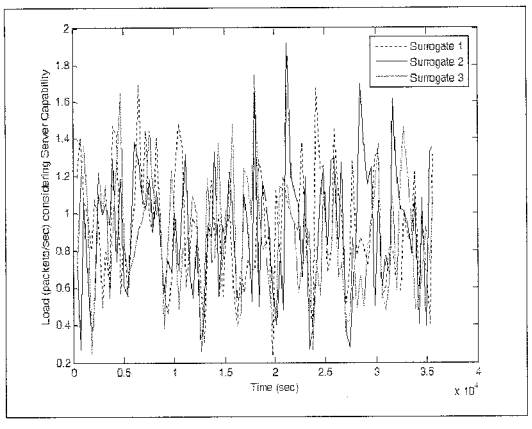
(Fig. 11) Ethernet Load (packets) with server capability concerned

〈Table 6〉 Traffic of load balancer with K value of 20%.

Node	Load Balancer Traffic Received (bytes/sec)	Load Balancer Traffic Received (packets/sec)	Load Balancer Traffic Received (packets/sec)	Load Balancer Traffic Sent (packets/sec)
Load balancer	498/497/497	0.408/0.408/0.408	44.3/44.2/44.1	0.312/0.312/0.311

〈Table 7〉 Traffic of web servers with K value of 20%.

Node	Ethernet Load (bits)	Ethernet Load (bits/sec)	Ethernet Traffic Received (bits)	Ethernet Traffic Received (bits/sec)	Ethernet Traffic Received (packets)	Ethernet Traffic Received (packets/sec)
server1	1,857,581	3,973	165,626	354	346	0.812
server2	1,966,380	3,987	174,407	354	354	0.812
server3	1,962,633	3,980	173,982	353	353	0.811

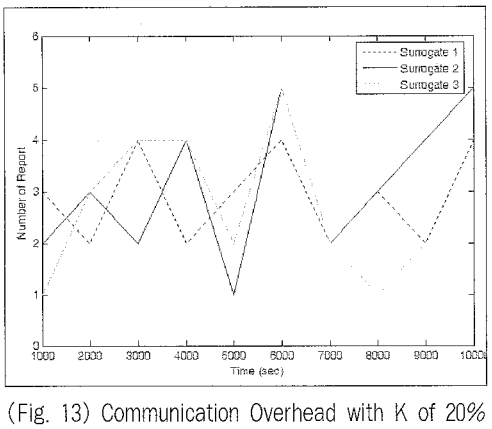


(Fig. 12) Ethernet Load (packets /sec) with server capability concerned

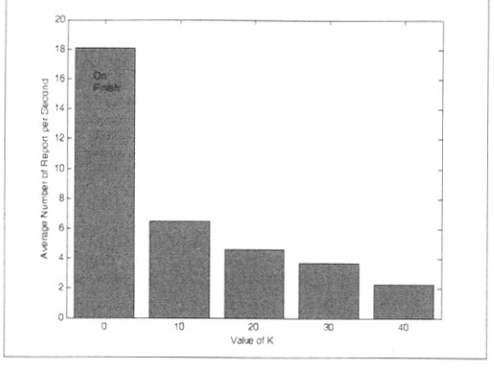
4.2.5 Load-Update Overhead

From the evaluation from section 4.2.1 to section 4.2.4 we can see our proposed algorithm can sustain good load balancing performance in evenly loads distribution. In this part, we will see how our proposed mechanism improves update-on-finish approach by alleviating communication overhead.

Figure 13 shows the overhead communication during 1000 milliseconds under the proposed algorithm when K is 20%. Here as it described in the former sections, the overhead communication means the non-periodical load-update information reporting. As the figure11 shows, it states that, as the requests are distributed to the web servers evenly, the communication overhead of each web server to report the load-update information to the load balancer is similar with each other. And the communication overhead only takes 0.5% of the traffic in average.



(Fig. 13) Communication Overhead with K of 20%



(Fig. 14) Communication Overhead with different K values

We have also compared load reporting overhead under different K values. Figure 14 shows the load report overhead in our simulation. This presents the change of number of reports during 40000 seconds with the four K values as well as update-on-finish approach, in which K is regarded as 0. As the figure shows, the average number of reports per second reduces as the K value increases. This is because with small K value, all the servers report to web switch frequently, and thus results in much communication overhead.

5. Conclusions and future work

Load-balancing algorithms founded on conventional periodic load-information update are not suitable for dynamic applications, which are common in Cyber Foraging, because of the difficulty to access the workload of dynamic pages before finishing real execution. The proposed algorithm is based on non-periodic load-updating mechanism and since the web switch only need to re-arrange the schedule of "Round-Robin" according to the reports from each server, it is easy to operate and thus can avoid the bottleneck at web switch and get higher throughput. With the proper K value which decides the reporting time, it also can alleviate communication overhead caused by frequently reporting in update-on-finish mechanism. By further considering the different capabilities of servers, our proposed algorithm is more reasonable for real use, and all the servers contribute their capabilities and resources evenly after load-balancing. Simulation results have shown that at least 50% communication overhead can be reduced compared to update-on-finish approach while good performance in terms of more evenly requests distribution than the periodic mechanisms in related works can be sustained. To find the proper threshold value, we would consider communication cost, CPU and memory capacity of every server, but it would be very difficult research task. Therefore finding proper threshold values will be left as future works.

References

[1] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang, "The Case for Cyber Foraging", Proc. of the 10th workshop on ACM SIGOPS European workshop, pp.87-92, September, 2002.

[2] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu, "The state of the art in locally distributed web-server systems," ACM Comput. Surv., Vol.34, No.2, pp.263 - 311, June, 2002.

[3] M. Andreolini, M. Colajanni, and R. Morselli, "Performance study of dispatching algorithms in multi-tier web architectures," ACM SIGMETRICS Perf. Eval. Review, Vol.30, No.2, pp.10 - 20, Sept., 2002.

[4] E. Casalicchio, V. Cardellini, and M. Colajanni, "Client-aware dispatching algorithms for cluster-based web servers," Cluster Comp., Vol.5, No.1, pp.65 - 74, Jan., 2002.

[5] X. Tang and S.T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," Proc. Conf. on Para. Proc., pp.373 - 382, 2000.

[6] Server Load Balancing: Algorithms, Published: Monday, May 17, 2004 http://content.websitegear.com/article/load_balance_types.htm

[7] T. Kunz, "The influence of different work-load descriptions on a heuristic load balancing scheme," IEEE Trans. Softw. Eng., Vol.17, No.7, pp.725 - 730, July, 1991.

[8] M. Dahlin, "Interpreting stale load information," IEEE Trans. Parallel Distrib. Syst., Vol.11, No.10, pp.1033 - 1047, Oct., 2000.

[9] M. Ok and M.-s. Park, "Request distribution for fairness with a new load-update mechanism in web server cluster," LNCS, Vol.3222, pp.221 - 229, Oct., 2004.

[10] Sachin Goyal and John Carter, "A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices". Appears in Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA2004), pp.186-195.

[11] T. Schroeder, S. Goddard, and B. Ramamurthy, "Scalable web server clustering technologies," IEEE Netw., Vol.14, No.3, pp.38 - 45, May/June, 2000.

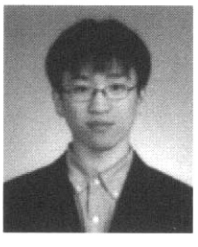
Xiaoyi Lu

e-mail : felicity_lu@ilab.korea.ac.kr
2004년 Department of Computer Science & Technology, Beijing Institute of Technology(학사)
2005년~현재 고려대학교 컴퓨터학과 석사과정
관심분야 : ubiquitous computing, embedded system, Ad-Hoc network, wireless sensor network

Zhen Fu

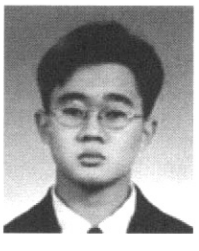
e-mail : fuzhen@ilab.korea.ac.kr
2004년 Department of Computer Science, Wuhan University(학사)
2005년~현재 고려대학교 컴퓨터학과 석사과정
관심분야 : ubiquitous computing, embedded system, Ad-Hoc network, wireless sensor network

최 원 일



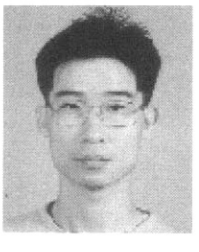
e-mail : wonil22@ilab.korea.ac.kr
2005년 고려대학교 컴퓨터학과(학사)
2005년~현재 고려대학교 컴퓨터학과 석사과정
관심분야 : ubiquitous computing, embedded system, RFID, wireless sensor network

강 정 훈



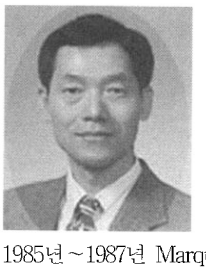
e-mail : kjhoun@ilab.korea.ac.kr
2005년 고려대학교 전산학과(학사)
2007년 고려대학교 컴퓨터학과(석사)
관심분야 : ubiquitous computing, embedded system, Ad-Hoc network, wireless sensor network

옥 민 환



e-mail : panflute00@paran.com
1996년 부산대학교 전산과(학사)
1998년 부산대학교 전산과(석사)
1999년~현재 고려대학교 컴퓨터학과 박사과정
2001년~2003년 창신대학 정보통신과 조교수
2004년~현재 한국철도기술연구원 선임연구원
관심분야 : parallel processing, clustering, storage server, load-balancing in locally distributed network.

박 명 순



e-mail : myongsp@ilab.korea.ac.kr
1975년 서울대학교 전자공학과(학사)
1982년 University of Utah, 공과대학 전기공학과(석사)
1985년 University of Iowa, 공과대학 전기 및 컴퓨터공학과(공학박사)
1975년~1980년 국방과학연구소 연구원
1985년~1987년 Marquette University 전기 및 전산과학과 조교수
1987년~1988년 포항공과대학교 전자전기공학과 및 전산과학과 조교수
1988년~현재 고려대학교 컴퓨터학과 교수
관심분야 : parallel processing, clustering, Internet computing, networks on storage, embedded systems, mobile IP and wireless communications.