Evaluation of Load Balancing in Multicore Processor for *AnT*

Takeshi Sakoda, Toshihiro Yamauchi, Hideo Taniguchi
Graduate School of Natural Science and Technology
Okayama University
Okayama, Japan
sakoda@swlab.cs.okayama-u.ac.jp, {yamauchi, tani}@cs.okayama-u.ac.jp

***Abstract*—Operating systems (OSes) that is based on microkernel architecture have high adaptability and toughness. In addition, multicore processors have been developed along with the progress of LSI technology. By running a microkernel OS on a multicore processor and distributing the OS server to multiple cores, it is possible to realize load balancing of the OS processing. In this method, transaction processing, which requires a large amount of OS processing, can be provided effectively in a multicore environment. This paper presents evaluations of distributed OS processing performances for various scenarios for *AnT* operating system that is based on the microkernel architecture in a multicore environment. In these evaluations, we describe the differences in performance by distribution forms when referring the data in a block. Moreover, we use the PostMark and Bonnie benchmark tools to evaluate the effects of load balancing for the distribution forms.**

***Keywords*-operating system; multicore processor; microkernel; distributing process**

I. INTRODUCTION

Microkernels[1, 2, 3] have an Operating System (OS) architecture which implements basic OS functions that are represented by the scheduler as a kernel. It also has an OS architecture which implements the file management function and many types of driver functions as some processes (OS server). In addition, it is possible to construct a distributed processing system that can distribute the OS processing by distributing the OS server.

Further, multicore processors[4] that have multiple instruction execution units on one processor have been developed along with the progress of LSI technology. By running a microkernel OS on a multicore processor and distributing the OS server to multiple cores, it is possible to realize the load balancing of OS processing. In this method, the transaction processing, which places a heavy load on the OS processing, can be provided effectively in a multicore environment.

In this paper, we evaluate and report the result for distributing of OS processing about ***AnT*** OS[5] that is based on microkernel in a multicore environment.

II. *ANT* OS FOR MULTICORE ENVIRONMENT

A. Basic Architecture

Figure 1 shows the basic architecture of ***AnT*** OS for a multicore environment (multicore ***AnT***). The OS consists of an internal core and an external core that run as a process. The internal core has a m-kernel that is launched first and runs on a Boot Strap Processor (BSP). It also has multiple p-kernels that are launched by m-kernel and run on Application Processors (AP). The m-kernel has all of the functions that the kernel needs. However, to keep the kernel lightweight design, the p-kernels have only three types of functions, which are the trap and interrupt function, the inter-server communication function, and the schedule function. The external core consists of program that is required by the adapted system. For example, the external core provides the file management function and the disk driver function as the OS server.

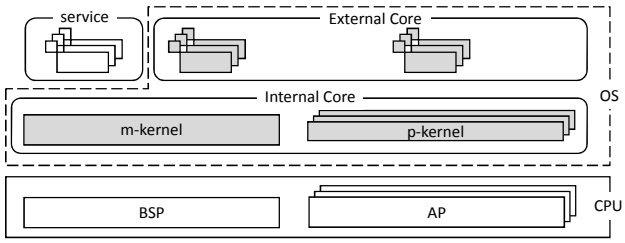


Figure 1. Basic architecture of multicore ***AnT***.

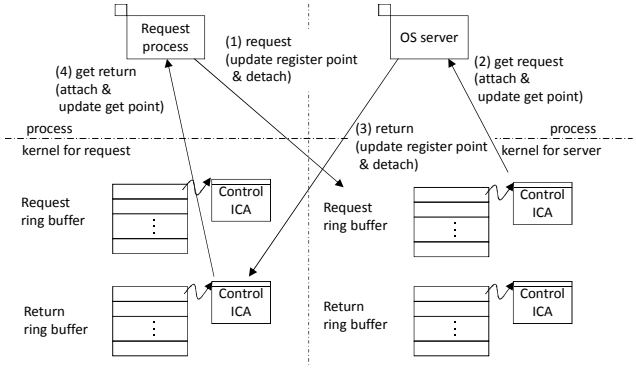


Figure 2. Flow of ISPC for multicore ***AnT***.

The virtual space in ***AnT*** consists of multiple virtual storage. In addition, ***AnT*** has an Inter-core Communication Area (ICA). This area is a space that is used by the internal core, the external core, and the service for data communication.

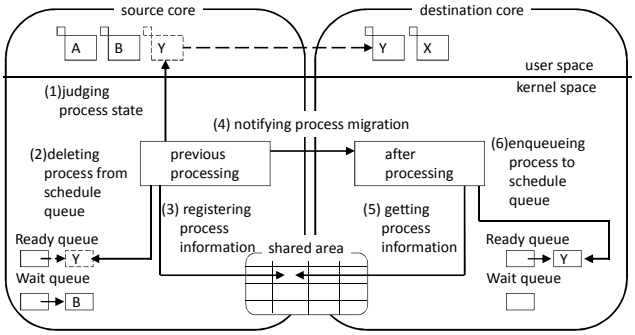


Figure 3. Flow of process migration.

B. *Inter Server Program Communication*

AnT has fast Inter Server Program Communication (ISPC). In addition, multicore ***AnT*** implements fast ISPC for a multicore environment. Figure 2 shows the flow of fast ISPC for multicore processors. Specifically, we limit the number of communication partners between OS servers and implement two exclusive controls in queue operation during communication. Additionally, to use the ring buffer control architecture, we implement detachment and attachment of the ICA for control (control ICA) using non-exclusive control. We improve the transfer method for ICA. Specifically, the kernels on the each core detach and attach the control ICA during ISPC.

C. *Process Distribution Mechanism*

Multicore ***AnT*** has a process migration function that is adapted to multicore environments. The process migration mechanism is a function that migrates a process that is running on a core (source core) to another arbitrary core (destination core). Figure 3 shows the flow of process migration. The procedure is described below.

(1) Based on the process state, a judgment is made about whether it is possible to perform a migration.
(2) When the process that is migrated is a RUN state, the process state is updated to a READY state and the context of that process is stored. When the process that is migrated is a READY or WAIT state, the process is deleted from each schedule queue.
(3) The process information of the process that is migrated is written to the shared area.
(4) A notification about the process migration is sent to the destination core using an Inter-Processor Interrupt (IPI).
(5) The process information for the process that is migrated is read from shared area.
(6) The process is queued up to correspond to the schedule queue because the state of all processes was changed to a READY or WAIT state in step (2) above.

We implement the method that the process that is distributed uses OS processing of other cores. This method is implemented as sending request to m-kernel and sharing unique functions of m-kernel. Additionally, we adapt these methods based on the processing time for the request.

The sending request to m-kernel is to send request of the kernel call processing issued by a process that is running on a p-kernel to m-kernel. The sharing unique functions of m-kernel is able to use unique functions of m-kernel from a process that is running on a p-kernel. We use the exclusive control for sharing unique function of m-kernel.

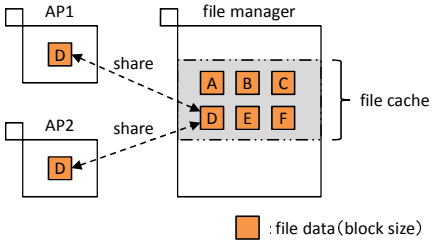


Figure 4. Basic method of the OMF.

D. *On Memory File Mechanism*

The On Memory File (OMF) mechanism is based on the following approach.

(1) The OMF shares a file cache with a process.
(2) The OMF can extend or reduce a file size.
(3) The OMF can update the referential date and the updated date of a file.

Figure 4 shows the basic method of the OMF. The OMF shares a file cache with a process. The file data of the process is referred or updated after the physical memory that stores the file data has been mapped in the virtual memory space. Therefore, unlike general Input/Output (I/O) functions, the OMF does not copy the data between the file cache and the virtual memory space of the process. The OMF manages the file data using a block (4 KB).

In contrast to the memory mapped file function[6], the OMF can extend or reduce the file size. Specifically, to extend or reduce the file size, the OMF updates the file size in the i-node which manages the file information and the storage location of the file data on the external storage unit. In addition, the OMF can update the referential date and the updated date of the file. To update the referential date and the updated date, the OMF updates them on the i-node. Updating them i-nodes is executed when writing file data to external storage unit.

III. EVALUATION

A. *Point of View and Environment for Evaluation*

Load balancing of the I/O processing for the OS is effective for obtaining a higher throughput because most of the transaction processing consists of OS processing. In this paper, we elucidate the effect of load balancing on OS processing. We focus specifically on file I/O processing.

In microkernel OSes, many OS servers run on the same core. Therefore, differences between the priorities for processes have effects on the performance. To clarify these differences, we evaluate the basic performance for the file I/O.

TABLE I. VIEW OF PROCESS DISTRIBUTION FORM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
BSP	A,F,B,D	A,F,B	A,F,D	A,B,D	A,F	A,B	A,D	A	A,F	A,B	A,D	A	A	A	A
AP1		D	B	F	B,D	F,D	F,B	F,B,D	B	F	F	F,B	F,D	B,D	F
AP2									D	D	B	D	B	F	B
AP3															D

*A: benchmark process, F: file management server, B: block management server, D: disk driver server

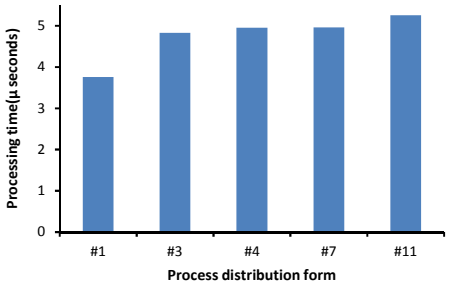


Figure 5. Basic performance.

Next, if the OS processing is distributed, the overhead for ISPC which includes IPI becomes a problem. To clarify this overhead, we evaluate it using the PostMark[7] benchmark. PostMark is a benchmark program that a single process executes the number of times specified opening files, referring the sequential data, and closing files on multiple files. In addition, in order to clarify the effectiveness of load balancing for OS processing, we perform an evaluation using a Bonnie[8] benchmark. Bonnie is a benchmark program that multiple processes refer random positions in an input file and perform updates based on any probability.

Moreover, we clarify the effect of differences in OS structure by running the benchmark programs on ***AnT*** and FreeBSD 6.3-RELEASE and comparing the results.

Table 1 shows the process distribution form that includes the OS server and the benchmark process. There are three types of OS servers that are related to the file I/O: the file management server, block management server, and disk driver server. The file management server manages the i-nodes. The block management server manages the cache of file data (file cache) and reduces the number of I/O operations. The disk driver server controls the disk I/O devices and inputs and outputs the file data. In Table 1, #1 is the case where the process is not distributed. #2 to #8 are the cases where the process is distributed between two cores. #9 to #14 are the cases where the process is distributed between three cores. #15 is the case where the process is distributed between four cores.

We evaluated all cases in Table 1 on the ***AnT*** OS using a computer that has an Intel® Core™ i7-2600 Processor (3.4 GHz). We set a high priority on the order of the disk driver server, the file management server, the block management server, and the benchmark process. The file management server and the block management server have the same priority.

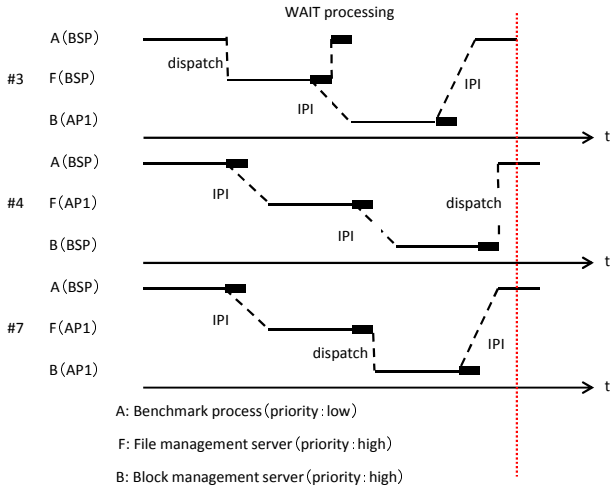


Figure 6. Situation of running process.

B. *Basic Performance*

Figure 5 shows processing times taken by a process to refer the file data of one block (4 KB). In these cases, the file data exists on the file cache. An outline of the read processing from the file cache is given below. First, the benchmark process sends a request to the file management server. Next, the file management server sends a request to the block management server. The block management server gets the data from the file cache. Finally, the block management server returns the result to the benchmark process. None of the requests involves the disk driver server. Therefore, the disk driver server does not run. Therefore, we focus on the distribution form for the file management server and the block management server in Figure 5. From this figure, we confirmed the following conclusions.

1) Processing time increases as the number of cores which the OS processing is distributed increases. As the number of cores increases, the OS processing becomes more distributed. This causes the number of inter-core communications to increase.

2) Distribution forms that distribute the OS processing to two cores (#3, #4, and #7) are same processing times because the number of inter-core communications is the same. However, the processing time for the distribution form that distributes the block management server (#3) is a little shorter than that of the others (#4 , #7). This result is

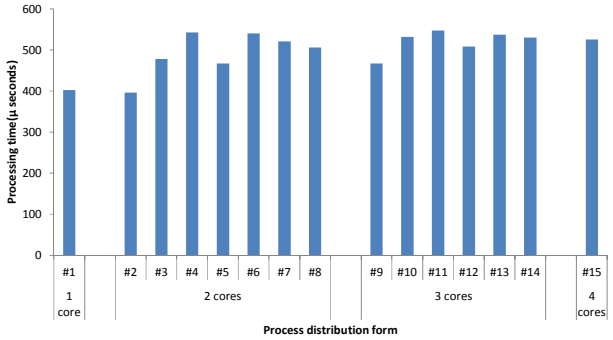


Figure 7. Results of PostMark.

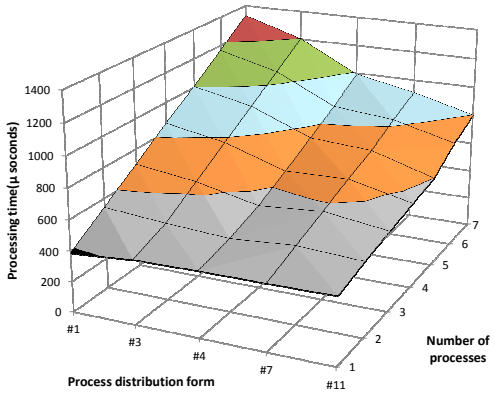


Figure 8. Result of Bonnie.

caused by the WAIT processing that is performed by ISPC. Figure 6 shows the active statuses of processes in #3, #4, and #7. Because the priority of a requesting process (benchmark process) is lower than that of a requested process (file management server) in #3, the WAIT processing for a requesting process does not affect the processing time if the WAIT processing time is shorter than the inter-core communication time. On the other hand, because the priority of a requesting process is equal to or higher than that of a requested process in #4 and #7, the WAIT processing for a requesting process is added onto the processing time. Therefore, the processing time of #3 is shorter only the WAIT processing time (0.1 microseconds) than the processing times of #4 and #7.

C. *PostMark*

We used the following parameters for the PostMark measurements. The number of files was 25, the number of execution times was 25 times, and the file size was 500 to 10000 bytes. Figure 7 shows the results. From this figure, we confirmed the following conclusions.

1) The processing times of scenarios where the disk driver server was distributed is equal to those of the scenarios where the disk driver server was not distributed. This is because the block management server does not send requests to the disk driver server. All of the file data exists on the file cache when creating files because the number of files is a few and the total of the file sizes are small. Therefore, if the distribution forms of the file management server and the block management server are the same, the processing time is equal, regardless of the distribution form of the disk driver server. For example, the processing time when distributing to four cores (#15) is equal to distributing to three cores (#11, #13, and #14).

2) If the benchmark process and the file management server are distributed to the same core, then the processing time is shorter than that when both processes are distributed to different cores. This is because inter-core communications are very limited when files open and close. Specifically, inter-core communications occur twice when a file opens and zero times when a file closes while distributing to the same core (#3, #5, and #9). Inter-core communications occur four times when a file opens and twice when a file closes while distributing to different cores (#4, #6, and #10). Inter-core communications occur twice when a file opens and twice when a file closes when distributing to different cores (#7, #8, and #12).

D. *Bonnie*

We used the following parameters for the Bonnie measurement. The number of processes was one to seven, the file size was 128 KB, the number of references was 4000, and the probability for an update to occur was 10%. Because the file sizes were small, as they also were in section III-C, all of the file data existed on the file cache. Therefore, requests to the disk driver server were limited to the beginning of the test. Accordingly, for these measurements, we ignored the distribution of the disk driver server. Specifically, we measured #1, #3, #4, #7, and #11 in Table 1. The results are shown in Figure 8. Figures 9 and 10 provide additional analyses of the information in Figure 8. Figure 9 shows the relationships between the benchmark process and the processing time. From this figure, we confirmed the following conclusions.

1) The effects of load balancing for the OS processings become larger when there are many benchmark processes. In each case, the processing time increases as the number of benchmark process increases. However, the rate of the increase declines as the number of cores increases. This result is clear when comparing #11 (three cores) with #1 (one core). Specifically, for #1, the processing time increases about 155 microseconds as the benchmark process increases. On the other hand, for #11, the processing time increases only about 50 microseconds as the benchmark process increases. However, the rate of the increases is large when the number of benchmark processes is six or more. This is due to the increasing demands on the file management server.

2) Distributing the file management server to other cores is effective. We focus on #3 and #4. Two cores are

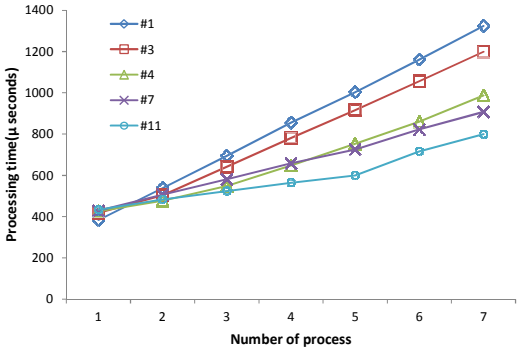


Figure 9. Relationship between number of processes and processing time.

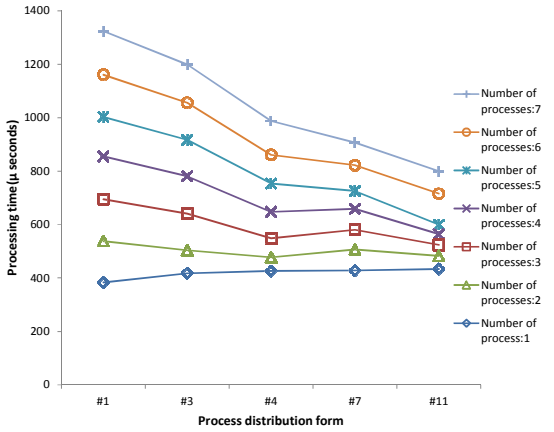
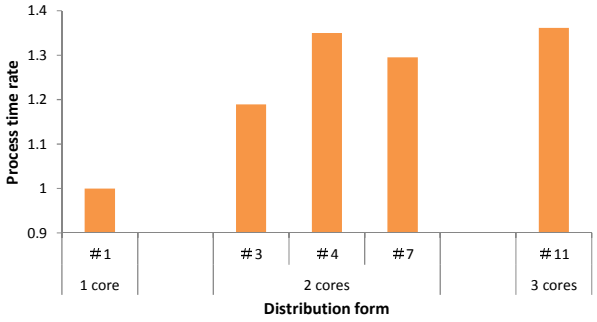


Figure 10. Relationship between process distribution form and processing time.

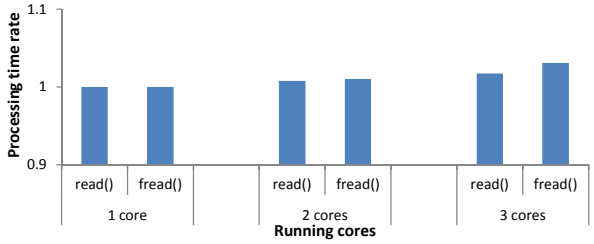
used in each cases. The difference between #3 and #4 is due to the distributing the block management server or the file management server to another core. Distributing the file management server to another other core is effective because the processing time for #4 is shorter than for #3. This is because the file management server requires more processing time than the block management server. Specifically, for #3, the processing time increases about 127 microseconds for the benchmark process increases. On the other hand, for #4, the processing time increases about 86 microseconds as the benchmark process increases.

3) Distributing the OS processing to the other core is effective. We focus on #4 and #7. Two cores are used in each cases. In scenario #7, the file management server and the block management server are both distributed to other cores. This seems to be effective when the number of benchmark processes increases. Therefore, environments which execute many benchmark processes are efficient.

Figure 10 shows the relationship between the process distribution form and the processing time. From this figure, we confirmed the following conclusion.



(a) ***AnT***



(b) FreeBSD

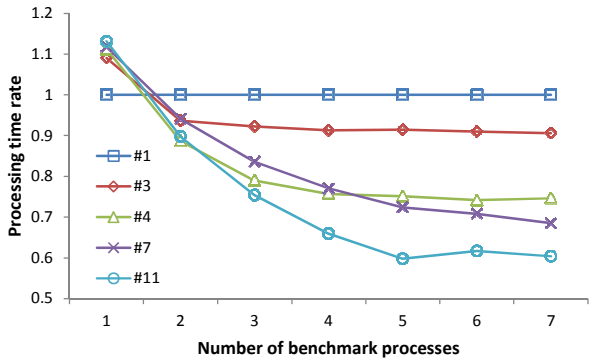
Figure 11. Rate to processing time in using 1 core for PostMark.

4) The number of cores causes significant differences in the process distribution form (one core (#1), two cores (#3, #4, and #7), and three cores (#11)). The overhead from inter-core communications causes the processing times to increase because the effect of load balancing is nothing if the number of benchmark processes is only one. On the other hand, distributing the OS processing is much more effective when there are many benchmark processes running. The effectiveness causes difference by processing of server that implement OS processing so that someone can understand comparing #4 with #7.

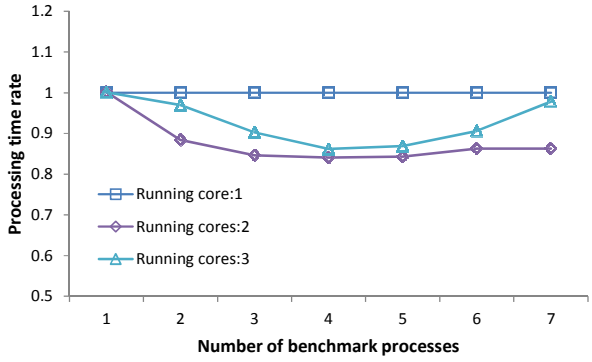
*E. Comparing **AnT** with FreeBSD*

We compared ***AnT*** with FreeBSD. The distribution forms for ***AnT*** are #1, #3, #4, #7, and #11. For FreeBSD the number of cores varied from one to three. Figure 11 and 12 show the rate of processing time when using one core. ***AnT*** was different from FreeBSD when using one core, two cores, and three cores. Specifically, because of differences in the distribution of the OS server there are three different scenarios for ***AnT*** for distributing the OS server to two cores. In FreeBSD, there are two types of file reading interfaces: read() and fread(). From these figures, we confirmed the following conclusions.

*1) In PostMark, the effect of load balancing of OS processing for **AnT** is nothing.* Inter-core communications overhead causes increases in the processing time for ***AnT*** due to the distribution of the OS processing and the fact that the number of benchmark processes is one in PostMark. On the other hand, the inter-core communications do not increase when using FreeBSD because FreeBSD has a



(a) ***AnT***



(b) FreeBSD

Figure 12. Rate to processing time in useing 1 core for Bonnie.

monolithic kernel and as a result there is no need to distribute the OS processing, regardless of the number of cores. Specifically, in ***AnT***, the processing time increases about 1.2 to 1.35 times when distributing to two cores, and 1.35 times when distributing to three cores. On the other hand, in FreeBSD, the processing time increases only about 1 to 1.01 times when running on two cores and 1.01 to 1.03 times when running on three cores.

*2) In Bonnie, the effect of load balancing for the OS processing for **AnT** is high.* When the number of benchmark processes is many for Bonnie, the processing time decreases in ***AnT*** due to the distribution of the OS processing. On the other hand, the processing time does not decrease in FreeBSD even when the number of cores increases. This is because the OS processing is a bottleneck in FreeBSD when there are numerous requests from benchmark processes. For example, if the number of benchmark processes is five, ***AnT***, which distributes to three cores, is possible to reduce processing time to 60%. On the other hand, FreeBSD, which also runs on three cores, is only able to reduce the processing time to 87%.

IV. CONCLUSION

We evaluated the effect of load balancing for file I/O process in the OS for ***AnT*** OS that runs on multicore processors.

In the evaluation of basic performances, we showed that the processing times increase as the inter-core communications increase. This is due to the distribution of the OS processing. Differences in the priorities of processes running on the same core also affect the processing times.

In the evaluation of the PostMark benchmark program, where the number of benchmark processes is one, we showed that there is no effect from load balancing for the disk driver server if all of the file data exists on the file cache. In addition, we showed that the processing time decreases when the benchmark process and the file management server are run on the same core.

In the evaluation of the Bonnie benchmark program, where the number of benchmark processes is one to seven, we showed the effect of load balancing is high in environments where several benchmark processes are running and numerous requests for benchmark processes are executed.

Finally, we measured PostMark and Bonnie on FreeBSD, and compared ***AnT*** with FreeBSD. For PostMark, we showed that processing time increases in ***AnT*** because the inter-core communication overhead increases due to the distribution of the OS processing. On the other hand, the processing time in FreeBSD increases very little because there is no need to distribute the OS processing. For Bonnie, we showed that it is possible to reduce processing time to 60% in ***AnT***, because it is possible to distribute the OS processing. On the other hand, With the FreeBSD, it is only possible to reduce the processing time to 87%, because OS processing is a CPU bottleneck.

I/O operations over networks are a remaining topic for further study.

ACKNOWLEDGMENT

This work was supported by JSPS Grant-in-Aid for Scientific Research (B) Number 24300008.

REFERENCES

[1] J. Liedtke, “Toward real microkernels,” Communications of the ACM, Vol.39, No.9, pp.70-77, 1996.

[2] A.S. Tanenbaum, J.N. Herder, and H. Bos, “Can we make operating systems reliable and secure?,” IEEE Computer Magazine, Vol.39, No.5, pp.44-51, 2006.

[3] D.L. Black, D.B. Golub, D.P. Julin, R.F. Rashid, R.P. Draves, R.W. Dean, A. Forin, J. Barrera, H. Tokuda, G. Malan, and D. Bohman, “Microkernel operating system architecture and mach,” Journal of Information Processing, Vol.14, No.4, pp.442-453, 1992.

[4] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, “Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge,” IEEE Micro, Vol.32, No.2. pp.20-27, 2012.

[5] ***AnT*** Group, “***AnT*** project,” 2005; http://www.swlab.cs.okayama-u.ac.jp/lab/tani/research/AnT/index.html.

[6] B.O. Gallmeister, “POSIX.4,” O’Reilly, pp. 128-129 and 389-391, 1995.

[7] J. Katcher, “PostMark: A New File System Benchmark, ” Technical Report TR3022, Network Appliance, 1997.

[8] T. Bray, “The Bonnie home page,” 1996; http://www.textuality.com/bonnie/