

國立交通大學

資訊科學系

碩士論文



利用遠端儲存日誌來提昇日誌式檔案系統之效能

Improving Performance of Journal File Systems by Remote Journaling

研究生：黃亭彰

指導教授：張瑞川 教授

中華民國九十四年六月

利用遠端儲存日誌來提昇日誌式檔案系統之效能
Improving Performance of Journal File Systems by Remote Journaling

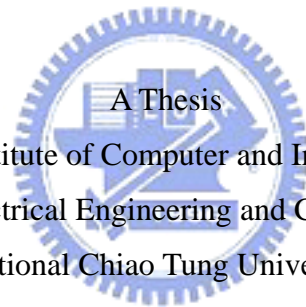
研究生：黃亭彰

Student：Ting-Chang Huang

指導教授：張瑞川

Advisor：Ruei-Chuan Chang

國立交通大學
資訊科學系
碩士論文



Submitted to Institute of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

利用遠端儲存日誌來提昇日誌式檔案系統之效能

研究生：黃亭彰 指導教授：張瑞川 教授
國立交通大學資訊科學研究所

摘要

日誌式檔案系統是一個常用的用來增加檔案系統效能與可靠性的技術，目前有很多檔案系統採用這樣的技術，然而採用這種技術的代價是需要額外的空間來儲存檔案系統日誌。此外，因為紀錄日誌引起的額外的磁碟活動也會導致效能降低。根據這篇論文的實驗顯示，不同周期的儲存日誌活動會導致不同的效能降低，最多可達 27% 左右。



在這一篇文章中，我們利用遠端機器透過網路來儲存日誌的方法，以提升因額外的磁碟活動而導致的效能低落。因為網路傳送資料的速度較快且不影響檔案系統之效能，幾乎可以達到極低週期之儲存日誌活動，這提供了幾乎相同的檔案系統一致性(file system consistency)保證，以及相對於傳統日誌式檔案系統效能提昇。

在實驗結果中，利用遠端儲存檔案系統日誌的方法，可以提升檔案系統效能大約 10%~21% 左右。對 100Mb/s 的 Ethernet 而言，每秒佔用的網路頻寬大約是 0.8% 到 5% 左右，而檔案系統的回復速度則與傳統的日誌式檔案系統差不多。

Improving Performance of Journal File Systems by Remote Journaling

Student: Huang-Ting Chang

Advisor: Dr. Ruei-Chuan Chang

Department of Computer and Information Science
National Chiao-Tung University

ABSTRACT

Metadata journaling is a matured technique for improving performance and reducing consistency checking time of file systems. Many file system adopts this scheme for improving performance and file system consistency. However, the overhead brings by metadata journaling is extra space and performance degrade caused by frequent journal data flush. According our experiment, different commit interval and journal mode result in different performance degrade to 27%.

In this paper, we improve performance of journal file system by remote journal scheme. The remote journal scheme removes the frequent journal I/O activities form local disk to remote server. Because network transfer is faster and will not effect file system performance, we can have better throughput and the same consistency semantic.

According to the experiments in this paper, remote journal increases about 10% to 21% performance, but the penalty is light. Although remote journal does need more CPU time for network transfer, the overhead is less than 10%. And the extra network bandwidth taken by remote journal is less than 7% in metadata bound workload.



Acknowledgements

I am so grateful to have much guidance from my advisor Professor R. C. Chang. He taught me the essential of research, guided me the way of thinking. I also very appreciate Dr. Da-Wei Chang. He advised me so that I can finish my thesis.

Besides, thanks to each member of the computer system laboratory for their encouragement and kindly help. I would like to thank my parents for their unlimited love. Finally, I thank all friends for all the joyous things that inspire my life.

Ting-Chang Huang

Department of Computer and Information Science

National Chiao Tung University



2005/6

TABLE OF CONTENTS

摘要	i
ABSTRACT	ii
Acknowledgements	iv
TABLE OF CONTENTS.....	v
LIST OF FIGURES	vii
LIST OF TABLES	vii
Chapter 1 Introduction.....	1
Chapter 2 Backgrounds	4
2.1 Metadata consistency in file systems.....	4
2.2 Metadata journaling.....	5
2.3 Write-ahead file system (WAES).....	6
2.4 Linux Ext3 file system.....	6
Chapter 3 Related work	8
3.1 Soft update.....	8
3.2 Non-volatile RAM (NVRAM)	8
3.3 Log-structured file system (LFS)	9
Chapter 4 Journaling analyses	11
4.4 The Effect of Commit Interval	14
4.5 Observation.....	15
Chapter 5 Remote Journaling	16
5.1 Concept of Remote Journaling	16
5.2 Applications	17
5.3 Implementations	18
Chapter 6 Experiments result	20
6.1 Performance comparisons.....	20

6.2 CPU utilities of remote journal.....	21
6.3 Network bandwidth taken by remote journal	23
Chapter 7 Conclusion and future works	25
Reference	28



LIST OF FIGURES

Figure 1 Performance comparisons of journal and non-journal Ext3 file system.....	13
Figure 2 Performance Comparisons of Different Commit Intervals	14
Figure 3 the example application.....	17
Figure 4 performance of remote journal.....	21
Figure 5 CPU utilities of writeback mode	21
Figure 6 CPU utilities of ordered mode	22
Figure 7 CPU utilities of journal mode	22
Figure 8 CPU time comparison	23
Figure 9 Network usage comparison	24



LIST OF TABLES

Table 1 Three journal type comparison	2
Table 2 Experimental setup	11

Chapter 1

Introduction

Metadata update problem is an important issue in recent file system design. Asynchronous metadata write leads to better throughput, but it can not guarantee file system consistency after a file system crash. On the contrary, synchronous metadata write with update dependency guarantees file system consistency after crash. However, it causes a large synchronous seek time, which results in poor performance.

Metadata journaling [2][7][9] is a matured technique for improving performance and reducing consistency checking time of file systems. Delayed-write scheme and write-ahead logging (WAL) protocol [7] are vital features of metadata journaling. The former reduces unnecessary synchronous metadata writes for consistency while the latter decreases the time for consistency checking after a file system crash [3].

The write-ahead logging (WAL) protocol [7] in metadata journaling technique means: (1) must force the log records for the metadata of corresponding data write before the regular data is updated to storage and (2) all log records must be flushed before a file system operation commit. The first rule implies the operation atomicity and the second rule implies durability.

Traditional file systems employ a whole-disk scanning approach which checks all data structures in the disk when an unclean mount is found. The scanning time grows with the number of files and directories, and thus it is not suitable for current file systems that based on large capacity disks. By grouping ordered log records of metadata and setting checkpoints, journal file systems enable roll-back and roll-forward of logged metadata when they perform

consistency checking[3]. Thus, whole-disk scanning is not necessary and file system recovery time is much shorter.

Typically, logged metadata can be placed in three places: a log file in a disk, another partition in a disk, and another disk. As table 1 indicates, all of those have different overheads, such as performance drops or extra disks cost [11].

	Disk space cost	Performance drop
Log file	can be dynamically change	extra I/O traffic in the same disk
Another partition	can not be dynamically change	extra I/O traffic in the same disk
Another disk	need more disks	better than two above

Table 1 Three journal type comparison

[journal data in extra isolated disk outperforms journal data in the same disk, but cost of an extra disk is much higher.]

Log file and another partition scheme in table 1 do not need an extra disk but involve more I/O traffic of metadata journaling. The previous research mentioned that I/O traffic of metadata journaling reduces performance. On the other hand, another disk scheme prevents the overhead brings by journaling I/O, but extra disks make more cost.

In this paper, we propose remote journal scheme which transfers journal data to remote journal server by network. Remote journal scheme reduces I/O traffic of metadata journaling and has low cost related to another disk scheme. The reason is that I/O traffic of metadata

journaling is switched to network and cost of remote journal server can be shared. The data of metadata journaling will not affect the network usage because the data of metadata journaling is much lower than network bandwidth. This is useful under some metadata bound workload, like online transaction processing (OLTP), news server, and web server.

We modified JBD (Journal Block Device) of EXT3, a popular file system used in Linux. JBD takes responsibility of flushing journal data. Our experiments indicate that remote journaling improves the performance of Ext3 by 10%~27% and only causes little network overhead (i.e., 0.7%~3% for a fast Ethernet link).




Chapter 2

Backgrounds

2.1 Metadata consistency in file systems

Metadata, such as inode, directory entry, and superblock, is used by a file system to manage regular data. When metadata corrupts or has some incorrect pointers, the file system becomes inconsistent and the users may not correctly access the data they need.

In order to prevent file system inconsistency, traditional file system follows the following update dependency rules to handle file system operations:

- 
- (1) Never set a new pointer to a uninitialized data
 - (2) Reallocate data before guaranteeing no pointer point it
 - (3) Keep the old properties before new properties are written

Dangling pointer can be avoided by rule 1 and rule 2 guarantees data blocks indexed by pointer are correct. And file system will not lose file system properties by rule 3.

File systems which follows the above rules have good consistency, but unfortunately, following the update dependency rules leads to performance degradation because synchronous writes cause a huge number of seeks for satisfying the above dependency constraint rules. According to previous study, such performance degradation can be larger than 20% to 25%.

Thus following file systems choose not to follow such rules so that they can get better performance. However, they have no guarantee about file system consistency after an unclean mount or a system crash. Therefore, they need a whole-disk scan if inconsistency happens. A whole-disk scan checks all metadata in the disk, and the time it needs grows with the number of files and directories. As disk capacity becomes larger, a whole-disk scanning becomes unacceptable because it increases downtime.

2.2 Metadata journaling

Metadata journaling employs additional space for storing metadata log records that can be metadata itself or operation on metadata. By following WAL protocol, information of metadata log records helps fast recovery to consistent state by replaying the log records. The write-ahead logging (WAL) protocol [7] in metadata journaling technique means: (1) must force the log records for the metadata of corresponding data write before the regular data is updated to storage and (2) all log records must be flushed before a file system operation commit. The first rule implies the operation atomicity and the second rule implies durability.

When mounting a journal file system, the superblock is examined first to see if the unclean mark exists or not. If it exists, a recovery process starts. Roll-forward or roll-back depends on if the recently used checkpoint is found in journal area or not. When file system crash happens, recovery utility does roll-forward if journal corrupt before the checkpoint and does roll-back if journal corrupt after the checkpoint.

2.3 Write-ahead file system (WAFS)

Write-ahead file system (WAFS) is a simple file system designed for storing file system journaling data. A WAFS can be mounted as synchronous or asynchronous I/O mode. A WAFS can also be placed on different disks and thus results in better performance because of I/O traffic independence.

The merit of WAFS is flexibility in configuration. Most read/write patterns of WAFS are similar, because most read request occurs only when recovery time. Therefore the administrators can optimize the layout and file system arguments according their needs.

2.4 Linux Ext3 file system



The Linux Ext3 file system extends the Ext2 file system with metadata journaling capability. With metadata journaling method, metadata is appended to a write-ahead log before metadata and then regular data are flushed to disk. After an unclean system shutdown, the information in the write-ahead log decreases time of file system recovery to insistent state.

Because of compatibility, Ext3 data structures on disk are largely similar to those in Ext2. Thus an Ext2 file system can be easily mounted as an Ext3 file system, and vice versa. And unlike other journal file system, Ext3 saves metadata itself as journal log record rather than metadata operation because of compatibility.

Ext3 logs metadata (and even data) operations in a fixed size and circular buffer based

file, and it uses *journaling block device* (JBD) to handle the journal affair. In addition, Ext3 has the following modes that allow the administrator to choose the metadata and data journaling method: (1) journal mode (2) writeback mode, (3) ordered mode.

In journal mode, all updated data and metadata are written twice. This is the slowest but safest mode because full data logging reduces the chances of data losing.

In ordered mode, only updated metadata are flushed to the journal file. Ordered means that regular data writes must be done before metadata writes, which provides more consistency. Ordered mode guarantees that both regular data and metadata are consistent after recovery from an unclean system shutdown

In writeback mode, only updated metadata are flushed to the journal file and the metadata/data writes are not needed to be ordered. This is the fastest among the three modes. Writeback mode just guarantees consistency of file system metadata

Ext3 utilizes a kernel daemon, called *kjournald*, to flush journal data periodically. The interval between successive flushes must be carefully decided because of balance between consistency and performance. A larger interval results in better performance but weaker consistency, while a smaller interval leads to stronger consistency but worse performance (due to extra small I/O operations).

Chapter 3

Related work

In this chapter, we discuss other schemes for improving file system consistency and performance. They are soft update, NVRam, and log-structured file system.

3.1 Soft update

Besides metadata journaling, another well-known scheme for improving asynchronous write file system is Soft update [4] [5] [6] [13]. Soft update is originally used to improve FFS [14] consistency by guaranteeing dependency ordering of metadata write. Soft update maintains fine-grained (i.e., field-based) dependency information. Like metadata journaling, soft update employs delay writes for performance consideration. When a updated block needs to be flushed to disk, soft update checks dependency information of the block and does block content rolled-back before flush if necessary. When a flush of one block violates dependency constraint, soft update rolls the block state back to safe state because it maintains dependency information for each block.

3.2 Non-volatile RAM (NVRAM)

As mentioned above, synchronous metadata write leads to stronger consistency but causes many time-consuming seeks which result in unacceptable throughput. In contrast, asynchronous metadata write results in better performance but weaker consistency. In order to achieve good performance and strong consistency, NVRAM [1] is proposed to store the metadata writes. In their approach, metadata is synchronously updated to NVRAM first and

then flushed to disk in any order. Therefore, it has a similar performance with that of asynchronous metadata update. In case of a file system crash, metadata in the NVRAM will not be lost and thus the file system can still stay in a consistent state. The cost of this approach is high and extra memory copies of metadata are needed.

3.3 Log-structured file system (LFS)

In the present days, a file system can use logging in two ways—file system layout or file system enhanced. Unlike journaling file systems that journal data and metadata for enhancement, log-structured file system [12] [15] [16] [18] uses logging as file system layout.

Log-structured file systems treat the whole disk space as a circular log and append written data to the end of the log always. This method is optimized for write operations because no time-consuming seeks are needed. Basic data structures in LFS are similar to those of FFS, like inodes. Thus, the read performance is also similar to that of traditional file systems. However, in read-after-write or write-after-read cases, the cost of writing a disk block becomes unacceptable. LFS divides disk space into fixed-size chunks called segments. Data updates are delayed and collected until the total size reaches a full segment. At that time, the segment can be flushed to the end of the log. Using segments can amortize the cost in read-after-write or write-after-read cases. Nevertheless, this approach requires a cleaner for collecting available data in old segments when there are little empty segments for flush. The jobs of the cleaner are: (1) select a candidate segment for cleaning according to the cleaning policy, (2) read the available data in the candidate segment, (3) collect available data until the size is larger than a segment, and then append the segment to the log. Those activities need huge I/O operations and thus degrade the performance. The higher the disk utilization is, the more

clean activities are needed.

Log-appended writing implies high consistency semantic. Because LFS employees delayed write and segment-based write, it can order the permutation of each metadata and regular data. Thus update dependency constraints rules we mentioned in section 2.1 can be satisfied.



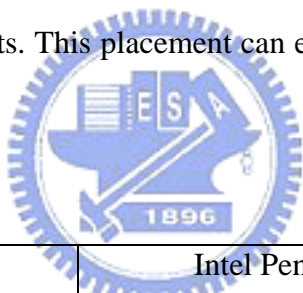
Chapter 4

Journaling analyses

We analyze metadata journaling about journal I/O traffic effect(in section 4.3) and commit interval effect(in section 4.4). Then remote journal scheme for file system performance improving is proposed in next chapter by observing and summarizing the result.

4.1 Experimental Environment

Table 2 shows the detailed hardware and software configuration. In the SCSI disk, we use a partition which resides on the middle tracks of the disk to prevent the zone effect from affecting the performance results. This placement can eliminate our estimate result error from zone effect in the disk.



CPU	Intel Pentium 4 1.6GHz
Memory	256MB DDR RAM
Disk	Seagate ST336753LW/P 15000 RPM, 3.9ms average seek time, 49 ~ 75 Mbytes / s transfer rate
OS	Linux kernel 2.6.5
NIC	Accton EN-1216 10/100Mb/s

Table 2 Experimental setup

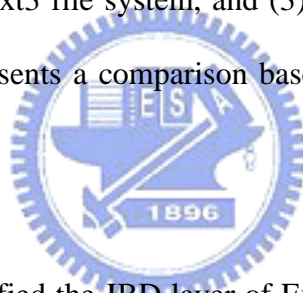
The benchmark we choose is postmark [10] version 1.5. Postmark is a benchmark which is suitable for simulating small-files based environment, like mail server ,news server and

OLTP environment.

In each experiment, we execute the benchmark for 10 times, with each time includes 150000 file and 20000 file system transactions which include file create, delete, read, and append. The type of transaction is choose randomly. The size of the benchmark files ranges from 500 bytes to 1000 bytes, and we start each time with a cold cache (i.e., reboot the system before a round is started).

4.2 Variance type of Ext3 file system

In the experiments, we estimate three types of Ext3 file system: (1) normal Ext3 file system, (2) non-journal (NJ) Ext3 file system, and (3) remote-journal (RJ) Ext3 file system. Normal Ext3 file system represents a comparison base relative to NJ-Ext3 and RJ-Ext3 file system.



In later sections, we modified the JBD layer of Ext3 file system in order to observe that how metadata journaling affects performance. That is, we want to estimate effect of superblock journal and metadata (and data in Ext3 journal mode) which are mainly journal I/O involved by Ext3 journaling. After JBD groups the journal data and commits it to the buffer cache layer, we intercept and drops the data and call the *journal_end_buffer_io_sync* function. We call it NJ-Ext3 file system in our experiments.

4.3 The Effect of Journaling

Although metadata journaling successfully reduces recovery time after a system crash, it brings performance overhead. Adding metadata journaling to a file system can cause 20% ~ 25% performances degradation, compared to a typical asynchronous-write file system [11],

especially in metadata bound workload.

In order to estimate how I/O activities of metadata journaling affects the performance of Ext3 file system, we run the postmark benchmark on Ext3 and non-journal Ext3 (NJ-Ext3). The results are shown in Figure 1.

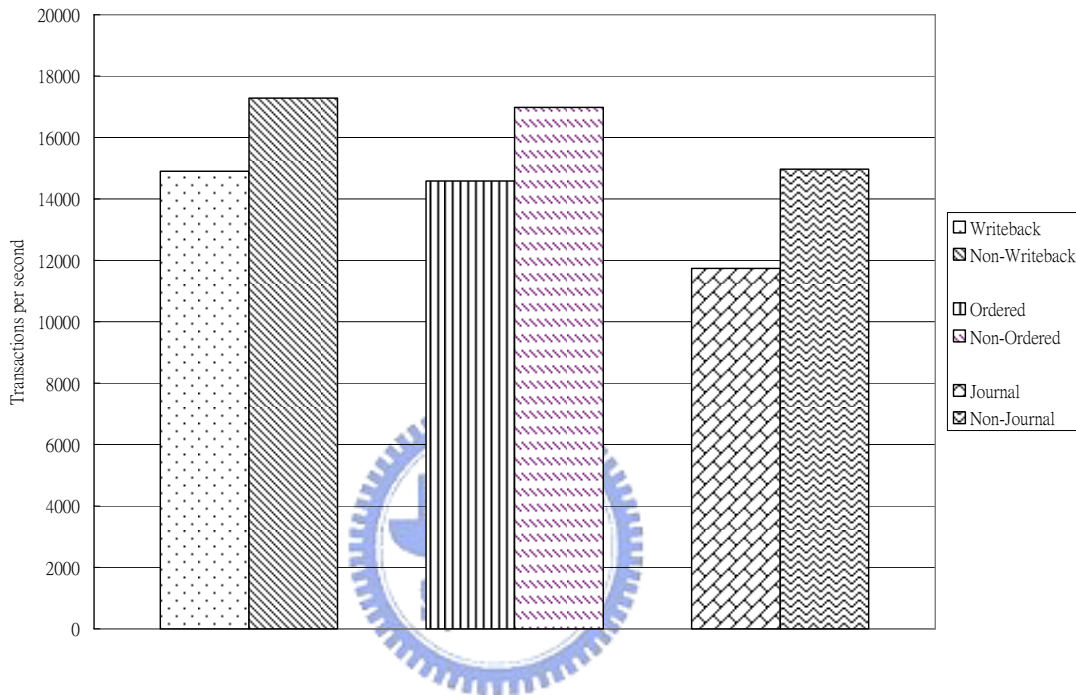


Figure 1 Performance comparisons of journal and non-journal Ext3 file system
[In this figure, journaling brings 16.42% overhead in the ordered mode; 15.97% overhead in the writeback mode; 27.51% overhead in the journal mode]

From the figure 1 we can see that journaling I/O causes performance degradation that ranges from 15.97% (in writeback mode) to 27.51% (in journal mode). The degradation comes from extra disk traffic and seeks. Take the journal mode as an example, the extra disk traffic is up to almost twice as big as original data size. The results reported by the NJ file system can be viewed as performance upper bounds of Ext3 file system.

Note that disk performance can also affect the results. Generally speaking, the performance degradation will be larger for a disk with a slower seek and rotation time.

4.4 The Effect of Commit Interval

In this section, we observe the effect of journal commit interval. The kjournald daemon groups journal data and commit it to the buffer cache layer periodically, and the commit interval is defined as time interval between two successive commits (i.e., 5 seconds for default). General speaking, a larger commit interval has a higher chance of data loss, and a smaller commit interval leads to lower performance because of extra seeks.

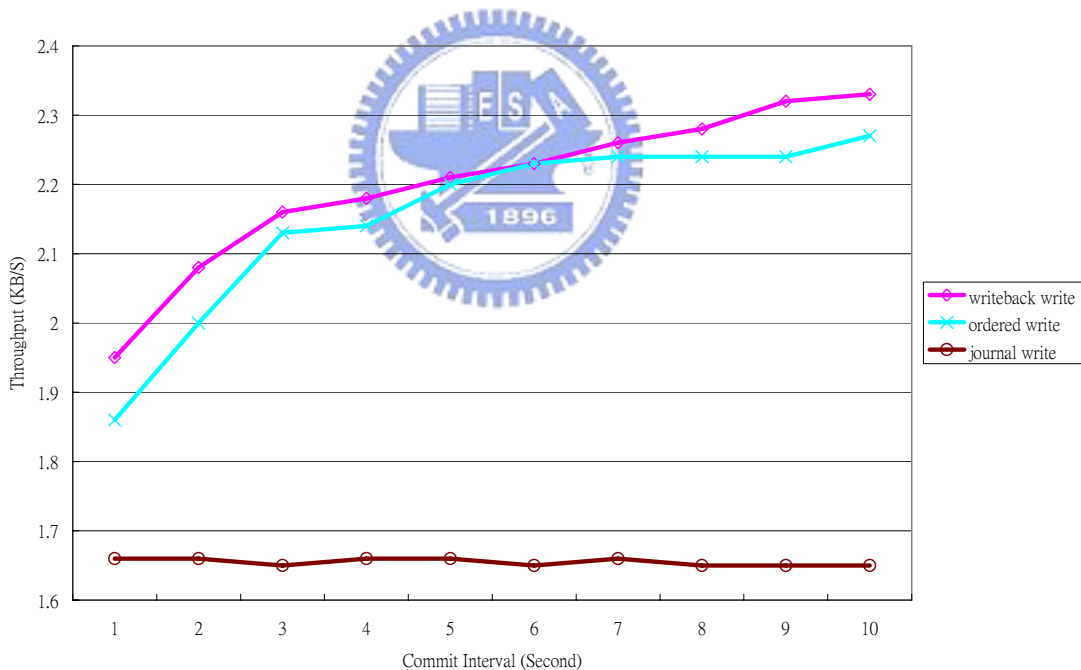


Figure 2 Performance Comparisons of Different Commit Intervals

X-axis indicates commit interval, which unit is second. y-axis indicates the throughput. Performance of ordered and writeback mode decreases with higher commit interval. Journal mode show almost the same performance in each commit interval.

Figure 2 shows the performance comparison among different commit intervals. From the figure we can see that, the throughput of writeback mode and journal mode increase as the interval becomes larger. A smaller interval results in worse performance since that it causes additional disk head seeks. A higher interval leads to better performance which benefits from delayed write effect. Note that the journal mode shows a different trend to the other modes. Journal mode handles large data traffic which includes regular data and metadata, so performance of journal mode is the worst in the three mode.

4.5 Observation

In section 4.3 and 4.4, we observe the factors that effect the journal file system. Journal I/O is necessary for consistency recovery but harms the performance. Higher commit interval brings higher performance which results from delayed write effect but will lose more data if crash happens.

We propose remote journaling in next chapter. Remote journaling removes journal I/O from disk to network and has low commit interval which implies high consistency semantic. The network overhead and throughput benefit of remote journaling will be estimated in our experiments.

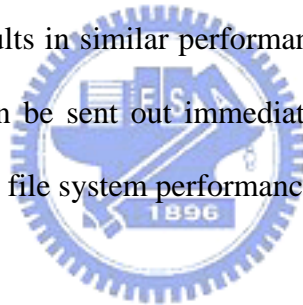
Chapter 5

Remote Journaling

We explain the concept of remote journal and how it can be used in this chapter, and will have experiments in next chapter.

5.1 Concept of Remote Journaling

In this chapter, we propose a remote journaling architecture, which journals data to a remote journal server instead of local disk. In addition to guaranteeing consistency, a remote journaling file system also results in similar performance with non-journal file systems. This is because the journal data can be sent out immediately when it is generated and thus the journal traffic will not harm the file system performance.



Different with disk, network transmission does not need position time in disk I/O. The position time includes seek time and rotation time are time-consuming and harms performance. Remote journal scheme can prevent more necessary position time when flushing journal to local disk and then improving performance.

Moreover, remote journaling is a cheap solution. Many hosts can share a single journal server at the same time. Since the workload of the journal server is write-dominated, the disk layout of the journal can be designed to optimize the write performance.

5.2 Applications

Remote journaling can be applied to any journal file systems. This mechanism is especially useful for metadata bound workloads, like online transactions environment, web server, or news server. The network bandwidth taken by remote journaling is acceptable when using in network applications. File system consistency recovery which is the same with traditional process besides reading journal from network reduces downtime, which is important in a commercial service. Moreover, remote journal can be used in a storage cluster, like [錯誤! 找不到參照來源](#)。3. Each storage server amortizes the cost of journal server.

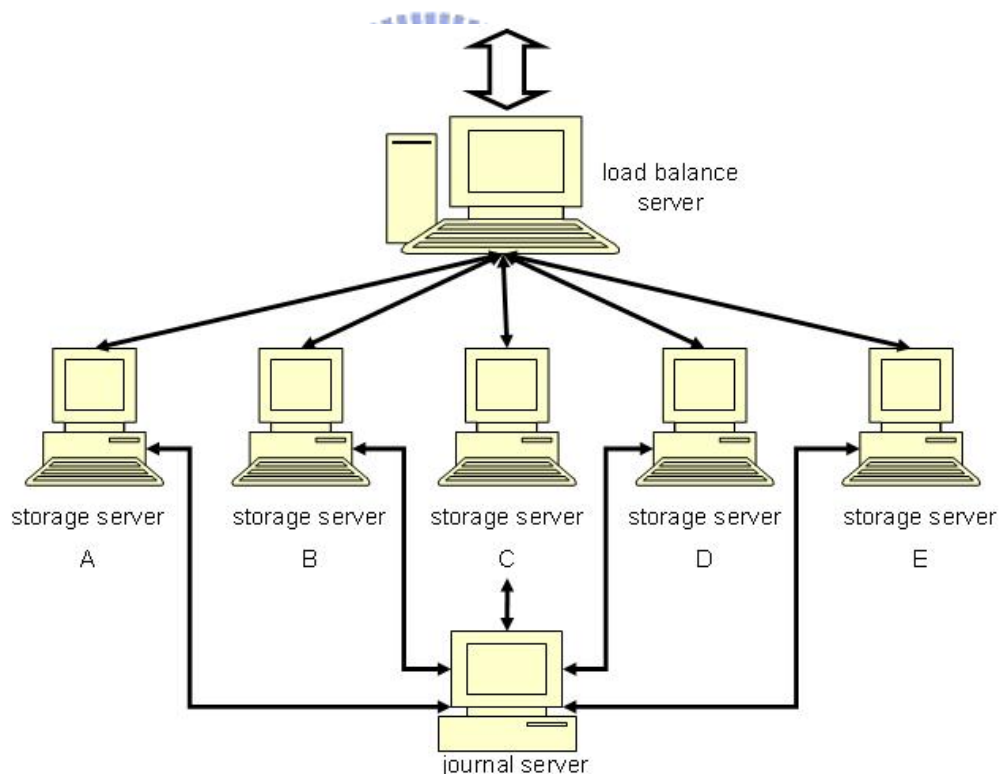


Figure 3 the example application

5.3 Implementations

We modify the daemon, `kjournald`, for remote journaling. When user specifies a remote journal mode in file system mount table, the modified `kjournald` tries to connect to the remote journal server. Metadata transfer is through TCP/IP, which guarantees the transfer can be accomplished without loss. However, if a transmission error happens, which may be caused by network congestion or server failures, the modified `kjournald` switches to the local journal mode for file system safety consistency.

The main function for journal flush in `kjournald` is `journal_commit_transaction`. `Journal_commit_transaction` first update journal superblock which includes journal information. We modify `journal_update_superblock` function from disk commit to network commit. Then `journal_commit_transaction` tries to commit data buffer in ordered mode. After flush of all data that is needed flushing before metadata completes, we collect buffers which have journal data and commit it from buffer cache layer to network layer. If all commits are accomplished, we insert a checkpoint and release this file system transactions.

When recovery process starts, we try to connect remote server. We read journal superblock and necessary information for recovery from remote server. If there is any error, unfortunately we have to do a whole disk scanning because we do not have any information in order to recovery.

The best choice of the file system on the remote journal server is log-structured file system. Because workload on remote journal server is write-oriented in most time and log-structured file system has excellent performance for such workload. In addition to the

performance consideration, log-structured file system can achieve high consistency for the file system. The fast recovery of remote server file system is important, because any error of remote server causes clients doing whole disk scanning.



Chapter 6

Experiments result

We estimate the performance and overhead which is bring by remote journal here. Remote journal scheme removes journal data traffic form disk to network. So we estimate performance raise, network bandwidth usage and other factors in this chapter.

6.1 Performance comparisons

There are throughput comparisons of three journal mode in Ext3 file system. We add non-journal serious as a the upper bound here. This helps us realizing the overhead of remote journaling. Figure 4 shows the overhead brings by remote journal is about 5% ~ 7% to upper bound and still outperforms about 10% (in writeback and ordered mode) to 21% (in journal mode).

Generally speaking, Ext3 journal mode that has to journal both data and metadata brings higher overhead, thus performance is only 78% of writeback mode. However, by remote journal the gap between writeback and journal mode becomes narrow. Figure 4 indicates the performance of journal mode raises to 86% of writeback mode by remote journaling. The raise is significant because it improves the availability of journal mode.

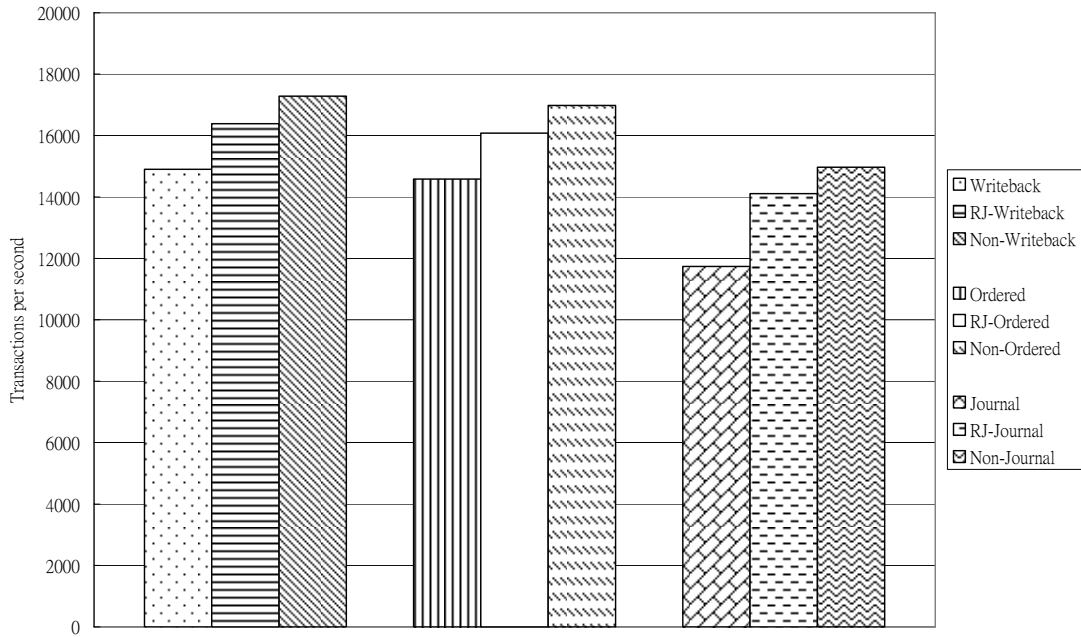
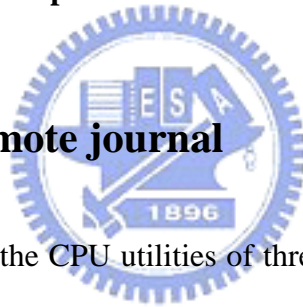


Figure 4 performance of remote journal

6.2 CPU utilities of remote journal



In this section, we record the CPU utilities of three mode and compare it. Figure 5, 6, 7 show the curves of normal mode, non-journal serious, and remote journal serious.

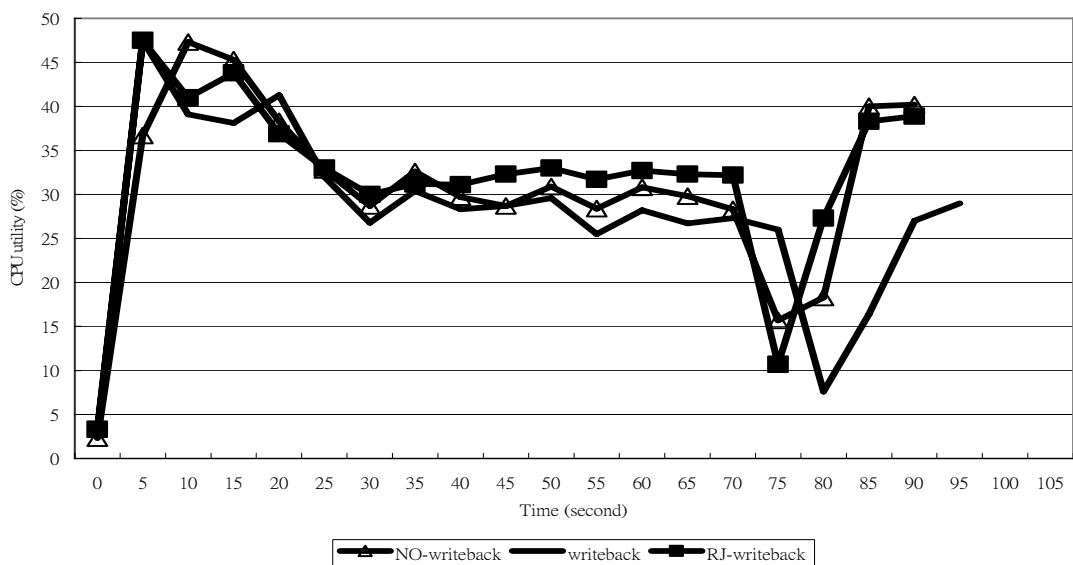


Figure 5 CPU utilities of writeback mode

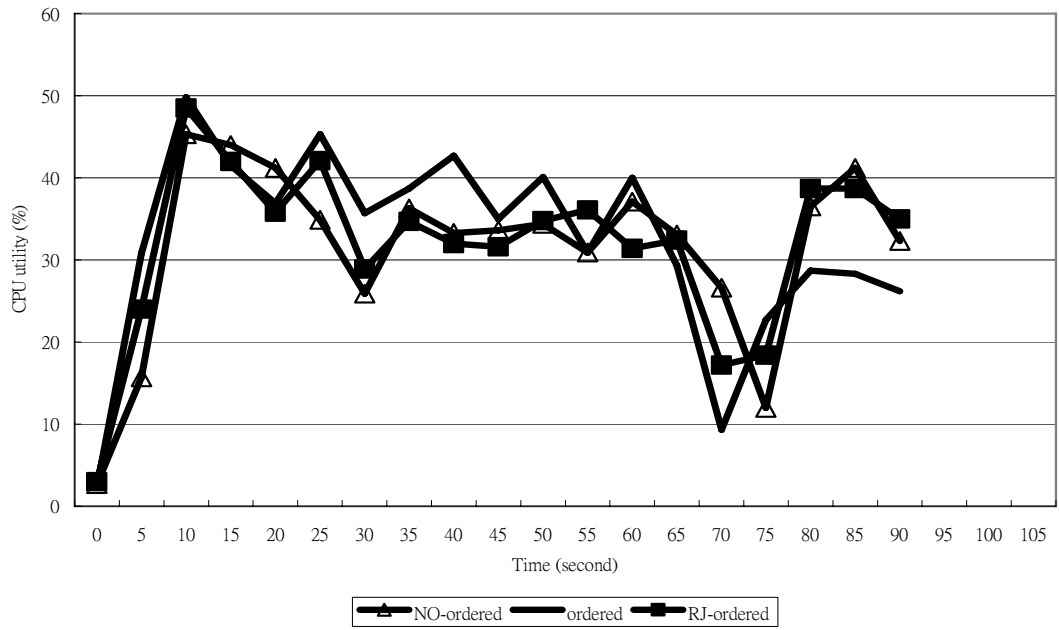


Figure 6 CPU utilities of ordered mode

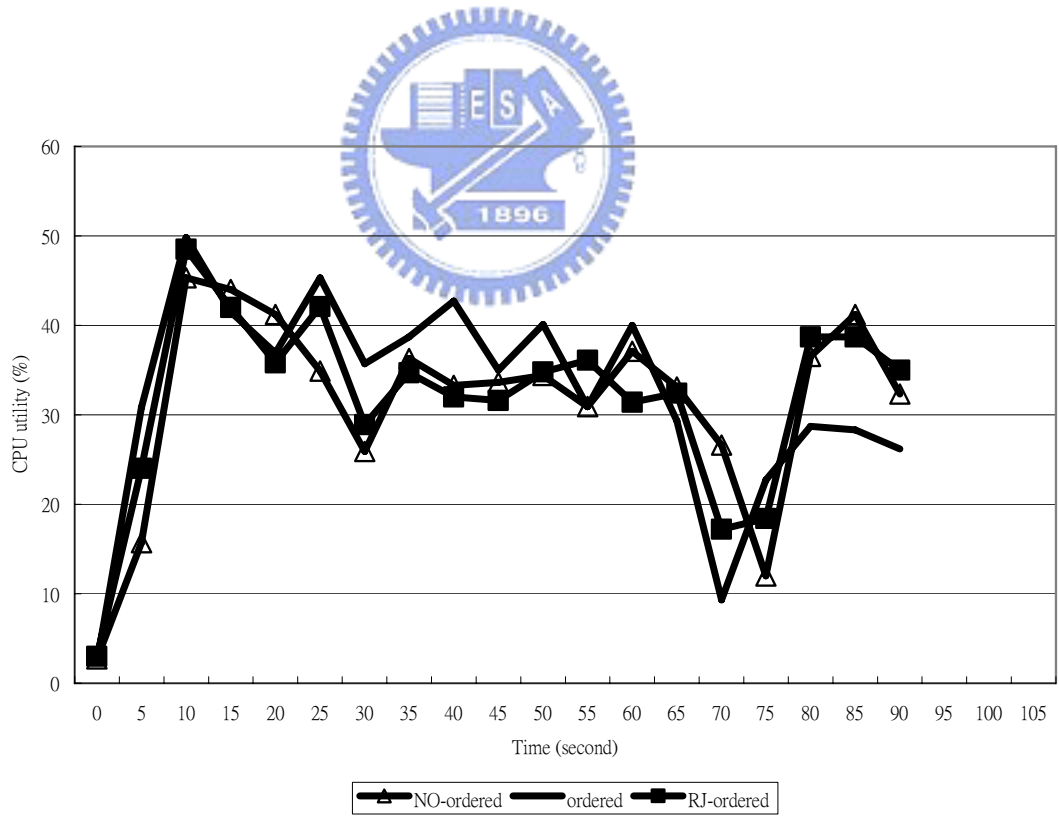


Figure 7 CPU utilities of journal mode

The overhead about CPU brings by remote journal can be observed here. We can see that

CPU usage time of non-journal and remote journal is higher than normal mode about 5% to 20%. However, the total time need to complete benchmark is less.

In order to understand how much CPU overhead will remote journaling brings, we integrate the area in figure 5 ~ 7 and show the result in table 3,4 and 5. Although remote journal brings higher CPU utilities, total CPU time approaches normal mode (lower than 10%).

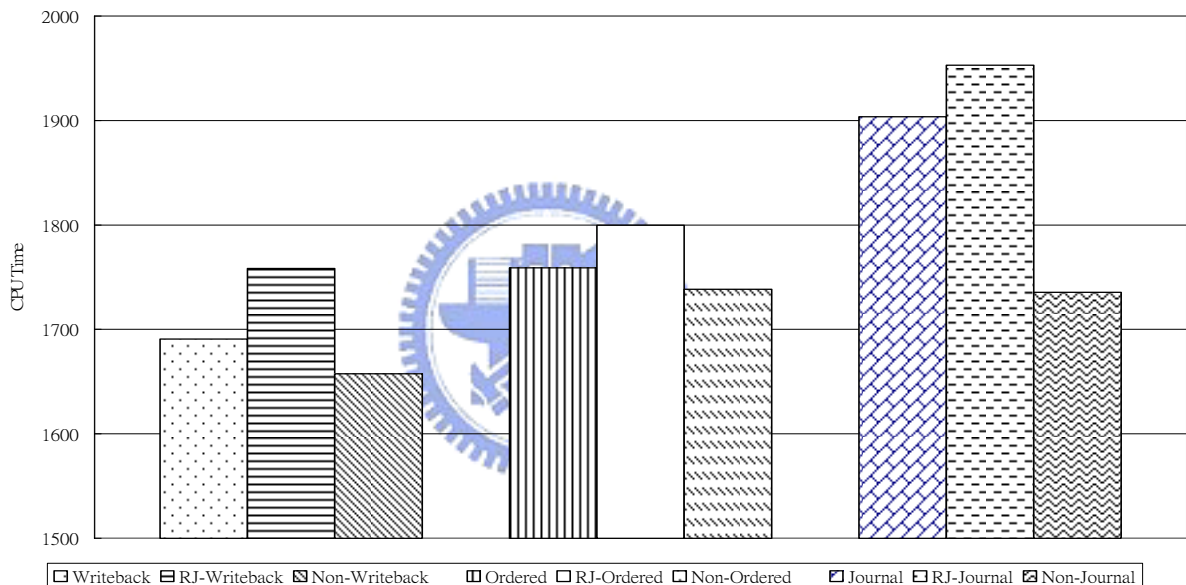


Figure 8 CPU time comparison

6.3 Network bandwidth taken by remote journal

Although file system performance benefits by removes journal I/O from disk to network in remote journal scheme, it may damage the network availability when using in network applications. Thus we estimate the network bandwidth taken by remote journal in three mode. The results are shown in Figure 9. Y-axis shows network bandwidth percentage which is used

by remote journaling in gigabit Ethernet and X-axis is three mode of Ext3 file system.

Because the writeback and ordered mode only log metadata and journal mode logs both metadata and data, the journal mode has heavier burden on network than other two mode. In our workload writeback and ordered mode have only less than 2% network bandwidth and journal mode uses 6.6% network bandwidth in gigabit Ethernet. Note that the network burden in journal mode may different with workload. Thus the network overhead brings by remote journal becomes larger when workload includes larger data.

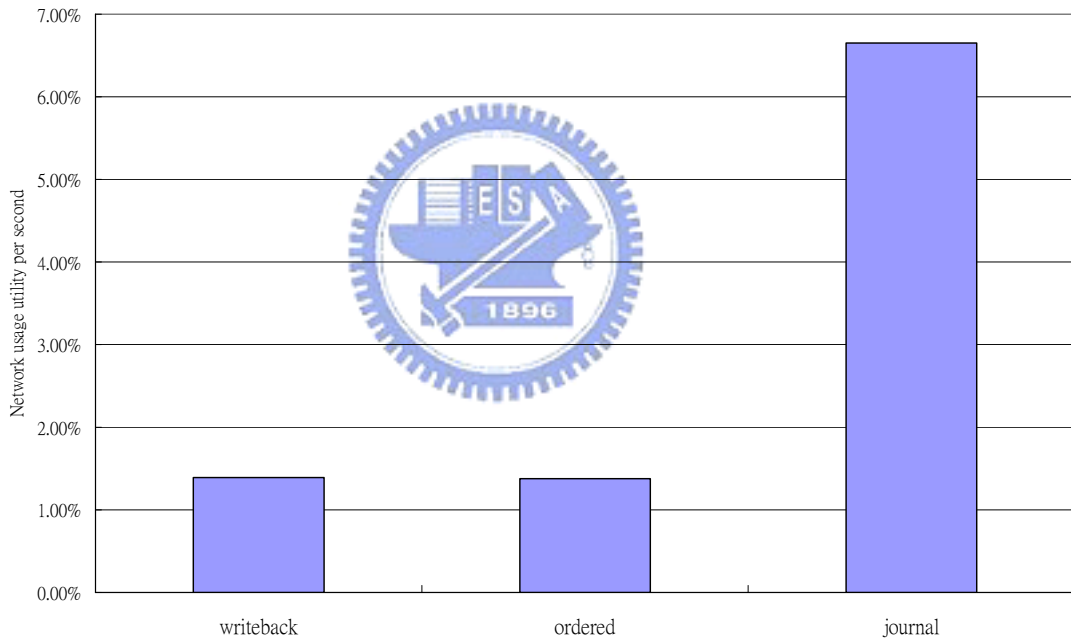


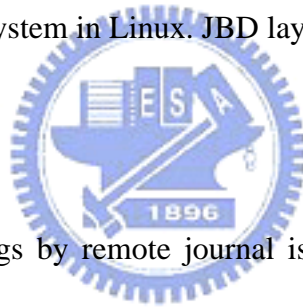
Figure 9 Network usage comparison

Chapter 7

Conclusion and future works

7.1 Conclusion

In this paper, we proposed a scheme named remote journal which improving performance of journal file systems. Remote journal improves file system performance by removing journal I/O from local disk to remote journal disk by network. If there is error when doing remote journaling, we switch the remote journal into local journal in order to guarantee the fast recovery. The consistency semantic of original file system will not be harmed because we do log the same journal data. We implement remote journal scheme in Ext3 file system, a popular journal file system in Linux. JBD layer in Linux and the daemon, kjournal, has been modified here.



The main advantage brings by remote journal is obvious performance upgrade which mainly results from remove of journal I/O. Another advantage is cost in hardware. A remote journal server can support many clients and thus the cost can be shared.

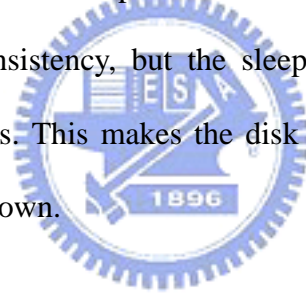
According to the experiments in this paper, remote journal increases about 10% (in Ext3 writeback and ordered mode) to 21% (in Ext3 journal mode) performance, but the penalty is light. Although remote journal does need more CPU time for network transfer, the overhead is less than 10%. On the other hand, we also consider that the journal traffic effect to network bandwidth. In our experiment result, overhead in writeback and ordered mode is light because only metadata is logged into journal by network. In our metadata bound workload, their overhead are just less than 2%. Nevertheless, Journal mode will have higher overhead because the journal traffic depends on workload. With big files workload, journal traffic is heavy and

overhead will be higher.

To sum up, remote journal scheme is a easy and cheap solution for improving performance of journal file system. It can be easily used after patching the kernel. In most time it brings better performance and keeps the same file system consistency semantic but low overhead penalty.

7.2 Future works

When we adopt the remote journal scheme in mobile storage, it can be used in disk power management. In order to save unused disk spinning power, most power saving approaches in disk try to make disk sleep time much longer. However, journal flush activities needs frequent update for consistency, but the sleep time in mobile storage suffers from frequent journal flush activities. This makes the disk wasting more energy on mode switch that includes spinning up and down.



Remote journal can solves this problem under this condition. Remote journal server can be a FTP server or a free mail space. If general Ethernet is used, remote journal can save energy by removing journal data I/O without any extra scheme. If wireless network is used, how to place journal needs more consideration. Because data transfer by wireless network also consumes much energy, a monitor and a arbiter are needed for controlling journal data flow. The monitor watches the status of disk and wireless network and know the power profile of the disk and wireless network. The arbiter controls the journal data placement by the monitor report. The journal superblock needs to be modified in order to indicate where the journal data is placed.

When the disk just enters sleep state, the arbiter redirects journal flush activities to remote server by wireless network. And if wireless network enters sleep mode or the energy needs by wireless transfer is greater then waking up disk, the arbiter flushes the journal data to disk.



Reference

- [1] Baker, M., Asami, S., Deprit, E., Ousterhout, J., Seltzer, M. “Non-Volatile Memory for Fast, Reliable File Systems,” Proceedings of the 5th ASPLOS, pp. 10–22. Boston, MA, Oct. 1992.
- [2] Chutani, S., Anderson, O., Kazer, M., Leverett, B., Mason, W.A., Sidebotham, R. “The Episode File System,” Proceedings of the 1992 Winter USENIX Technical Conference, pp. 43–60. San Francisco, CA, Jan. 1992.
- [3] Elkhardt, K., Bayer, R. “A Database Cache for High Performance and Fast Restart in Database Systems,” ACM Transactions on Database Systems, 9(4), pp. 503– 525. Dec. 1984.
- [4] Ganger, G., Patt, Y. “Metadata Update Performance in File Systems,” Proceedings of the First OSDI, pp. 49–60. Monterey, CA, Nov. 1994.
- [5] Ganger, G., Patt Y. “Soft Updates: A Solution to the Metadata Update Problem in File Systems,” Report CSETR-254-95. University of Michigan, Ann Arbor, MI, Aug. 1995.
- [6] Ganger, G., McKusick, M.K., Soules, C., Patt, Y., “Soft Updates: A Solution to the Metadata Update Problem in File Systems,” to appear in ACM Transactions on Computer Systems.
- [7] Gray, J., Reuter, A. Transaction Processing: Concepts and Techniques. San Mateo, CA: Morgan Kaufmann, 1993.

- [8] Hagmann, R. “Reimplementing the Cedar File System Using Logging and Group Commit,” Proceedings of the 11th SOSP, pp. 155–162. Austin, TX, Nov. 1987.
- [9] Haskin, R., Malachi, Y., Sawdon, W., Chan, G. “Recovery Management in QuickSilver,” ACM Transactions on Computer Systems, 6(1), pp. 82–108. Feb. 1988.
- [10] Katcher, J., “Postmark: A New File System Benchmark,” Technical Report TR3022. Network Appliance Inc., Oct. 1997.
- [11] M. I. Seltzer, G. R. Ganger, M. K. McKusick, K. A. Smith, C. A. N. Soules, and C. A. Stein. Journaling versus soft updates: Asynchronous meta-data protection in file systems. In Proc. of the 2000 USENIX Annual Technical Conference: San Diego, California, USA, June 2000.
- [12] Matthews, J., Roselli, D., Costello, A., Wang, R., Anderson, T. “Improving the Performance of Log-Structured File Systems with Adaptive Methods,” Proceedings of the 16th SOSP, pp. 238–251. Saint-Malo, France, Oct. 1997
- [13] McKusick, M.K., Ganger, G., “Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem,” Proceedings of the 1999 Freenix track of the USENIX Technical Conference, pp. 1–17. Jun.1999.
- [14] McKusick, M.K., Joy, W., Leffler, S., Fabry, R. “A Fast File System for UNIX,” ACM Transactions on Computer Systems 2(3), pp 181–197. Aug. 1984.
- [15] Rosenblum, M., Ousterhout, J. “The Design and Implementation of a Log-Structured File System,” ACM Transactions on Computer Systems, 10(1), pp. 26–52.Feb. 1992.

- [16] Seltzer, M., Bostic, K., McKusick, M.K., Staelin, C. “An Implementation of a Log-Structured File System forUNIX,” Proceedings of the 1993 USENIX Winter Technical Conference, pp. 307–326. San Diego, CA, Jan. 1993.
- [17] Stein, C. “The Write-Ahead File System: Integrating Kernel and Application Logging,” Harvard University Technical Report, TR-02-00, Cambridge, MA, Apr. 2000.
- [18] Wilkes, J., Golding, R., Staelin, C., Sullivan, T. “The HP AutoRAID hierarchical storage system,” 15th SOSP, pp.96–108. Copper Mountain, CO, Dec. 1995.

