Development of a WWW Server Management Support System

Yutaka Nakamura
yutaka-n@rd.center.osaka-u.ac.jp
Shinji Shimojo
shimojo@cmc.center.osaka-u.ac.jp
Cyber Media Center of Osaka University

Suguru Yamaguchi
suguru@is.aist-nara.ac.jp
Nara Institute of Science and Technology

Eiji Kawai
eiji-ka@is.aist-nara.ac.jp
Japan Science and Technology Corporation

Hideki Sunahara
suna@is.aist-nara.ac.jp
Nara Institute of Science and Technology

Abstract

Internet becomes popular, and many network servers are standing up. These network servers are evolving to large scale servers, because they provide many services such as broadcast of several events, reservation of a travel ticket and so on. Server administrators should offer stable service to users. In addition, requests from clients are balanced to several distributed WWW servers, and they should be thus managed in a centralized manner, instead of a distributed one. In this paper, we develop a new system to help the administrators to manage a distributed server system in a centralized manner. For example, that will enable the administrators to recognize the status of the server, and further to detect whether the system is overloaded with requests or not in a timely manner. That will be also able to reduce the cost related to the management.

1. Introduction

A lot of societies are shifting to the information-oriented ones. In such a society, the Internet will act as nerves that convey every conceivable kind of information from/to everywhere. Currently World-Wide Web (WWW) is one of the most important services on the Internet, and it becomes accessible through non-PC devices such as portable telephones and PDAs. Thus, a vast number of requests are surging to the WWW servers all over the world.

WWW is a vital service in the information-oriented society and the quality of WWW service should be thus strictly managed. These network servers are evolving into large scale servers, because they provide many services such as broadcast of several events, reservation of travel tickets and so on. Server administrators should offer stable service to users. In addition, requests are balanced to several distributed WWW servers, and they should be thus managed in a centralized manner, instead of a distributed one.

Many administrators still perform job control relying on their intuition and experiment that frequently cause misconfiguration of the servers, because administrators have not been established performance indices for server management. We can investigate the performance of server systems quantitatively by benchmark tests on LAN environment. However, the results of estimation by the benchmark system are different from the actual performance of the operated server system. Consequently, server administrators desire a system that makes evidently detection of bad condition of the server and the reason of occurrence of the event. Furthermore, they want the management information of distributed server systems collectively in real time.

In this study, we develop a new system to help the administrators to manage a distributed server system in a centralized manner. Our system will enable the administrators to recognize the status of the server, and further to detect whether the system is overloaded with requests or not in a timely manner. It will be also able to reduce the cost related to the management.

2. Performance indices of a WWW server

In this section, we give some performance indices to represent a WWW server system.

2.1 Performance indices per TCP connection

- Response time \(T_r\)

The definition of Response Time of a HTTP transaction is the period between the time the first HTTP request packet from the client is captured and the time
the first HTTP response packet from the server is captured. Note that this definition is slightly different from the generic definition of RTT (Round Trip Time). Since we discuss the behavior of WWW services through the packet monitoring, we use the period A in Figure 1. Therefore, the Response Time is normally shorter than the RTT, in general.

- **Connection Continuation time \(T_c\)**
  Connection Continuation Time is defined as the period from TCP connection establishment to its shutdown. More precisely, the period from the time when the first SYN packet (the packet #1 in Figure 1) transmitted to the server by the client is observed to the time when the server’s ACK packet (the packet #6 in Figure 1) corresponding to the last FIN packet generated by the client is monitored.

- **Data Transfer time**
  Data Transfer Time is defined as the period from the time when the first HTTP data packet transmitted by the server is observed to the time when the last data HTTP packet generated by the server is detected (The period B in Figure 1). Note that the Data Transfer time includes the period for exchanges of FIN packets and their ACK, because these packets sometimes include the actual WWW data.

2.2 **Performance indices of total server system**

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

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

- **The number of concurrently processing connections**
  The number of the concurrently 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.

2.3 **Performance indices into a server system**

The rep2 retrieves the following performance indices.

- **Free memory**
  The amount of unused memory of the server host.

- **Used memory**
  The amount of used memory of the server system excluding that used by the kernel.

- **The number of processes**
  The total number of processes that stay in the server system.

- **Time stamp**
  The timestamp when the rep2 derives the information from the kernel.

In case of a multiprocessor system, the rep2 can monitor the following performance indices per CPU.

- **CPU idle rate**
  The rate of time when the CPU does not have any tasks.

- **CPU kernel rate**
  The rate of time when the CPU works in privileged (kernel) mode.

- **CPU user rate**
  The rate of time when the CPU works in non-privileged (user) mode.

- **The number of context switches**
  This index is the number of generated context switches in the server system per second. The context switch is a process switch, and if this becomes too high, the server system cannot process the service because of a high load.

- **The number of system calls**
  This is the number of invoked system calls in the server system per second. If the server program invokes a system call, the operating system copies data between the user space and the kernel space. Therefore, if this value becomes larger, the load of the server system becomes larger.
The number of intrrupts
This is the number of generated interrupts in the server system per second. If an interrupt occurs in the system, the CPU executes the interrupt handler routine. Consequently, if this value increase, the server system receives more load.

The number of page faults
This is the number of page faults generated by MMU (Memory Management Unit) on the CPU. If the page fault occurs, the CPU jumps to the routine that prepares free pages and load data to it. So, if this value increase, the server system is running short of memory.

3 Design

In this section, We explain modules to measure some performance indices of a WWW server.

• Packet monitor module
  The packet monitor module observes the packets that convey connection data between the client and the server in real time. This module sends monitored packets to the protocol analysis module that can distinguish special flags of TCP packets.

• Kernel monitor module
  The kernel monitor module extracts several states in the server in real time. This module forward kernel information such as CPU utilization and the number of context switches from the server to an exterior host to generate graphs.

• Protocol analysis module
  The protocol analysis module is one of core modules in our system. A important function which the protocol analysis module provides is to record the time when each packet is observed. Furthermore, the protocol analysis module provides the following functions:
  - By the information in both IP and TCP headers in the packet, the protocol analysis module distinguishes each TCP connection and tracks down its state transitions.
  - The protocol analysis module counts up the number of IP datagrams observed in each TCP connection. The total length of the payload in the IP datagram is also recorded. These datagrams are monitored separately for the upstream (from the client to the server) and the downstream (from the server to the client).
  - The total length of the TCP payload is also recorded separately for the upstream and the downstream.

The protocol analysis module writes these information to a single data file. This data file is used for the further statistical analysis by statistics analysis module. The protocol analysis module analyzer also provides the shared memory for other programs in order to enable them to make a realtime data visualization and analysis.

• External data collection module
  The external data collection module transmits real time data by generating protocol analysis module to an exterior host to analysis them.

• Internal data collection module
  The internal data collection module sends more detail a information in a server system to grasp server behavior to an exterior host.

• Visualization module
  The visualization module forms several graphs such as kernel parameter and request rate by using output results of external/internal data collection modules in real time.

• Statistics analysis module
  The statistics analysis module generates statistics graphs such as cumulative distributed function of connection continuation time and response time by using the output of the protocol analysis module.

4 Implementation

We implement our system as a portable system for several UNIX platforms. Currently, the packet monitor module was implemented in ANSI C using the GNU compiler.
(gcc) on FreeBSD. However, we can make it as a "platform-independent" program with using LBL’s packet capture library. In this section, we provide several technical notes on our implementation of the system.

4.1 Packet monitor module

The packet monitor module in the current implementation uses the LBL’s packet capture library (libpcap [1]) to capture the packets on the network. The library provides a common API for both the Berkeley Packet Filter (BPF [2]) on many BSD variants and the Network Monitoring Protocol (the packet snopper) on Sun’s Solaris and SGI’s IRIX.

4.2 Kernel monitor module

Each operating system has its own internal data set of the kernel. We used KVM interfaces to derive the kernel information for the kernel monitor module. In this reason, we want to reduce kernel dependent part in the kernel monitor module. With the KVM interface, we can use a unified interface to find the kernel information on Solaris and FreeBSD.

However, the KVM interface cannot handle some of the kernel information such as the buffer status of the protocol stack and input/output packets. Therefore, we used some functions dependent on the operating system to get such kernel information.

- Solaris
  In case of Solaris, we used a kstat interface. The kernel monitor module can derive network information through that interface.

- FreeBSD
  In case of FreeBSD, we used a sysctl interface. The kernel monitor module can retrieve the information about the protocol stack through the sysctl interface.

4.3 Protocol analysis module

In the connection analysis module, we had to implement a module to monitor the TCP state transitions of each TCP connection. This module uses the sequence number, acknowledgment number, and flags in TCP headers for tracking down all the state transitions. The algorithm which we use in the connection analysis module is shown in Figure 3.

In this algorithm, the connection analysis module allocates a block of memory for each connection when its SYN packet is observed. This memory block is used as a memo for recording the state for each TCP connection. In the case the TCP connection is shut down normally, the memory block is released and the connection analyzer writes several performance measures listed in the Section 3 to the data file. However, there are several cases where the memory blocks are not released. These cases can happen in the following situations:

- The client tries to establish the TCP connection, however, the server rejects the request. In this case, since the client sends some SYN packets to the server, the connection analyzer allocates a memory block. This situation may cause by lack of system resources (e.g., the listen queue in the socket layer) in the server.

- The server has been clashed during the TCP connection is established. In this case, the client tries to keep the connection so that the client does not send any FIN or RST packets to the server. Therefore, the connection analyzer cannot release the memory block.

- The packet monitor module drops FIN and/or RST messages. In this case, a TCP connection was established but neither FIN nor RST packets are observed

Figure 3. The connection analysis algorithm
on the connection. Therefore, the connection analyzer cannot release the memory block.

In order to handle these cases listed above, the connection analyzer has a garbage collection mechanism for these "unreleased" memory blocks. In the current implementation, the connection analyzer tries to find any memory blocks that were not updated in the last 24 hours.

4.4 External/Internal data collection module

The external/internal data collection modules demand several real time data from the protocol analysis module and the kernel monitor module every 10 seconds. These modules output real time data as test file, and visualization module generate real time graphs by using this text file.

4.5 Visualization module

The visualization module is a program to display several data about server system such as kernel parameters, request/response rates and so on. We show an example of this module in figure 4.

4.6 Statistics analysis module

The statistics module analyzes a log file that protocol analysis module outputs, and aggregates frequency distribution of connection continuation time and response time. Figure 5 shows an example of cumulative distributed function of response time.

5 Experiment

To show that the our system is effective, we give an example to apply them to the large scale actual operation server. We observe WWW servers using broadcasting over the Internet about the 82th National High-School Baseball Games of Japan. In these servers, we had the maximum access about 46 million per day.

5.1 System composition

Figure 6 shows network environment in this experiment. As shown figure 6, we prepare four server machines and a layer 4 switch of Foundary Inc. on the up link. The layer 4 switch applies round robin method and distributes access load from clients between the servers.

WWW server application and its hardware spec are shown in table 1. WWW1 uses apache server system, from WWW2-4 use Chamomile [4] server system that is developed originally for the Koshein project.

The Chamomile system adopts single-process and multi-thread architecture. One of the design goals of Chamomile is to reduce the overhead of the context switch.

Furthermore, in this experiment, we perform observation
outside the server using our system in figure 6. The observation data is sent to the visualization host through control line, and visualize the data in real time.

5.2 Result

In this subsection, we present the result of analysis about the greatest access concentration period (August 21, 14:45-15:45).

5.2.1 Result of packet monitoring

In figure 7, the total number of concurrent connection of the four server hosts is about 60,000 in peak hours. The total peak rate connection arrival and connection-processed are about 1,800 requests / second. The peak traffic is about 70Mbps / second.

Difference of the connection arrival rate and the connection processed rate is quite small because the server system is not over-loaded and the requests are processed very quickly.

5.2.2 Result of kernel monitoring

- The number of processes

Figure 8 shows time transition of the number of processes of Apache [3] and Chamomile. The Apache employs multi-process architecture so that the number of processes goes up and down around 3000. However, Chamomile is a single process so that the number of process is stable.

If the number of process increases, free memory is consumed by growing process control blocks in the kernel on the server system.

- Relationship between the amount of free memory and the number of page faults

Figure 9 shows relationship between the amounts of free memory and the number of page faults on Apache system. As shown in figure 9, when free memory is decreasing, the number of page faults spikes occasionally.

The reason of these page faults is to search unused memory in the server system. Therefore, when free page allocation triggered by page faults cannot be satisfied in time, the server system suffers performance degradation.

- Context switch

The number of context switches on WWW1 and WWW2 is shown in figure 10. WWW1 executes more context switches than WWW2 because the number of processes on WWW1 is larger than WWW2.

<table>
<thead>
<tr>
<th>Table 1. System Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>OS</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Application</td>
</tr>
</tbody>
</table>
Figure 7. The result of total 4 servers

Figure 10. Time transition of the number of context switches

Figure 11. Time transition of the number of system calls

- System call

Figure 11 shows time transition of the number of system call invocations on WWW1 and WWW2. As shown in figure 11, WWW2 invokes a little more system calls than WWW1. The Chamomile on WWW2 uses poll() system call, and Apache on WWW1 uses select() system call. So, difference of the number of system call is caused by the difference of the poll and select system calls.

- CPU utilization

CPU utilization of WWW1 and WWW2 is shown in figure 12. The CPU kernel rate is almost same on WWW1 and WWW2. However, the CPU user rate of WWW2 is larger than WWW1. According to the reference [5], the server system is saturated when the CPU kernel rate becomes over 90%. Therefore, the WWW1 and the WWW2 are not saturated and can process more requests from the clients.
5.2.3 Consideration

If the server administrators take notice the relationship between the amount of free memory and the number of page faults, they can prevent saturated state by reducing the amount of memory consumed by the server processes.

While, if the request rate increases a little at WWW1, WWW1 may be saturated. WWW1 did not have enough free memory so that memory allocation invokes a large number of page faults. If the request rate of WWW1 increases, the allocation of free memory through page faults is not satisfied in time, and the server system might get in a slashing state.

To prevent this state, the administrator should restrain page faults in the server system. In case of WWW1 using Apache system, the administrator should configure Apache to keep a constant number of server processes because each process creation through fork() system call consumes not a small amount of memory.

6 Conclusion

Server administrators wanted to detect whether a server system is overloaded with requests or not in real time. They also wanted to understand a reason of troubles in the server. Furthermore, they wanted to be managed in a centralized manner, instead of a wide area distributed server.

Therefore, we developed a new system to help the administrators to manage a distributed server system in a centralized manner. The administrators with our system could understand the server status and detect whether the system is overloaded with requests or not in real time. That would be also able to reduce the cost related to the management.

Acknowledgement

This work was supported in part by Research for the Future Program of Japan Society for the Promotion of Science under the Project "Integrated Network Architecture for Advanced Multimedia Application Systems" (JSPS-RFTF97R16301).

References


