# Practical Schemes using Logs for Lightweight Recoverable DSM [†]

Youngjae Kim

Computer and Software Technology Laboratory
Electronics and Telecommunications Research Institute
Gajeong-dong Yusong-gu, Taejeon, 305-350, Korea
kimyj@etri.re.kr

Soyeon Park Seung Ryoul Maeng

Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
Kusong-dong Yusong-gu, Taejeon, 305-701, Korea
{sypark, maeng}@calab.kaist.ac.kr

## Abstract

In the existing Fault-Tolerant Software Distributed Shared Memory (FT-SDSM) with the message logging, the logs are used only to recover the failed nodes. In our previous work, we have implemented a lightweight logging protocol, called *remote logging*, on the SDSM for fault tolerance, which incurs low logging overhead with a fast network and a remote memory for back-up data. In this paper, we propose two practical schemes for the logs, which enhance our based remote logging protocol. In these proposed schemes, the logs are applicable to reduce the stalled times for updating the invalid pages, minimizing the failure-free execution time.

## Key Words

Distributed System, Software Distributed Shared Memory, Fault-tolerance System, Message Logging

## 1. Introduction

Software Distributed Shared Memory (SDSM) [1] provides a global memory abstraction on the top of the physically distributed memories to the programmers and simplifies the parallel programming tasks. The SDSM has been attractive for the large clusters due to its high performance and scalability. However, the probability of failures increases as the system size grows. Thus, an impressive amount of research has been conducted for FT-SDSM without degrading the system performance during the failure-free execution.

The message logging with independent checkpointing is an attractive approach on home-based SDSM with the fault-tolerant capabilities [2,3,4,5,6]. It guarantees the bounded recovery time and ensures no *domino effect* [5]. Each node logs the received messages into a proper storage during the failure-free execution. If a node comes to fail, it restarts from the last checkpoint and regenerates the previous state of the system, using the logs, before the crash happened.

The major concern of FT-SDSM is to minimize the logging overhead because it might lead to increase the failure-free execution time of an application. A few logging schemes have been proposed for a lightweight and scalable FT-SDSM [5,6].

In this paper, we propose two practical schemes for the logs, called *Page-home Log Exploitation Scheme (PLES)* and *Log-home Log Exploitation Scheme (LLES)*, which improve the performance of FT-SDSM by exploiting the logs. They are implemented on our previous work, remote logging [7], which is tightly proposed for home-based SDSM [8]. It only logs the coherence-related messages into the volatile memory of a remote node and utilizes a high-speed network. The diff logs are used to facilitate the fast updates of the invalid pages, reducing total failure-free execution time.

To evaluate our schemes, we have conducted some experiments on a cluster of eight PCs under the five applications from SPLASH-2 benchmark suit [9]. The results demonstrate that our proposed schemes reduce the times for the synchronization operations as well as for the waiting times to update the invalid pages, and hence minimize total failure-free execution time of FT-SDSM with a remote logging protocol.

The remainder of this paper is organized as follows: Section 2 describes a home-based SDSM. In section 3, the remote logging protocol is given as our base model. Section 4 describes our proposed exploitation schemes for the logs. The experimental results are present in Section 5. Finally, Section 6 concludes the paper.

## 2. Home-based SDSM

Home-based SDSM is based Home-based Lazy Release Consistency (HLRC) [8] which is a variant of Lazy Release Consistency (LRC) [10]. The LRC is a representative implementation of the release consistency protocol [11]. It guarantees the memory consistency only after the synchronization point because the update propagation is delayed to the next lock acquire or the barrier operation. For the memory consistency, it uses a

[†] This research is supported by KISTEP under the National Research Laboratory program.

write-invalidation protocol that invalidates all copies of the shared memory pages modified before the lock is acquired. The write operations are grouped into intervals delimited by the synchronization operations. The intervals represent the order of the write operations on a shared memory page. The invalidation is performed at a synchronization point with the write-notices piggybacked with the lock grant and barrier release messages. The write-notices include all page numbers to be invalidated. The LRC also uses a multiple-writer protocol that allows each process to modify the non-overlapping parts of a shared memory page concurrently, reducing the network traffic overhead even though it consumes a significant amount of memory and incurs an extra computation.

In the recent years, HLRC is a popular memory consistency protocol. Each shared memory page has an assigned home-node that maintains the most recent pages. It gives some advantages to the SDSM as follows: (i) The local write or read operation on a shared memory page in a home-node does not cause any virtual memory trap. In addition to that, the summary of modifications (i.e., diffs) is not created. The diff is generated from comparing the modified page with the previous copy of the same page (i.e., twin). (ii) An up-to-date page is fetched from its home-node by a single round-trip message, which simplifies the consistency protocol. These benefits enable the SDSM to be more efficient and scalable.

## 3. Our Previous Work

### 3.1 Failure Model

The fail-stop model assumes that a failed node should not affect any other running ones and the network is considered to be failure-free. Thus, some communication operation failure is handled as a remote node failure. We use a low overhead communication layer, VIA (Virtual Interface Architecture) [12], which tolerates the transient network fabric errors by the packet retransmission and guarantees the FIFO message delivery.

### 3.2 Remote Logging Protocol

Each shared page has both a home-node, called a page-home, and a back-up node, called a log-home and each node has a write-notice back-up node, called a notice-node. In this paper, for each shared page belonging to some node (or page-home), the log-home of a shared memory page and the notice-node of a node (or page-home) are supposed to be the same. The remote logging protocol uses the volatile memory of a remote node rather than local disk for back-up data as in the previous stable logging protocol [5]. The elimination of this disk access minimizes the logging overhead as shown in [7]. Also, the high-speed network will reduce the network traffic overhead. It tolerates a few multiple node failures except when a page-home and its log-home or a node and its notice-node come to fail simultaneously. The protocol stores the write-notice messages received by a node into its notice-node and the diff messages of some pages into both their page-homes and their respective log-homes. The remote copy pages are not logged because they can be regenerated by their page-homes during the recovery.

### Write-notice and Diff Logging

In HLRC, two major messages affecting the memory consistency are the write-notice and diff messages.

When a lock releaser sends the write-notice messages to the next lock requester for invalidating the old copies, the latter logs them in the volatile memory of its notice-node (or log-home). A failed node can extract the corresponding write-notices from its log-home at every synchronization point during the recovery. The write-notice logging is also conducted during the barrier operations similarly to the lock acquire operations.

Also, when a page writer creates some diffs at a synchronization point, it sends them not only to their page-homes but also to their corresponding log-homes. The recent communication layer (e.g., VMMC and VIA) provides a useful function, the direct data transfer between the sender's and the receiver's virtual address spaces, and hence the diff logging can be performed more quickly and without the intervention of the host-processor. Contrary to the write-notice logging, the diff logging has to be performed in the log-homes as well as in the page-homes. The diff logs in a log-home are used to recover its peer page-home on a failure and those in a page-home are used to regenerate the old pages requested by other failed nodes during the recovery. Thus, even though a page-home attempts to write on its self-home pages, it must create the diffs and log them in its local memory, which is different from the previous HLRC protocol.

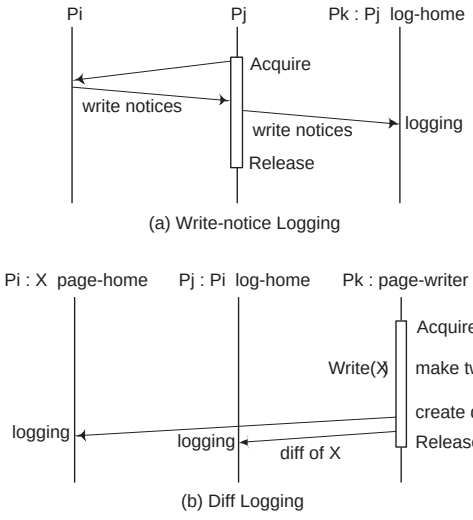


**Figure 1. A Snapshot of Remote Logging**

### Example

Figure 1 shows a snapshot of the remote logging protocol. Figure 1(a) shows an example of the write-

notice logging at the lock acquire. A process $P_i$ is a previous lock owner. A process $P_j$ is a next lock requester. A process $P_k$ is a notice-node (or log-home) of $P_j$. $P_j$ receives a lock grant message with some write-notices including all information of the modified pages by $P_i$. $P_j$ forwards the received write-notice messages to $P_k$ and invalidates its local remote copy pages. The invalidation and logging operations are performed in parallel.

Figure 1(b) shows an example of the diff logging. In this example, a shared page *X* has a page-home $P_i$ and a log-home $P_j$. $P_k$ is a page-writer on *X*. When a write miss occurs at a page *X*, $P_k$ creates the twin of *X* and modifies it. At a lock release, $P_k$ creates a diff message of *X* to send it to its page-home $P_i$ and directly logs the same one into $P_j$. In $P_i$, the diff of *X* is applied to its home page of *X* and is logged into its local memory.

## 4. Practical Schemes for the Logs

In our previous work (i.e., remote logging) and the other traditional logging protocols, the logs were useless during the failure-free execution and the process has to fetch a whole page from its page-home whenever the invalid page is accessed, incurring high cost. In this paper, based on the remote logging protocol, we will introduce two efficient exploitation schemes for the logs stored in the page-home as well as in the log-home because the diff logs in the memory can be sufficiently usable to reduce the times for updating the invalid pages. The first scheme is the *Page-home Log Exploitation Scheme (PLES)* that uses the diff logs of page-homes and the second one is the *Log-home Log Exploitation Scheme (LLES)* that uses the diff logs of log-homes for better performance. To our knowledge, this is the first work that uses those diff logs to enhance the performance of FT-SDSM during the failure-free execution.

### 4.1 Page-home Log Exploitation Scheme (PLES)

In HLRC, a process updates its invalid page by fetching a whole page from its page-home even though it needs only its small parts of the page for updates. We first have experimented on measuring the impact of the communication latency with the message length between two nodes. Each node contains a Pentium III 850 MHz processor and a NIC holding LANai 9.1 processor. They are interconnected with 1.3 Gbps switched Myrinet. We used VI-GM [13], a kind of VIA implementations provided by Mricom [13]. The results showed that the data transmission latency scales linearly with the message length. For example, a half round-trip time for 2 Kbytes message is $118\mu s$ and for 4 Kbytes is $215\mu s$. It means that the superfluous data transmission might lead to somewhat performance degradation.

We suggest an adaptive strategy in which a page-home can adaptively send a whole page or only its modified parts of the page with the diff logs. In a case of servicing with the logs, the computation overhead will increase while the communication overhead will decrease. The page-home wastes some CPU cycles in computing the modified ranges of the page and making them on a message. To reduce the non-negligible cost, we use some optimization techniques and try to estimate whether the CPU cost will exceed some threshold or not while searching for the appropriate logs.

#### Adaptive strategy

If a process attempts to access an invalid page, it must wait for update from its home. The update could be either a whole page (4 Kbytes) or the only modified parts since the last access. To make a choice, we define $T_{page}$ as a waiting time for a whole page and $T_{diff}$ as a waiting time for the only modified parts of the page. By estimating them, a page-home can decide whether to service for the request with the logs or not according to the minimum waiting time $T_{best}$ (i.e., $T_{best}$ = Min ($T_{page}$,$T_{diff}$)). Most of $T_{page}$ occupy the data transmission of 4 Kbytes whereas $T_{diff}$ includes three kinds of overheads as follows: (i) $T_{XB}$ is a transmission time for *X* bytes where *X* is the size of the modified parts of the page. (ii) $C_{cup}$ is the overhead consumed in searching for the corresponding diff logs and computing its modified ranges in a page. (iii) $C_{mem}$ is the memory copy overhead for creating a message with the modified parts of the page. If the most parts of a page have been modified, the service with the logs might not be desirable in performance. Even though the size of the modified parts can be very small, since $T_{diff}$ is the sum of $T_{XB}$, $C_{cup}$ and $C_{mem}$, the overhead of servicing with the diff logs can be dramatically higher than that of only sending a whole page.

For the decision purpose, we define two thresholds as follows: (i) $S_{size(H)}$ is a size threshold to service the only modified ranges of the page and (ii) $S_{depth(H)}$ is an interval depth threshold to search for the corresponding set of the logs, where H represents a page-home. We can find the dependency between $S_{size(H)}$ and $T_{XB}$, and between $S_{depth(H)}$ and $C_{cpu}$. In this strategy, when a page-home receives a page request, it first checks the version difference between the home page and the remote copy. If it is higher than $S_{depth(H)}$, the page-home simply sends a whole page. If a page-home learns that the size of the modified ranges in a page exceeds $S_{size(H)}$ in the middle of servicing with the logs, it also quits servicing with the logs and simply sends a whole page.

#### Optimization

A straightforward method to reply for a page request with the logs is to extract the previously received diff messages from the log pool and to simply pack them up on a message. However, it incurs the non-negligible overhead of redundant data in a message. Thus, we devise two optimization techniques to reduce such an overhead of the memory copies (i.e., $C_{mem}$). In the *diff range technique*, a page-home finds the only modified ranges of a page in advance before the packing operation begins. It has some advantages of eliminating an unnecessary copy

operation and reducing the size of a response message. However, there is still some problem of the frequent memory copies if the modified parts of the page are highly scattered in the whole range of a page. Thus we propose the *page blocking technique*. It divides a page into blocks and uses a write vector indicating which blocks are modified in a page. The size of a write-vector is the same as the number of blocks in a page. If a page-home finds a modified block of the page during searching for the appropriate diff logs, it sets the corresponding bit of the write-vector. Then it makes all blocks set by 1 on a message and sends it to the page requester. The performance tradeoff is in between the frequency of the memory copies and the size of the superfluous data. We fix the unit size of a block with 32 bytes for efficiency.

### 4.2 Log-home Log Exploitation Scheme (LLES)

A log-home logs all diffs which its peer page-home receives. If a log-home accesses an invalid page of which page-home is its peer one, it can update the invalid page with its local diff logs instead of requesting a page to its page-home. However, the log-home cannot use its local diff logs if the page-home has ever written the page since the last time that its log-home has fetched it. In this case, the log-home simply requests a page to its page-home. The diffs created by the page-home are not logged in its log-home because they can be regenerated by the page-home during the recovery.

Typically, the cost of the memory copy operations is less than that of fetching a whole page from its remote node. However, the frequent memory copies might lead to higher overhead. Thus, to limit the frequency of the memory copies, we define an interval depth threshold $S_{depth(LH)}$ to search for its local diff logs in a log-home where LH represent a log-home. If an interval difference between the version of the expected page and that of the old page is beyond $S_{depth(LH)}$, the log-home stop searching for the diff logs and simply request a page to its page-home.

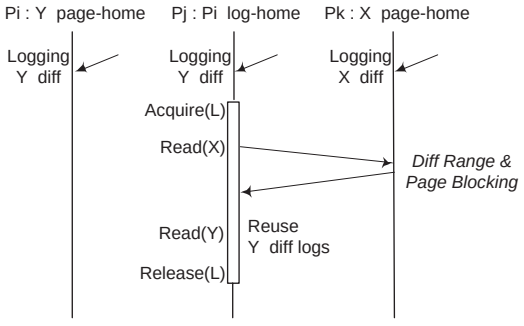


**Figure 2. A Snapshot of Practical Schemes for the Logs**

### 4.3 Example

Figure 4.2 shows an example of our proposed schemes. In this example, *X* and *Y* are the shared pages and their page-homes are assigned to the processes $P_k$ and $P_i$ respectively. A process $P_j$ attempts to read an invalid page *X*. This attempt causes a virtual memory trap and makes $P_j$ request a page to $P_k$. Then $P_k$ services adaptively for that with the diff range and the page blocking techniques as described above. Since $P_j$ is the log-home of $P_i$, when a page fault of an invalid page *Y* occurs in $P_j$, $P_j$ first compares the version of the invalid page *Y* with the expected version of that. If either the interval depth is beyond $S_{depth(LH)}$ or $P_i$ has modified the page *Y* since the last time that $P_j$ has fetched it, $P_j$ will simply request a page to $P_i$. Otherwise, $P_j$ updates the invalid page *Y* with its local diff logs for itself.

## 5. Performance Evaluation

Section 5 describes our experiment environments and presents the evaluation results of our proposed schemes. We first describe the adequate thresholds used in our implementation of different schemes for each application. Secondly, we show the usability ratio of diff logs for each scheme. We then evaluate the network overhead based on total number and size of messages transferred between all nodes. And finally we show the detailed breakdown of total execution time of each application.

### 5.1 Experiment Setup

Our experiments are performed on a cluster of eight PCs running LINUX version 2.2.15. Each node contains an 850 MHz Pentium III processor. All nodes are interconnected via a fast Myrinet of 1.3 Gbps switch. We evaluated our proposed schemes by incorporating them into the modified version of FT-SDSM with a remote logging protocol [7]. For measuring the failure-free execution time, the five parallel applications from SPLASH-2 benchmark suit [9] were used, including Ocean, Water-Spatial, Raytrace, 3D-FFT and Radix. Table 1 presents the characteristics of each application.

**Table 1. Applications' Characteristics**

| Program | Dta Set Size | Synchronization |
|---|---|---|
| **Ocean** | 515 X 512 grid | Locks and barriers |
| **Water-Spatial** | 1728 molecules | Locks and barriers |
| **Raytrace** | car.evn (256 X 256) | Locks and barriers |
| **3D-FFT** | $2^8$ X $2^7$ X $2^7$ data | Barriers |
| **Radix** | 4M keys | Locks and barriers |

The performance evaluation is compared with the following five schemes based on home-based SDSM.

- **No Logging Scheme**
  It is a home-based SDSM without any fault-tolerant capabilities.

- **Remote Logging Scheme**
  It is a home-based SDSM supporting a lightweight logging protocol for the fault-tolerance [7].

- **Page-home Log Exploitation Scheme (PLES)**

It uses the only diff logs of page-homes in FT-SDSM with a remote logging protocol.

- **Log-home Log Exploitation Scheme (LLES)**
  It exploits the only diff logs of log-homes in FT-SDSM with a remote logging protocol.

- **Hybrid Log-usable Scheme**
  It uses all applicable diff logs of both page-homes and log-homes in FT-SDSM with a remote logging protocol.

## 5.2 Service Thresholds

In Section 4, we explain some service thresholds (i.e., $S_{size(H)}$, $S_{depth(H)}$ and $S_{depth(LH)}$) described in Section 4. To analyze the effect of the service thresholds in performance and to find the set of the optimal values, we have performed some experiments with the various values of thresholds for each application.

**Optimal Threshold of $S_{size(H)}$ :** We considered only PLES for adjusting $S_{size(H)}$. We fixed $S_{depth(H)}$ with the highest value in such a way it will not affect $S_{size(H)}$. In Ocean, 3D-FFT and Radix, we adjusted $S_{size(H)}$ to 3072 bytes to make the maximal use of diff logs. In addition, the value of $S_{size(H)}$ higher than 3072 bytes is not helpful in performance. In Water-Spatial and Raytrace, we adjusted $S_{size(H)}$ as 1024 bytes.

**Optimal Thresholds of $S_{depth(H)}$ and $S_{depth(LH)}$ :** We first began by adjusting $S_{deptg(H)}$ for PLES. We fixed $S_{size(H)}$ with the corresponding values above. We then, adjusted $S_{depth(H)}$ to use the diff logs as many as possible without degrading the performance in searching for the diff logs. The results showed that 200 is appropriate for $S_{depth(H)}$ in Ocean and Radix, and 100 is appropriate for $S_{depth(H)}$ in Water-Spatial, Raytrace and 3D-FFT. Secondly, we adjusted $S_{depth(LH)}$ for LLES. We have obtained the results from our experiment that most of all version differences to be searched are within 100 for all applications.

## 5.3 Ratio of Usable Diff Logs

**Table 2. Usability Ratio of Diff Logs**

| Scheme | PLES | LLES | Hybrid scheme |
|---|---|---|---|
| **Ocean** | 1103/5619 (19.68%) | 384/5613 (6.84%) | 1388/5607 (24.76%) |
| **Water-Sp** | 21868/21882 (99.93%) | 2608/21883 (11.92%) | 21869/21884 (99.93%) |
| **Raytrace** | 8496/12928 (65.72%) | 687/12931 (5.51%) | 8211/12586 (65.24%) |
| **3D-FFT** | 5/13223 (0.04%) | 960/13223 (7.26%) | 965/13223 (7.30%) |
| **Radix** | 186/4313 (4.31%) | 204/4308 (4.72%) | 377/4314 (8.73%) |

**(Usability Ratio : the number of pages serviced with diff logs over total number of page faults in a node)**

Table 2 provides the usability ratio that a node can update its invalid pages with the diff logs. Three schemes are compared for the impact of diff logs. For LLES, the usability ratio between five applications is not highly different. For PLES, Water-Spatial and Raytrace have achieved the high usability ratio compared to the other applications. In a hybrid log-usable scheme, we can see from Table 2 that each log-usable ratio of five application is higher than about 7%.

## 5.2 Message Overhead

The message overhead represents the network traffic while the page faults contribute to the memory miss idle time, SDSM overhead and time spent in OS kernel. In general, the network overhead is categorized into two factors, total amount and number of messages over the network.

**Table 3. Message Overhead over the Network**

| Scheme | Remote Logging | PLES | LLES | Hybrid Scheme |
|---|---|---|---|---|
| **Ocean** | 44952 | 44977 | 41802 | 41789 |
| **Water-Sp** | 175075 | 175069 | 154208 | 154207 |
| **Raytrace** | 103223 | 103012 | 97393 | 96727 |
| **3D-FFT** | 105785 | 105785 | 98105 | 98108 |
| **Radix** | 34478 | 34506 | 32843 | 32945 |

**(a) Number of Messages**

| Scheme | Remote logging | PLES | LLES | Hybrid scheme |
|---|---|---|---|---|
| **Ocean** | 184 | 160 | 171 | 149 |
| **Water-Sp** | 717 | 186 | 631 | 163 |
| **Raytrace** | 423 | 307 | 399 | 304 |
| **3D-FFT** | 433 | 433 | 401 | 401 |
| **Radix** | 147 | 137 | 134 | 131 |

**(b) Amount of Messages (MB)**

Table 3 describes the message overhead over the network under five different schemes. When a remote logging scheme serves as a baseline for comparison, the number of messages is reduced by 4-12% and the amount of messages is minimized by 11-78% for the hybrid log-usable scheme in all applications. This reduction mainly comes from the high usability of diff logs. The experiment shows that our proposed schemes have very large positive impact in performance.

## 5.3 Total Execution Time

Figure 4 describes a detailed breakdown of each normalized execution time for five applications. For the comparison purpose, we used total execution time of FT-SDSM with a remote logging protocol as a performance baseline. Each experiment is broken down into four categories, from top to bottom as follows: (i) time spent in computation and OS kernel, (ii) time spent in waiting for (ii) lock or (iii) barrier and (iv) time spent in waiting for updating the invalid pages. The five different schemes used for each application are as follows. (i) no logging scheme, (ii) remote logging scheme, (iii) PLES, (iv) LLES and (v) hybrid log-usable scheme.

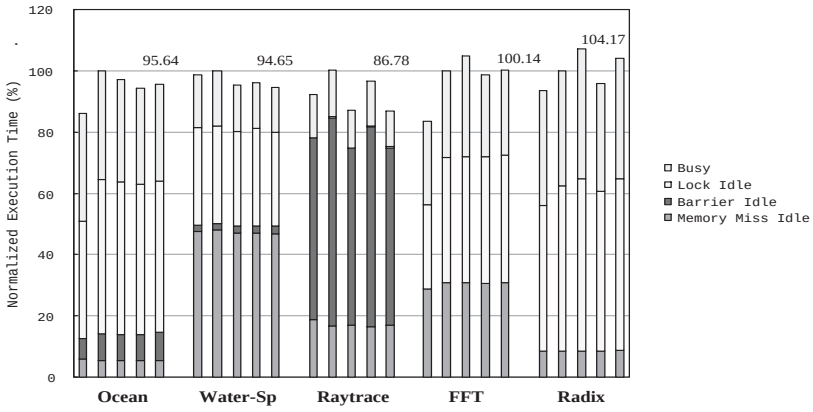


**Figure 3. Impacts of Practical Schemes for Logs**

Over all, our proposed schemes show good performance improvement compared to the remote logging scheme. For the hybrid log-usable scheme, we can see that total execution time is reduced by 4-5% in Ocean, 6-7% in Water-Spatial and 13-14% in Raytrace whereas no improvement is achieved for FFT and Radix. This performance degradation in FFT and Radix is due to the low usability ratio of the diff logs. In these applications, only to apply LLES rather PLES is more desirable for the higher performance. Especially, in Water-Spatial and Raytrace, the hybrid log-usable scheme shows a higher performance than no logging scheme. This is so much promising for our FT-SDSM with a remote logging protocol to challenge the previous home-based SDSMs even if they don't support the fault-tolerant capabilities. This figure shows that the high reduction of total execution time is due to the reduction of page fault and lock synchronization times. Especially, in Raytrace, the reduction of the lock synchronization time is mainly due to the reduction of the page fault time.

## 6. Conclusions

This paper proposes two new schemes based on a remote logging protocol. To our knowledge, it is the first study to use the logged data for higher performance during the failure-free execution. Our experiment results show that our proposed schemes substantially reduce the communication overhead. The desirable results are due to the reduction of the times stalled for updating the invalid pages and the times spent for the synchronization operations. Thus we conclude that our practical schemes for the logs are quite effective on FT-SDSM with a remote logging protocol and they are sufficiently applicable to reduce the memory miss idle time in some fault-tolerant SDSM with a volatile logging.

## References

[1] K.Li and P.Hudak, Memory Coherence in Shared Virtual Memory Systems, *ACM Transactions on Computer Systems, 7*(4), 1989, 321-359.

[2] G.G.Richard III and M.Singhal, Using Logging and Asynchronous Checkpointing to Implement Recoverable Distributed Shared Memory, *Proc. of 12th IEEE Symp. on Reliable Distributed Systems (SRDS-12)*, 1993, 58-67.

[3] G.Suri, B.Janssens, and W.K.Fuchs, Reduced Overhead Logging for Rollback Recovery in Distributed Shared Memory, *Proc. of 25th Int'l Fault-Tolerant Computing Symp. (FTCS-25)*, 1995, 279-288.

[4] M.Costa, P.Guedes, M.Sequeira, N.Neves, and M.Castro, Lightweight Logging for Lazy Release Consistent Distributed Shared Memory, *Proc. of 2nd USENIX Symp. Operating Systems Design and Implementation (OSDI-2)*, 1996, 59-74.

[5] A.Kongmunvattana and N.-F.Tzeng, Coherence-Centric Logging and Recovery for Home-Based Software Distributed Shared Memory, *Proc. of 1999 Int'l Conf. Parallel Processing (ICPP '99)* , 1999, 274-281.

[6] F.Sultan, T.D.Nguyen and L.Iftode, Lazy Garbage Collection of Recovery State for Fault-Tolerant Distributed Shared memory, *IEEE Transactions on Parallel and Distributed Systems, 3*(10), 2002, 1085-1098.

[7] S.Y.Park, Y.J.Kim, and S.Y.Maeng, Remote Logging for Fault Tolerant Software Distributed Shared Memory, *Proc. of 30th Korea Information Science Society Conf. (KISS-30)*, 2003, 70-72.

[8] Y.Zhou, L.Iftode, and K.Li, Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Virtual Memory Systems, *Proc. of 2nd USNIX Symp. Operating Systems Design and Implementation (OSDI-2)*, 1996, 75-88.

[9] S.C.Woo, M. Ohara, E. Torrie, J.P.Singh, and A.Gupta, The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. of 22nd Int'l Symp. Computer Architecture (ISCA-22)*, 1995, 22-36.

[10] P.Keleher, A.L.Cox, and W.Zwaenepoel, Lazy Consistency for Software Distributed Shared Memory, *Proc. of 19th Int'l Symp. Computer Architecture (ISCA-19)*, 1992, 13-21.

[11] K.Gharachorloo, D.Lenoski, J.Laudon, P. Gibbons, A. Gupta, and J. Hennessy, Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *Proc. of 17th Int'l Symp. Computer Architecture (ISCA-17)*, May. 1990.

[12] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merrit, E. Gronke, and C. Dodd, The Virtual Interface Architecture, *IEEE Micro, 18*(2), 1998, 66-75.

[13] http://www.myrinet.com