第14部

WWWキャッシュ技術

第１章　WWWA ワーキンググループ

WWWA (World-Wide Web Architecture) ワーキンググループは、WWW システムのプロトコル自体の改良や、キャッシュ以外のシステムを含めて、WWW アーキテクチャ全体を見直すことを目標とした研究グループである。

WWW が普及する過程においては、インターネットにおける WWW トラフィックの軽減技術として WWW キャッシュシステムが有効であると考えられ、大規模な分散 WWW キャッシュシステムの構築が試みられてきた。WIDE プロジェクトでも、WIDE CacheBone という大規模分散 WWW キャッシュシステムの構築と運用を W4C (WIDE WWW Cache) ワーキンググループで行なってきた。しかし、既存の WWW キャッシュシステムでは、WWW におけるアクセスやコンテンツの特性からキャッシュによる大きなトラフィックの削減は期待できないことが W4C ワーキンググループによって報告されている。また、インターネットのユーザ層が一般に広がるに従い、トラフィックの削減以外の要求が WWW キャッシュシステムに求められるようになっている。

WWWA ワーキンググループはこのような要求に答えるために、WWW キャッシュシステムを含めた WWW アーキテクチャ全体を対象にして研究活動を行なっている。

今回の報告書では、以下の内容に関する報告を行なう。

- キャッシュオブジェクトの複製技術を用いた、分散 WWW キャッシュシステムのためのロバストなハッシュルーティングアルゴリズム
- パケットモニタによる WWW サーバの状態の解析、推定

第２章　Duplicated Hash Routing

2.1 Duplicated Hash Routing: A Robust Algorithm for a Distributed WWW Cache System

Hash routing is an algorithm for a distributed WWW caching system that achieves a high hit rate by preventing overlaps of objects between caches. However, one of the drawbacks of hash routing is its lack of robustness against failure. Because WWW becomes a vital service on the Internet, the capabilities of fault tolerance of systems that provide the WWW service come to be important. In this paper, we propose a *duplicated hash routing* algorithm, an extension of hash routing. Our algorithm introduces minimum redundancy to keep system performance when some caching nodes are crashed. In addition, we optionally allow each node to cache objects requested by its local clients (*local caching*), which may waste cache capacity of the system but it can cut down the network traffic between caching nodes. We evaluate various aspects of the system performance such as hit rates, error rates and network traffic by simulations and compare them with those of other algorithms. The results show that our algorithm achieves both high fault tolerance and high performance with low system overhead.

2.2 Introduction

Increase of network traffic on the Internet caused by WWW makes its service delay fairly large. Many researchers have been working on this problem and various solutions are proposed. Caching is one of the solutions that reduce the network traffic on the Internet and its service delay. In these days, distributed WWW caching systems, where caching nodes share their cached objects with each other, are being employed among many sites. The distributed WWW caching systems are put into two categories from a viewpoint of a role of each caching node: *loosely* coupled systems and *tightly* coupled systems.

In a loosely coupled system, each caching node caches all the objects requested by its *local* clients. A local client of a caching node means a client that sends requests directly to the node. If the caching node which receives a request does not have the requested object in its local cache, it tries to retrieve the object from the other nodes by some method. Internet cache protocol (ICP) [146] is one of the protocols used in this kind of systems.

In a tightly coupled system, each caching node is assigned a portion of name space of WWW objects and caches only the objects whose names are in the assigned space. All the requests for a WWW object are forwarded to a single caching node and cached there. Because these systems optimize their whole storage capacity, they can achieve higher hit rates than loosely coupled systems. There are several systems put into this category and some of them are implemented [147, 148, 149, 150].

There are two major disadvantages in a tightly coupled system. One is its poor scalability. A tightly coupled system cannot be employed in an environment with low bandwidth between caching nodes. Advances in backbone network technology will make it possible to operate the tightly coupled caching system over a distributed area in the near future. However, even if such a desirable situation comes true, a kind of localization of communication, which reduces network traffic between caching nodes, is desirable.

The other disadvantage is its lack of robustness. Because each caching node has to handle requests from all clients in the system, failure of a single node inevitably has influence on all the clients. This drawback can be removed by introducing redundancy to the system.

In this paper, we propose a *duplicated hash routing* algorithm, an extension to the simple hash routing [150]. Hash routing is a simple and efficient algorithm for tightly coupled caching systems. By introducing cache redundancy to the system, our algorithm can keep its high hit rate even when some caching nodes are in failure. The redundancy of our system is moderate, i.e., decrease of the hit rate and increase of the network traffic between caching nodes is small. In addition, we optionally enable *local caching* at each caching node to cut down much of the network traffic.

Section 2.3describes ICP and hash routing, typical algorithms for distributed WWW caching systems. In section 2.4, we introduce and give details of the duplicated hash routing algorithm. We evaluate its performance and compare it with that of other algorithms. We describe simulation models of these systems in section 2.5, and examine the relationship among various parameters such as the number of caching nodes, the disk size of each node, frequency of duplication, network traffic, and hit rates in section 2.6.Finally, we conclude this paper with a summary of our results and a discussion of some open issues.

2.3 Background

A variety of technologies for distributed caching systems are being developed. We describe some of them within the scope of our study.

2.3.1 ICP

ICP is implemented in many caching systems as a protocol based on queries to neighbor caching nodes. When a caching node receives a request from its local client and does not have the requested object in its local cache, the caching node sends ICP queries to all the neighbor proxies. If one of the neighbors has the object, the caching node that sent the ICP query retrieves the object from the neighbor node and put it in its local cache. After the object is copied in the local cache, requests for the object from any local clients of the caching node can make local hits.

One of the major drawbacks of ICP is its high communication cost of queries [150, 151, 152, 153].

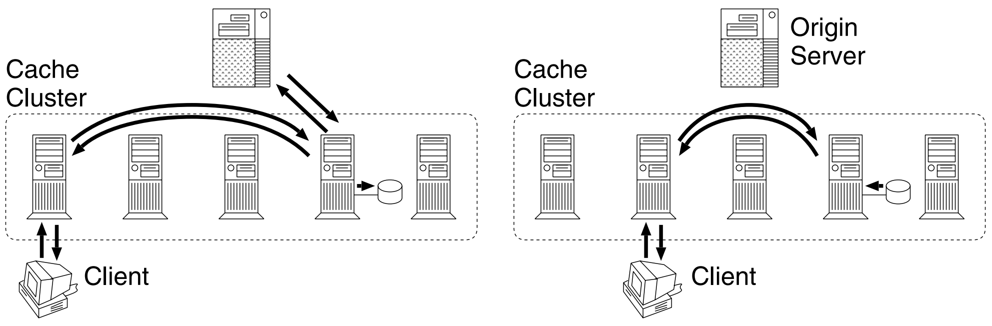

図 2.1 Simple hash routing (left: cache miss, right: cache hit)

Caching systems such as Crisp cache [154] and Summary cache [155] avoid queries per local cache miss, i.e., they exchange a list of cached objects periodically in a highly efficient way. However, despite its lack of scalability, ICP is still used widely. We adopt ICP as a reference algorithm to compare with other ones.

2.3.2 Hash Routing

Hash routing requires neither a query to a neighbor nor an exchange of an object list. Figure 2.1 depicts a basic model of hash routing, called a simple hash routing. A client sends an HTTP request to its local caching node. All the caching nodes in a system share a single hash function and are assigned a part of name space of WWW objects without overlaps by the hash function. Thus the local caching node calculates a hash value of the request and forwards it to some node according to the hash value. After the object is cached at one of the caching nodes, requests for the object from any clients in the system make cache hits.

Because an object is cached at only one caching node, hash routing can achieve a higher hit rate. On the other hand, it has two drawbacks as we mentioned in section 2.2.One is its large network traffic between caching nodes. Because a client cannot retrieve objects directly from the cache of its local caching node in most cases, a large number of object transmissions between caching nodes occur in the hash routing system. Therefore hash routing requires the situations that all the caching nodes are within short distance and connected to each other by networks with high bandwidth. The other drawback is its lack of fault tolerance. Because an object is only cached at one node, all clients can suffer from errors or cache misses even when just a single caching node in the system gets failure.

2.3.3 Robust Hash Routing

There are many possible points of failure in a distributed WWW caching system. In our study, we focus on failures at caching nodes. We do not discuss other points such as networks or origin servers because a failure on a network can be counted in a failure of a caching node and a failure at an origin server has little impact on the total availability of service compared with that at a proxy server.

When a failure occurs at a caching node of a hash routing system, rebuilding its hash function is a straightforward way to recover. However, there is a weak point that the fraction of objects no longer in correct caches can be large [156]. This leads to performance degradation of the whole system during and after failure of a caching node. To overcome such weakness, *robust hashing* was proposed [150] and actually implemented [149].

With robust hashing, the URL of a requested object and the name of each sibling cache together are used to generate a hash value or score; the object is then mapped to the sibling cache with the highest score. This technique keeps all cached objects valid. However, a certain degree of per-

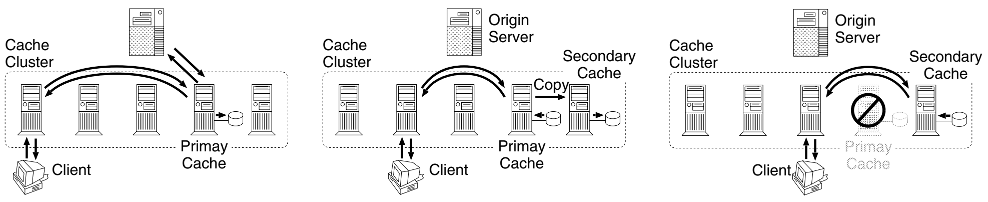

図 2.2 Duplicated hash routing (left: cache miss, center: cache hit and copy, right: failure and cache hit at the secondary cache)

formance degradation is still inevitable.

2.3.4 Proxy Auto Configuration

As a client-side technology to evade failure of a local caching node, the Proxy Auto Configuration (PAC) technology is widely used. When PAC is used, a client receives a list of proxy servers. If the local proxy is in failure, the client can bypass it and forward its requests to one of other caching proxies according to the proxy list. In our study, we assume all clients use the PAC technology.

In case of hash routing, PAC is not sufficient. For example, if a remote caching node gets failure, the objects cached at the node cannot be retrieved from the cache. All the requests for such objects result in error or cache miss depending on system implementation. In our study, we assume that such errors are avoided by forwarding the requests to the origin servers directly or to some other caching nodes.

2.4 Duplicated Hash Routing

The goal of hash routing is to optimize system cache capacity. The key idea in our algorithm, duplicated hash routing, is introducing minimum redundancy to keep its high hit rate when some of the caching nodes are in failure. Our algorithm differs from robust hash routing, mentioned in section 2.3.3, in the point that our algorithm prepares for system failure and keeps its performance during the failure period.

As illustrated in figure 2.2, an object is cached at a single node in the same way as the simple hash routing algorithm. We call this caching node a *primary* cache of this object. When a following access to the object makes a cache hit at the primary cache, it duplicates the object to another cache, which we call a *secondary* cache of the object. If the primary cache gets failure, the request for the object is forwarded to the secondary cache and makes a cache hit.

In our system, all the caching nodes share two hash functions. One is used to decide the primary cache of an object and the other is to decide the secondary cache of the object. By this cache redundancy, we can achieve robustness against failure.

Besides object duplication, we optionally enable local caching at each node. Local caching can reduce network traffic between caching nodes. Here, we discuss hit duplication and local caching in our system.

Hit Duplication With duplicated hash routing, each caching node copies its cached object to the secondary node. If the object duplication is performed frequently, the traffic between caching nodes gets increased significantly. Moreover, these duplications introduce some overlap between caches; this algorithm wastes system cache capacity. Because these copies are only for fault tolerance capability, we have to minimize the frequency of copying.

At this point, we can consider two types of solutions. One is avoiding the object duplication when the object is not expired at the secondary node. With this technique, we can reduce the network traffic caused by the object copies. However, to know whether the secondary cache has the object, some kind of querying protocol such as ICP is required, which makes the system design and implementation complicated. The other solution is taking long intervals between object duplications. If this technique can suppress the network traffic to a permissible degree without performance degradation, the querying protocol is not necessary.

Local Caching The hash routing algorithms remove overlaps of caches among nodes and optimize the system cache capacity. In compensation for its high hit rate, network traffic among caching nodes is fairly large. Though we assume the situation that the caching nodes in the system are connected with each other by high speed networks, traffic localization in the system still has great significance for system performance such as reduction of service delay. For that reason, we can optionally allow each caching node to cache objects requested by its local clients. Of course, local caching reduces the system cache capacity. If the hit rate decreases substantially when local caching is enabled, the benefit of hash routing is spoiled. Therefore, we have to examine whether the system has adequate cache capacity before enabling local caching.

2.5 Simulation Model

We evaluate the duplicated hash routing algorithm by simulation. In this section, we describe some topics on the simulation to be considered.

2.5.1 Workload

Web accesses received at a caching proxy show distribution in conformity with Zipf's law[1] [158, 157]. More precisely, the number of requests f for an object is expressed as $f = C/r^k$, where r is the access ranking of the object and C and k are constants depending on access characteristics of the user community. However, Zipf's law cannot explain the characteristics of whole web accesses from various communities.

To simulate a distributed WWW caching system, we put a strong assumption that the total set of requests from all clients in the system follows Zipf's law. Although this assumption satisfies the requirement that the distribution of accesses received at each caching node conforms Zipf's law, the sufficient condition of this proposition is not proved yet. Even if this assumption is somewhat different from actual situations, we can get some hints on the transition of hit rates or network traffic according to the variation of several parameters of the system.

In Zipf's law, k means the scatter of requests from clients and C means the order of the total number of requests. In our simulation, we set the parameter of k at 0.66, which is reported in [159]. From the parameter k and the total number of requests, we can calculate the parameter C. In our simulation, we set C at 50,000 and generate about 35,000,000 requests.

2.5.2 Object Size

The distribution of WWW object size conforms log-normal distribution [160]. Its probability distribution function is expressed as follows.

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} \exp\left(\frac{-(\log x - \mu)^2}{2\sigma^2}\right)$$

We set the parameters of μ and σ at 7.3 and 1.7 respectively observed in [159] and get the probability distribution of object size. Using this distribution, we choose the size of each object within the limit of 100MB. Too large objects give not a small influence to cache efficiency especially when its capacity is relatively small, even though appearance of such large objects is quite rare. Moreover, download of such a huge object through WWW is unrealistic.

2.5.3 Replacement Algorithms

Dozens of object replacement algorithms for a

[1] This distribution is called a "Zipf-like" distribution in [157], because it does not follow strictly the original Zipf's law.

cache are proposed, and developing better algorithms is still one of hot topics in the field of WWW caching (for example, see [161, 162, 163]). However, a better cache replacement algorithm is not the main motivation of this study, we adopt a classical LRU algorithm in our simulation, which is proved to achieve a moderate hit rate compared with other algorithms.

2.5.4 Failure Rates

System failure can be characterized by two factors, mean time between failures (MTBF) and mean time to recover (MTTR). In our simulation, we take the number of requests received by a caching node in substitution for passage of time at the caching node. In other words, our simulation is request-driven.

We fix the parameters of C and k to produce about 35,000,000 requests in total and assume that 2% of these requests meet failure at their local caching nodes[2]. We set the mean number of request receptions between failure at 100,000; each caching node recovers after a certain number of request receptions, which is determined randomly from 0 to 200,000. Under this situation, the occurrence probability of failure at each request is set at 0.02/100,000 (2×10^{-7}).

To discuss robustness against system failure, it is important to use realistic values of parameters. In our simulation model, at a caching node which receives 100,000 requests per day from its clients, MTBF is 50 days and MTTR is one day (and this node shall be repaired within two days from an occurrence of failure). We consider this situation is typical of many organizations where caching proxies are operated.

2.5.5 Copy Intervals

With the duplicated hash routing algorithm, a cache hit may cause an object copy. Each caching node has its threshold of a copy interval. When a cache hit occurs, the node checks the time when it copied the object for the last time. If a longer period than the threshold passes after the last copy, the node tries to copy the object to the secondary node. A system with capability of querying an object to the secondary node checks whether the object is cached at the secondary node; it can duplicate the object only when the object is already expired.

In our simulation, the threshold of an interval between object duplications is decided by the number of requests the caching node receives from its local client. By default, object duplication can occur after d/s receptions of accesses, where d and s denote the cache size of the node and the mean size of objects, respectively. Here, we introduce a copy factor, a factor of the threshold, and examined the cases that the copy factor is 0.0, 0.25, 0.5, 1.0, 2.0, 4.0, and 8.0. The copy factor of 0.0 means that caching node checks and/or duplicates the object every time a request makes a cache hit. The copy factor of 8.0 means that a caching node duplicates an object after $8.0d/s$ receptions of requests.

2.5.6 The Number of Nodes and Cache Size

In a simulation of a WWW caching system, it is important to choose appropriate disk size against the number of requests processed by a caching node. In our study, the system receives about 35,000,000 requests as a whole. This means that total of about 300GB of data is retrieved by clients. We set the total system cache capacity at 32GB, 64GB, 96GB, 128GB, and 160GB. We can examine both situations with poor and rich cache capacity.

Cache capacity of each node is decided by this total capacity. For example, the capacity of a single cache is set at 4GB when the system capacity is 64GB and the number of nodes in the system is 16.

[2] However, these requests are forwarded to some other nodes and do not result in errors by the PAC technology.

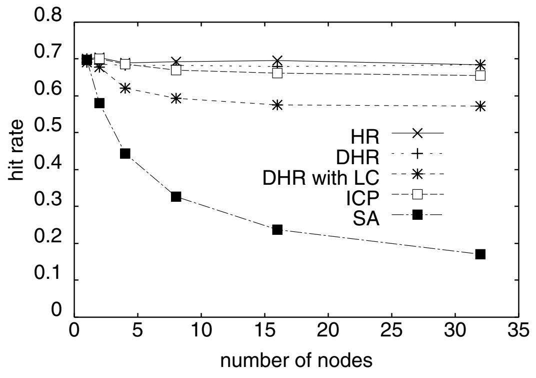

図 **2.3** Hit rates of robust hash routing (HR), duplicated hash routing (DHR), duplicated hash routing with local caching (DHR with LC), internet cache protocol (ICP), and stand alone caching (SA) with 64GB of cache capacity

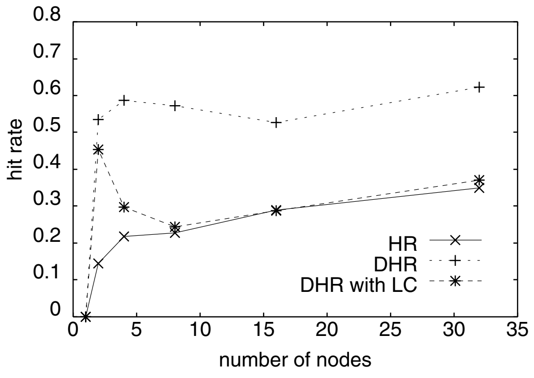

図 **2.4** Hit rates on failure (with 64GB of cache capacity)

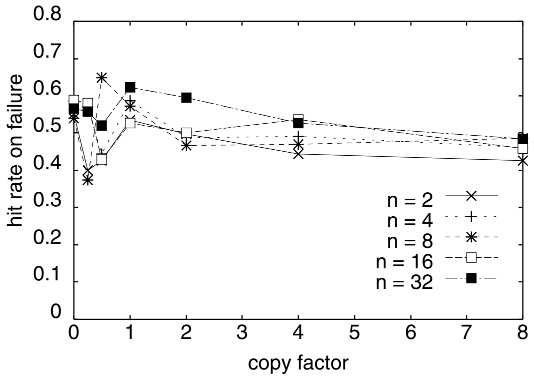

図 **2.5** Hit rate on failure and copy factor of duplicated hash routing (n is the number of nodes and cache capacity is 64GB)

2.6 Results

We simulated caching systems, stand alone caching, ICP-based caching, robust hash routing and our duplicated hash routing. We discuss the performance of these algorithms in terms of hit rates, cache capacity, and network traffic. Robust hash routing and duplicated hash routing are examined in both cases with local caching enabled and disabled. In case of ICP and stand alone caching, local caching is always enabled.

2.6.1 Hit Rates

Figure 2.3 depicts total hit rates with the robust hash routing algorithm, the duplicated hash routing algorithm, the internet cache protocol, and the stand alone caching. As this chart shows, distributed caching systems can achieve higher hit rates.

The algorithm that gets the highest hit rate is robust hash routing. Because overlaps of caches between nodes are removed with robust hash routing, the hit rate is kept high even if the scale of the system grows. Duplicated hash routing also attains a hit rate as high as robust hash routing does. This implies that the decrease of system cache capacity caused by cache duplications has almost no influence on the hit rate of the system. However, when we enable local caching, the hit rate decreases about 10%. This means that we cannot ignore the decrease of system cache capacity caused by local caching. Here, we have to recall the suitability of system cache capacity to the number of requests. The high hit rate of the hash routing algorithm indicates that this capacity is large enough to produce ideal results. We will discuss the cache capacity later in section 2.6.2.

Contrary to our expectation, the difference in hit rates of ICP and hash routing is small. However, this high remote hit rate is obtained at the expense of large traffic between caching nodes, as we will mention later in section 2.6.3.

In case of failure at a caching node, both robust hash routing and duplicated hash routing can continue their services to their clients. The objects which are to be cached at the failed node are cached at some other nodes decided by their scores or the secondary hash function. In contrast to robust hash routing, duplicated hash routing can prepare cached objects at secondary nodes for system failure. In other words, as for failure, robust hash routing always makes a *cold start*. We extract requests for objects which are to be cached at failed nodes and examine the hit rates of them. As figure 2.4 shows, our algorithm keeps high hit rate even in case of failure. When local caching is enabled, however, the hit rate degrades considerably. Local caching decreases system cache capacity and that gives more influence on these hit rates than on the total hit rates. We will discuss this topic later in section 2.6.2.

We also examine a relationship between hit rates in case of failure and copy factors, and the result is depicted in figure 2.5. Though we expected that the hit rate decreased rapidly as we increased the copy factor, this result indicates that our expectation is fortunately not true. The impact of the copy factor on the hit rate is turned to be fairly small.

2.6.2 Cache Capacity

In the previous section, we mentioned that local caching wasted the cache capacity of the system. This means that we can keep the high hit rate if we add extra cache capacity to the system. We simulated the systems with robust hash routing and duplicated hash routing in case with and without local caching.

Figure 2.6 shows the hit rates of two algorithms without local caching. From this result, we can see that 64GB of system cache capacity is enough to this request sequence. When we set the capacity at 32GB, the hit rate gets evidently lower than the others. However, the hit rates with more than 64GB of cache capacity are almost the same.

Figure 2.7 represent the cases we enable local caching. With low cache capacity, local caching makes hit rates low. However, with larger cache capacity, we can prevent the decline of hit rates

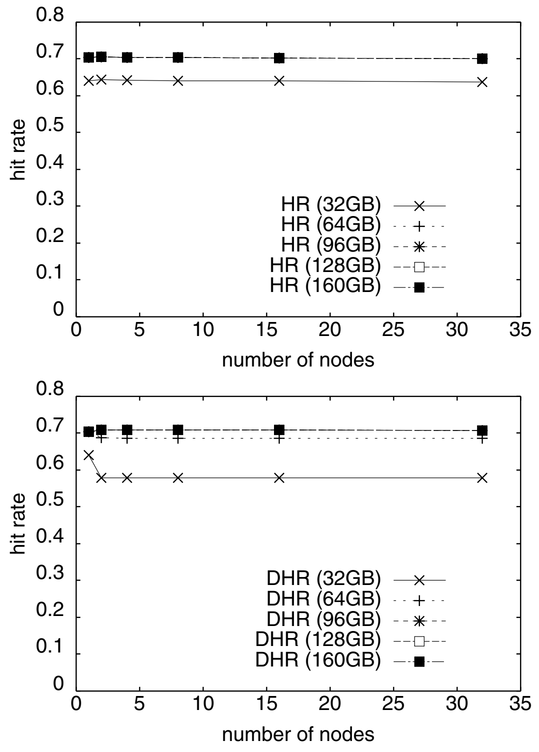

図 2.6 Hit rates with local caching disabled (upper: robust hash routing, lower: duplicated hash routing)

caused by local caching. From the result, we can see that almost the same hit rate as the case without local caching can be achieved with 128GB of cache capacity in both hash routing systems and duplicated hash routing systems.

Figure 2.8 shows a correlation between the hit rate in case of failure and the total cache capacity using our algorithm with local caching enabled. As we mentioned in section 2.6.1, this hit rate degrades when local caching is enabled. However, this graph indicates that a fairly high hit rate can be achieved with larger cache capacity.

2.6.3 Network Traffic

Figure 2.9 shows the network traffic between caching nodes. In this result, we count neither the local traffic between clients and their local caching nodes nor the traffic between caching nodes and origin servers, because we focus on intra-system traffic here.

表 2.1 Summary of hash routing, duplicated hash routing, duplicated hash routing with local caching, ICP, and stand alone caching

	total hit rate	hit rate on failure	network traffic between nodes	scalability
HR	high	medium	medium	medium
DHR	high	high	medium – large (improvable)	medium
DHR with LC	medium (improvable)	medium (improvable)	medium (improvable)	high
ICP	high	high	depends on the number of nodes	low
SA	low	nil	nil	–

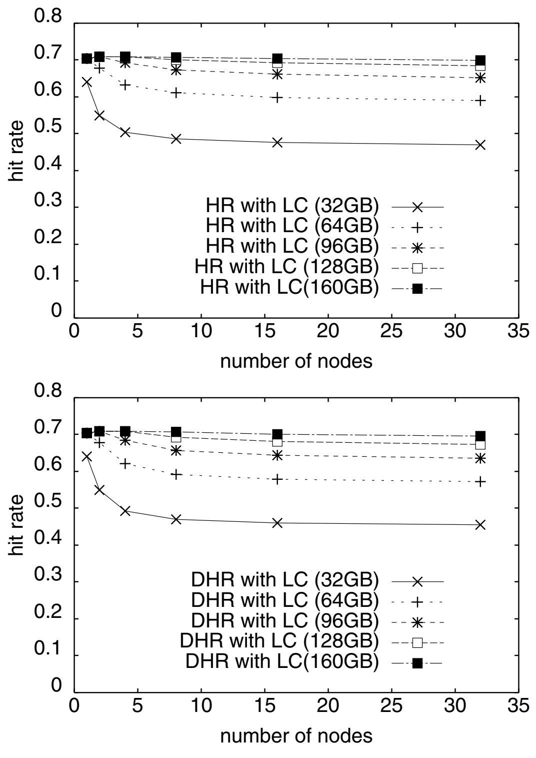

図 2.7 Hit rates with local caching enabled (upper: robust hash routing, lower: duplicated hash routing)

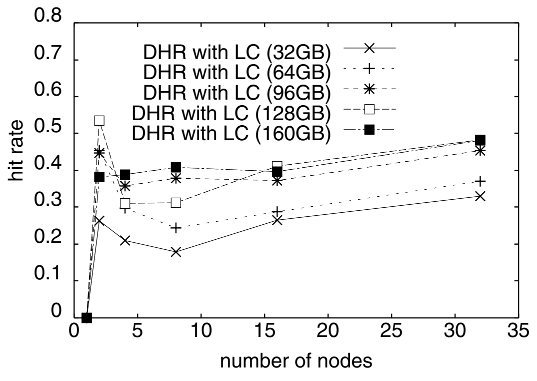

図 2.8 Hit rates of duplicated hash routing on failure with local caching

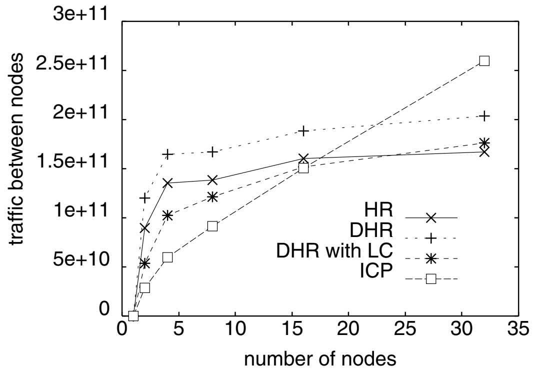

図 2.9 Network traffic between nodes with hash routing, duplicated hash routing, duplicated hash routing with local caching, and ICP (with 64GB of cache capacity)

When the number of caching nodes is small, ICP requires relatively small network traffic between nodes. However, ICP cannot adapt to a large scale system, because it uses broadcast to send queries to all nodes. Even if each query is small in its size, the total network traffic between caching nodes gets explosively large as the system scale grows up. On the other hand, the increase rates of network traffic with robust hash routing and duplicated hash routing are fairly low. The

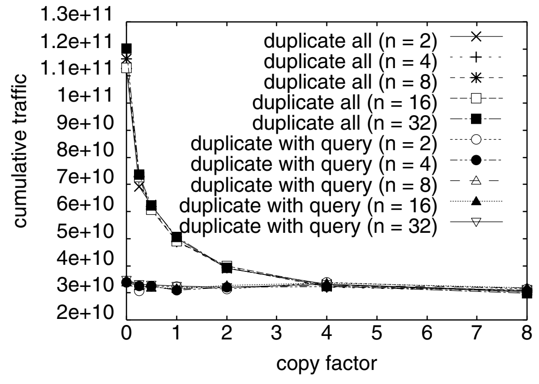

図 2.10 Network traffic and copy factor with duplicated hash routing (n is the number of nodes and cache capacity is 64GB)

difference in the traffic between these two algorithms is caused by object duplication that is a feature of duplicated hash routing. To make the network traffic low, it is the best way to transmit an object to the secondary node only when the object is not cached by the node. However, it requires a querying protocol and takes some communication cost between caching nodes. Instead of this querying technique, we take an approach of employing a large copy factor. In our simulation, we set the copy factor of duplicated hash routing at 4.0 and it makes the network traffic between caching nodes fairly low. We will revisit this topic later.

When we enable local caching in a duplicated hash routing system, the increase rate of network traffic gets slightly increased. However, compared with the duplicated hash routing algorithm without local caching, it significantly reduces the network traffic especially in case with a small number of caching nodes.

As we mentioned before, we can decrease the traffic between nodes by increasing the copy factor. Figure 2.10 shows the relation between the copy factor and the network traffic between caching nodes. This graph shows two patterns: one is a case that each node always duplicates an object without a query after an interval based on its copy factor and the other is a case with a query. In the former case, unnecessary object duplications raise the network traffic when the copy factor is small. In the latter case, we put the communication cost of sending a query to the secondary node is the same as that of an ICP query. From the results, the difference between the two patterns is neglectable if we set the copy factor larger than 4.0.

Considering the fact that the copy factor does not affect on the hit rate so much, as we mentioned in section 2.6.1, it is unnecessary to send queries to secondary caches if we set the copy factor at more than 4.0. If we avoid such queries, we can keep the system design simple and its implementation easy.

2.6.4 Summary

We summarize the features such as total hit rates, hit rates on failure, network traffic between caching nodes, and system scalability of each algorithm in table 2.1. Our extension to hash routing gains durability against system failure at a relatively small expense of increased network traffic. Moreover, this expense can be reduced by local caching and a large copy factor. ICP also achieves both high durability and high hit rates. However, its scalability is highly restricted by its large increase rate of network traffic.

2.7 Conclusion and Open Issues

WWW becomes an indispensable service on the Internet and the fault tolerance of systems which provide the WWW service has great importance. We have proposed the duplicated hash routing algorithm for a distributed WWW caching system, which has robustness against failure. Our simulation results suggest that our algorithm can achieve a high hit rate even when some caching nodes are in failure. In addition, optionally we can enable local caching at both robust hash routing system and our duplicated hash routing system. If we enable local caching, the traffic between caching nodes is reduced, though the cache capacity of the system gets lower. Therefore, the total hit rate also gets lower when local caching is enabled. This phenomenon can be prevented as we add extra cache capacity to the system. If we increase the cache capacity, we can both keep the high hit rate achieved by robust hash routing and reduce network traffic between caching nodes substantially. In duplicated hash routing system, intervals between object duplications have great importance. With a relatively long copy interval, we can omit the object querying mechanism from the system and design the system to duplicate objects always after the interval because the increase of network traffic is small enough.

We can point out lack of discussion about service delay in our study. Because one of the major goals of WWW caching system is reducing the service delay experienced by end users, an analysis of service delay is important. However, we can only mention here that our extension to the hash routing algorithm adds no extra delay. Object duplications are not necessarily synchronized strictly with requests from clients and can be prioritized low in the system. More precise analysis of service delay is our future work.

第3章 An Analysis of WWW Server Status by Packet Monitoring

A management of WWW server is still relying on the expertise and heuristic of administrators, because the comprehensive understandings of server behavior are missing. The administrators should maintain the WWW server with good states that they should investigate the WWW server in real time. Therefore, it is exactly desirable to provide a measurement application that enables the WWW server administrators to monitor WWW servers in the actual operational environment. We developed a measurement application called ENMA (Enhanced Network Measurement Agent) which is specially designed for WWW server state analysis. Furthermore, we applied this application to the large scale WWW server operation to show its implementation and advantages. In this paper, we analyze the WWW server states based on precise monitoring of performance indices of WWW system to help the server management.

3.1 Introduction

The world enters the era of global communications. The Internet is rapidly growing and has become a significant fundamental infrastructure in this era. Especially, World-Wide Web (WWW) is the most important service on the Internet. Therefore, providing WWW services without any troubles becomes a more important issue.

To achieve WWW server operation with less server suspending period, there are many tools and systems proposed so far. However, they still require expertise and heuristic to the administrators, because these tools do not provide the comprehensive view of the WWW server behavior.

To ease the server administrators, we developed a set of applications to estimate the WWW server performance in real time, called ENMA [164] that stands for Enhanced Network Measurement Agent. The ENMA can measure several performance indices of the WWW server such as a parameter of TCP connections and the traffic volume of the WWW server processes, and provide the more precise view of the server behavior in graphical forms.

Therefore, this application can be used to know if the WWW server is overloaded by the excessive requests. The ENMA also can give hints on which software module related to the WWW server system is not implemented correctly.

We propose the state transition model of the WWW server behavior. With this model, we develop a new measurement method of server performance, and we give some hints to the server administrators, which enable them to measure the WWW server easily.

3.2 Server Performance Measurement Method

In this section, we describe some performance measurement method of the WWW server, and show advantage or drawback of these methods.

3.2.1 Log Analysis

The most popular way of performance analysis of the WWW server is an analysis of the WWW server log files to know the total byte transferred from the server to clients, the number of HTTP requests, and request rates to the server. However, this analysis cannot provide any hints on server behavior such as response time, data transfer time and the number of concurrent connections.

3.2.2 Kernel Level Monitoring

Web Monitor [165] is one example of kernel level monitoring. Kernel level monitoring reveals a number of performance indices related to the WWW server operation such as the total number of network I/Os in a system call, signals and file system operations, etc. However, this method obviously requires modifications of the kernel to know some performance indices. The modification of the kernel is a sophisticated task in general so that this method can be considered difficult for ordinary administrators. Furthermore, commercial OS and server systems with special hardware cannot be measured in this method, because there is no way to make modification on the kernel. This method also may cause serious performance degradation in the server.

3.2.3 Benchmarks

SPECweb99 [166], WebStone [167] and httperf [168] are famous benchmark software for the WWW server. The benchmark can evaluate their performance and give many hints for administrators to set up its operational environment. However, it is quite hard to apply this benchmark software to the WWW server in the actual operation, because it is hard to suspend the WWW service only for the benchmarking. Therefore, the benchmark does not help administrators to improve the WWW service in operation.

3.2.4 Network Monitoring

We can know the server performance to observe packets on the network segment attached the WWW server. The network monitoring does not cause any influences on the WWW server and its internals, however, it is required to have software to make analysis on the monitored packet. It is popular to use tcpdump [10] to monitor the packets. Since this program is a general-purpose monitoring tool, we have to develop a set of programs to process the output generated by the tcpdump and to know the HTTP protocol interactions and packet payloads.

MRTG [169] is also a traffic monitoring tool using SNMP applied to routers and switches. The administrator can measure the traffic using the MRTG on the network attached to the WWW server. MRTG is suitable for the long-term monitoring such as 1 week or month long, because of its observation granularity.

3.2.5 Our Strategy

ENMA is an application for the WWW server performance measurement using packet monitoring developed by our research group.

To ease the server administrator, ENMA can provide performance indices to help them to manage the server: the number of the currently processing connections, the number of processed connections per second, the volume of traffic and so on. ENMA can also supply a function to describe these performance indices in real time with a visualization module.

It is desirable that our measurement is monitored for the short-term. Therefore, since the processing in the server is saturated, we must report the server administrator to its trouble.

ENMA can monitor the special HTTP packets precisely to observe the server behavior. For example, ENMA enables the server administrator to understand the response time in the server con-

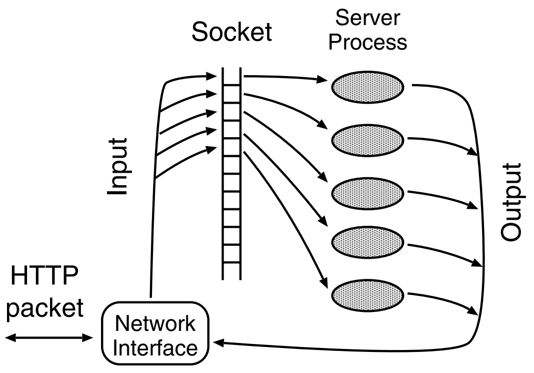

図 3.1 The model of server internal processing flow

cerning the HTTP request of clients. ENMA also can set up an easy configuration of observation granularity to observe the server behavior for the short-term. Furthermore, to mention the server administrator on performance indices, ENMA can measure many performance indices: the volume of traffic, the number of the currently processing connections, the number of processed connections per second and the number of arrival connections per second.

3.3 Performance Indices in the Server System

In this section, we discuss the performance indices to decide the server state.

Section 3.3.1 describes the processing model of TCP connection at HTTP communication in the server. Section 3.3.2 discusses about performance indices based on outside the server.

3.3.1 Connection Processing Flow in the Server

Figure 3.1 shows general flows to process a TCP connection in the server. Step one is to allocate a socket for establishing TCP connection when SYN packet arrives from clients. Step two is server program processing the established connection for receiving the HTTP request and transmitting the HTTP response to clients.

Consequently, the number of sockets and the number of server processes are important performance indices in the server.

3.3.2 Measurement Outside the Server

In case of measurement from outside the server, several performance indices ware required: connection processing rate, connection arrival rate and the number of the currently processing connections. We present these performance indices as follows.

- Connection processed rate

  The connection processed rate is the number of processed connections per unit time in the server.

- Connection arrival rate

  The connection arrival rate is the number of arriving connections per unit time from clients.

- The number of the currently processing connections

  The number of the currently processing connections means a set of TCP connections handled concurrently in the WWW server.

Comparison between the connection processing rate and the connection arrival rate reveals whether the server is enough to provide WWW service or not. Measuring the number of the currently processing connections is enabling the server administrator to know the processing number of TCP connections in the server at that time.

3.4 Experiments to Define the Server State

In this section, we present experiment to decide the server behavior based on performance indices through the benchmark. In this experiment, to get rid of an overhead by disk I/O, we set up that client generate HTTP request by fixed URL. We observed the difference of server performance when server system is saturated with several server configurations using the above performance indices. The object of these experiments is what make up that server becomes saturate state and measures server state at that time.

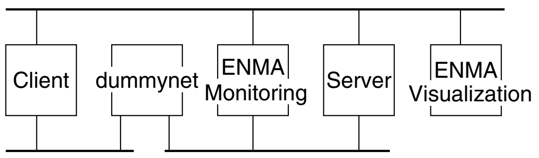

図 3.2　Experiment Network Architecture

表 3.1　Configuration Parameter

	Part	No. of sockets	No. of processes (start/min/max)	request rate
1	1	1064	5/10/150	80
	2	16424	5/10/150	400
2	1	16424	5/10/150	700
	2	16424	5/100/150	700
3	1	16424	5/100/256	1000
	2	16424	128/100/256	1000
	3	16424	256/100/384	1000

3.4.1 Environment

Figure 3.2 shows experiment environment. Five hosts are used: server, client, monitor, dummynet [170] and visualization.

The client generates HTTP requests to the server and the monitor observes the packet on the network to the server. The dummynet inserts the delay of 100 milliseconds to emulate WAN environment. This order of delay is usual in many actual network. Server, client, monitor, visualization use FreeBSD 3.2 and dummynet use picoBSD. A server program is apache [171] 1.3.9 and a client program is httperf 0.6.

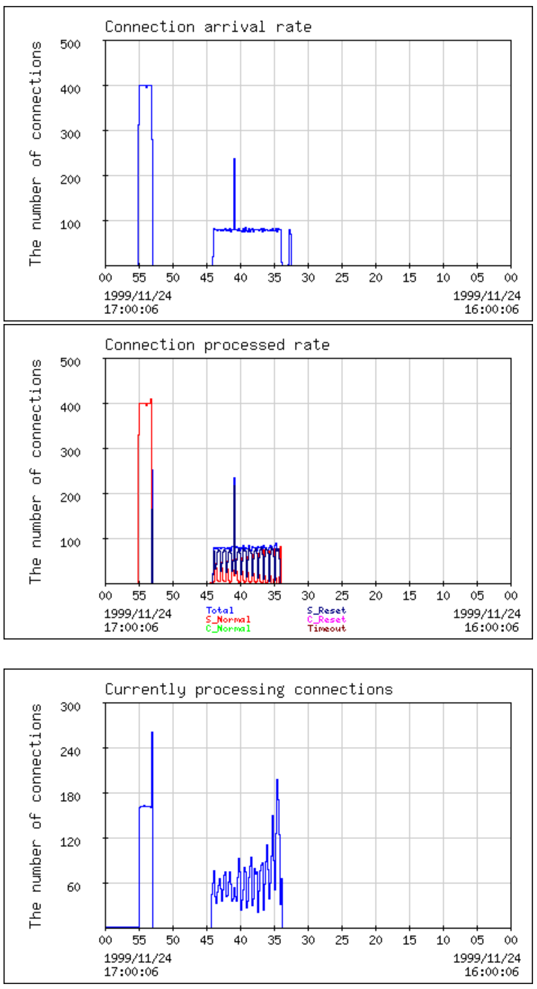

図 3.3　Shortage of the number of sockets

3.4.2 Results

Table 3.1 shows the configuration parameter of kernel and service program.

In this experiment 1, we notice the maximum number of sockets in the server sytem. The experiment 2 is modified to the maximum and minimum number of server processes.

In the experiment , we discuss a trade-off both the number pf server processes and CPU idel cycles.

3.4.3 Experiment 1

This experiment consists of two parts. The part 1 is between 16:34 and 16:44 and uses GENERIC kernel parameter of FreeBSD and default configuration file of apache. The HTTP request rate is 80 request per second in table 3.1. The part 2 is between 16:53 and 16:55 and uses as increase the number of sockets from 1064 to 16424 in the kernel and default configuration file of apache. The HTTP request rate is 400 request per second. In the part 1, the connection arrival rate becomes stable, however, the connection processed rate and the currently processing connections become unstable in figure 3.3. However, in the part 2, the connection arrival rate, the connection processed rate and the currently processing connections are stable.

As the result of two parts, the number of sockets in the kernel is important performance index in

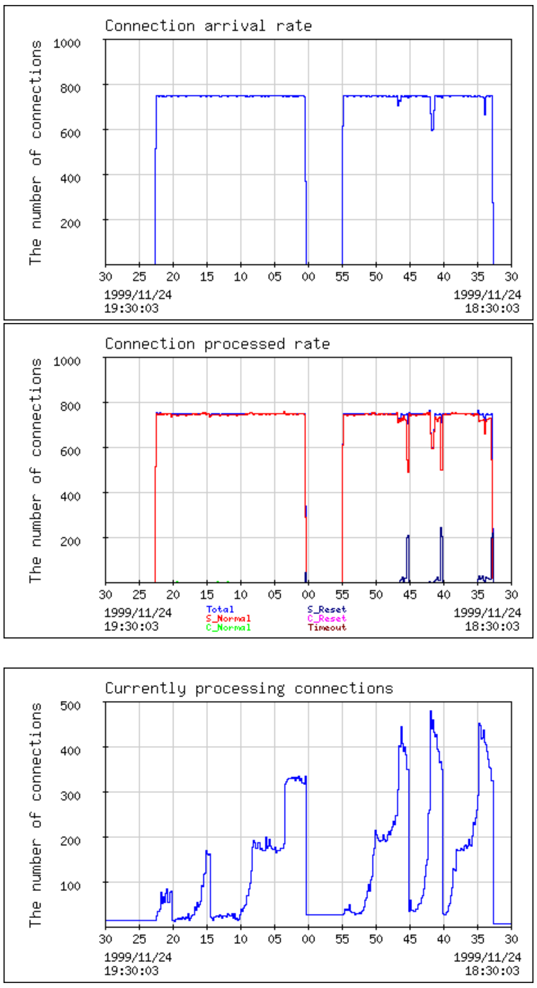

図 3.4 Shortage of the number of server processes

the server system.

3.4.4 Experiment 2

The experiment 2 also consists of two parts. The part 1 is between 18:33 and 18:55 and uses 16424 sockets, 150 maximum server processes and 700 request per second. The part 2 is between 19:00 and 19:23 and used 100 minimum server processes in table 3.1.

We consider saturated state in the server around 18:40 and 18:45 shown in the part 1 in figure 3.4, because the connection processed rate appear at spike form at that time. The currently processing connections are rapidly increased at saturate state, because the server program executes fork

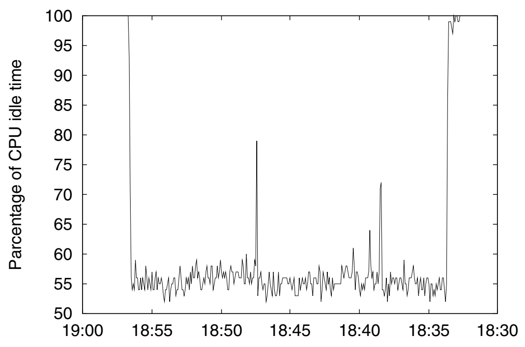

図 3.5 Percentage of CPU idle time

system calls at 18:40 and 18:45 so that its execution cause page faults in the server system. In the part 2, the minimum number of server processes of apache is modified. Both the connection arrival rate and the connection processed rate is stable.

As shown in figure 3.5, the CPU has idle cycles, because this system may be able to process more HTTP requests. Then, the administrators should take care of the parameter of the maximum and minimum number of server processes.

3.4.5 Experiment 3

This experiment also consists of three parts. The part 1 is between 20:32 and 20:48 and uses 16424 sockets, 256 maximum server processes, 100 minimum server processes and 1000 request per second in table 3.1. The part 2 is between 20:52 and 21:07 and uses 16424 sockets, 128 start server processes and 1000 request per second. The part 3 is between 21:15 and 21:21 and uses 16424 sockets, 384 maximum server processes, 100 minimum server processes, 256 start server processes and 1000 request per second.

In figure 3.6, the saturated state in the server occurs from 20:32 to 20:33, from 20:37 to 20:40 and from 20:44 to 20:47 in part 1. The reason of this saturated state is insufficiency of CPU cycles shown figure 3.7.

The saturate state also appears in the workload 2. Therefore, if the request rate increases frequently, modifying the parameters in the server

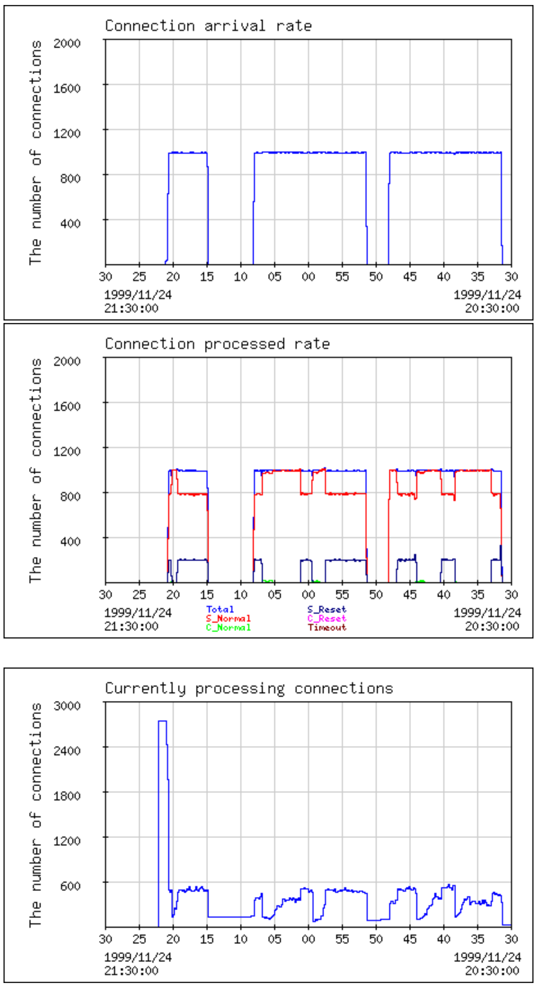

図 **3.6**　Shortage of system ability

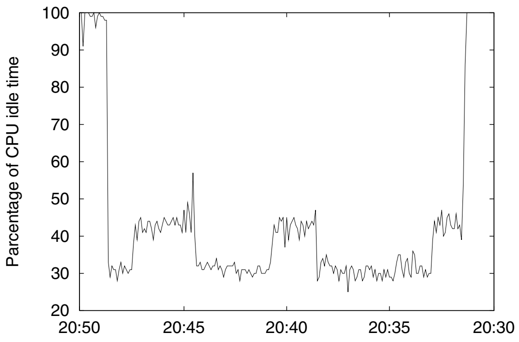

図 **3.7**　Percentage of CPU idle time

system does not help so much. Then, administrators must think to replace a new system.

On the other hand, The number of concurrent connections rapidly increases at 21:21 in figure 3.6 in workload 3, because system panic cause.

3.4.6 Consideration

As above results, in case of operating the apache server system, we can show the following recommendation.

- The server state transfers stable to unstable modifying the number of sockets as shown in experiment 1.

- The server state also transfers stable to unstable modifying the number of maximum and minimum server processes as shown in experiment 2.

- Between to maintain maximum performance of the server and to be the stable server system is trade-off. It is between the number of server processes and processing ability of CPU when the request rate increases as shown in experiment 3.

Consequently, we conclude the important performance indices to estimate the server state: the maximum number of sockets, the number of server processes, processing ability of CPU.

3.5 The Definition of Server State and Management Method

In this section, we describe management method corresponding to the server state based on performance indices. We can divide the server state in three states.

1. Running State
   We define the running state as that the server responds to the HTTP request from clients without some troubles. The server administrator should maintain the WWW server in this state.
2. Saturate State
   The saturate state is defined as that some TCP connections are not processed correctly in the server. A server administrator should avoid for the server to remain in this state.

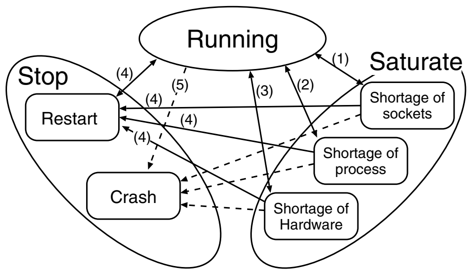

図 **3.8** State Transition in the Server

We can divide the saturate state more precisely in three states based on the previous experiments as follows

- Shortage of sockets
- Shortage of server processes
- Shortage of system ability

3. Stop State

The stop state is defined as the server system halting. We can divide the stop state in two state as follows.

- Restart
- Crash

Figure 3.8 shows state transition model based on the experiments.

The experiment 1 reveals the state transition (1) from the running state to the shortage of sockets in figure 3.8. If the WWW server enters the shortage of sockets, the server administrator should increase maximum sockets in the server system. For example, we suggest the administrator to increase *maxuser* that is one of the parameter in general BSD Unix system.

The experiment 2 shows the state transition (2) from the running state to the shortage of server process in figure 3.8. If the WWW server enters the shortage of server process, the server administrator should increase the maximum and minimum number of server processes. For example, we recommend the administrator to increase *MaxClient* and *MaxSpareServers* parameters in apache server system.

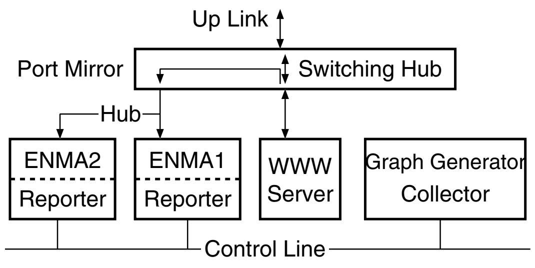

図 **3.9** Network Architecture of Target System

The experiment 3 shows the state transition (3) from the running state to the shortage of system ability in figure 3.8. If the WWW server enters the shortage of system ability, the server administrator should find out a bottleneck in the server system. If the processing ability of CPU is not sufficient, it should be exchanged to a faster one. If the amount of memory is not enough, the server administrator should install more memory.

The state transition (4) from the running, the shortage of sockets, the shortage of process and the shortage of system ability to the restart is performed by the server administrator. After the restart, the server state returns the running state.

In these experiments, we can see the state transition (5) only once from the running to the crash. Another state transitions may exist, because finding these state transition is future work.

3.6 An Example of Observation in an Actual Server

To estimate the WWW server performance, we applied the ENMA to observe the WWW server in actual operation. The target WWW server is the server of "The 81th National High-school Baseball Games on the Internet".

3.6.1 Environment

Figure 3.9 shows the network architecture of the target system. The application and the system specification of the target system are shown in table 3.2. The ENMA can measure the performance indices in the server through port mirroring of the attached server. We provided two ENMA

表 3.2　The System Environment of Observation

	WWW server	ENMA host 1	ENMA host 2
OS	SunOS 5.6	SunOS 5.6	FreeBSD 3.2
CPU	Ultra SPARC II 300MHz (x2)	Ultra SPARC II 300MHz (x2)	PentiumII 400MHz
Memory	512MBytes	256MBytes	128MBytes
Appli-cation	Apache 1.3.6	ENMA (19990823)	ENMA (19990823)

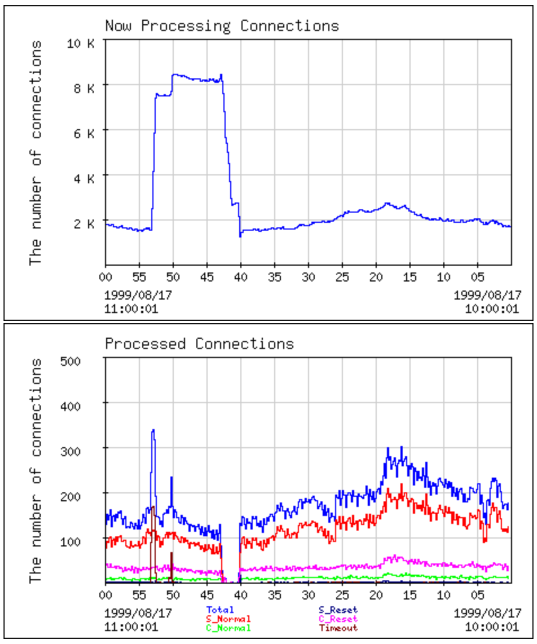

図 3.10　Restart the WWW server

hosts per one server to avoid an obstruction in the ENMA host by some trouble. Each ENMA hosts send the real time data to analysis host using control line and the analysis host generates graphs of current states in the WWW server.

3.6.2 Case Studies

3.6.3 Restart the WWW Server

Figure 3.10 illustrates graphs when the server program restarts.

The number of the currently processing connections appears increasing rapidly and the number of processed connections is reduced to zero at 10:40 August 17 in figure 3.10.

We consider that the server administrator restarted the server by some reasons at that time.

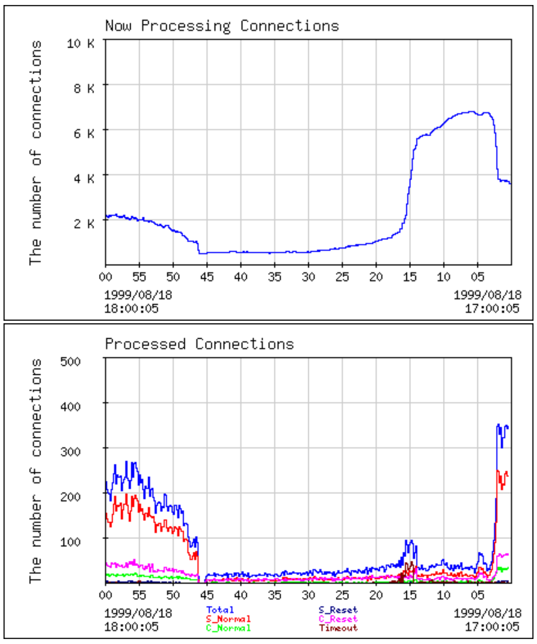

図 3.11　Server System Saturated

3.6.4 Server System Saturated

Figure 3.11 shows the state of the server system saturated.

The number of the currently processing connections appears increasing rapidly and the number of processed connections is reduced at 17:02 August 18 in figure 3.11.

When the number of connection timeouts observed by ENMA raises rapidly, the number of the currently processing connections decreases at 17:15. At 17:02, the server administrator judged the shortage of the amount of system memory by system messages and restart the system at 17:45 after increasing the swap space.

In this saturate state, the administrator decided the server system to be the shortage of system ability through the system messages. However, they should investigate the parameter in the server system. Therefore, the number of maximum server process was 1536 and the percentage of the CPU idle time was between 40 and 50. In this condition, we judge the reason of the shortage of the amount of memory through the too many number of maximum process.

Therefore, the server administrator must set up the maximum and minimum number of server processes corresponding to system ability.

3.7 Future Works

3.7.1 The Alert Method to the Administrator

We need to design the method to alert an administrator when the troubles are occurring in the WWW server. For example, ENMA mails the troubles to the administrator when the WWW server is in the saturate state.

3.7.2 Building a system that gets server internal state

To knowing the server internal state, we should get the socket activity ratio, the number of server processes, consumption of memory, percentage of CPU idle time and so on. In our experiments, we use vmstat program. However, this program is not enough to get these performance indices. Therefore, we must implement a new system to be able to obtain these performance indices.

3.8 Conclusion

World-Wide Web (WWW) is the most important service in the Internet. To maintain the WWW service, it is quite important to run the server with less suspended period of its service. However, server management method still require expertise and heuristic to the administrators, because measurement tools do not provide the comprehensive view of the WWW server behavior. In order to ease the administrators, we developed an application, called ENMA, to monitor several performance indices of the WWW server in the actual operation in real time. We derive the server state through our laboratory work using ENMA. We also analyze the server status based on some performance indices.