

Prototyping PHLOX, A High Performance Transaction Processing System on a Workstation Cluster with Shared Disks

Kyungoh Ohn

Raechan Hwang

Haengrae Cho

Department of Computer Engineering, Yeungnam University
Kyongsan, Kyongbuk 712-749, Korea
Email: hrcho@yu.ac.kr

Abstract

Coupling multiple workstations for high performance transaction processing has become increasingly attractive for reasons of capacity, cost, and availability. PHLOX is a prototypical transaction processing system running on the workstation cluster with shared disks. PHLOX supports a message oriented file system for database sharing, global cache coherency and concurrency control, affinity-based transaction routing, and sophisticated database recovery protocol. This paper describes the current prototype of PHLOX and discusses of the major design decisions that went into its construction. The lessons learned from this prototype and its predecessors are also presented.

1 Introduction

There has been an increasing growth of *online transaction processing* (OLTP) applications, such as communication applications, stock trading, banking, and so on [2]. Furthermore, a number of public services will be supported via high-speed information networks, and electronic commercial transactions between companies will be more activated. Note that a personal computer or a workstation cannot support the complex OLTP applications, and it must be very expensive to use high performance computer systems, such as supercomputer and MPP. We believe that a parallel transaction processing on the workstation cluster will be the reasonable solution to develop high performance OLTP systems.

There are two primary flavors of workstation coupling to parallel transaction processing: *shared nothing* and *shared disks* [1, 9]. In shared nothing, each data partition is owned by a single workstation, and only the workstation can directly read and modify its partition. Cache and lock management are done locally by each partition server. In shared disks, all workstations are connected via a high-speed network and share a common database at the disk level. Worksta-

tions need to hold locks on shared data, both when active transactions need the data and when the data is cached at the workstations. Such shared access requires distributed locking and cache management. Although research and system developments have concentrated on shared nothing approach [9, 15], we feel the shared disks approach offers a number of advantages, such as dynamic load balancing, availability, and seamless integration, that make it attractive for high performance transaction processing.

In this paper, we present design overview of a prototypical high performance transaction processing system, named PHLOX, which can support the parallel transaction processing on the workstation cluster with shared disks. The PHLOX project is supported by the Korean Ministry of Information and Communication since 1999. Figure 1 describes the architecture of PHLOX, and the significant features of PHLOX can be summarized as follows.

- PHLOX is scalable in the sense that it can share data with a remote file system.
- PHLOX supports direct data transfer between memory buffers of each workstation; hence, it can reduce the number of disk IOs.
- PHLOX supports a distributed fine-granularity locking and a global cache coherency schemes (STEAL and not FORCE buffer management [10]) that extends the notion of primary copy authority [15] and ARIES/SD [11].
- PHLOX implements a transaction router that supports affinity-based transaction routing.
- PHLOX supports a logging and recovery software that extends ARIES/SD.

In the following sections, we discuss each feature in detail and present the lessons learned from prototyping PHLOX and its predecessors.

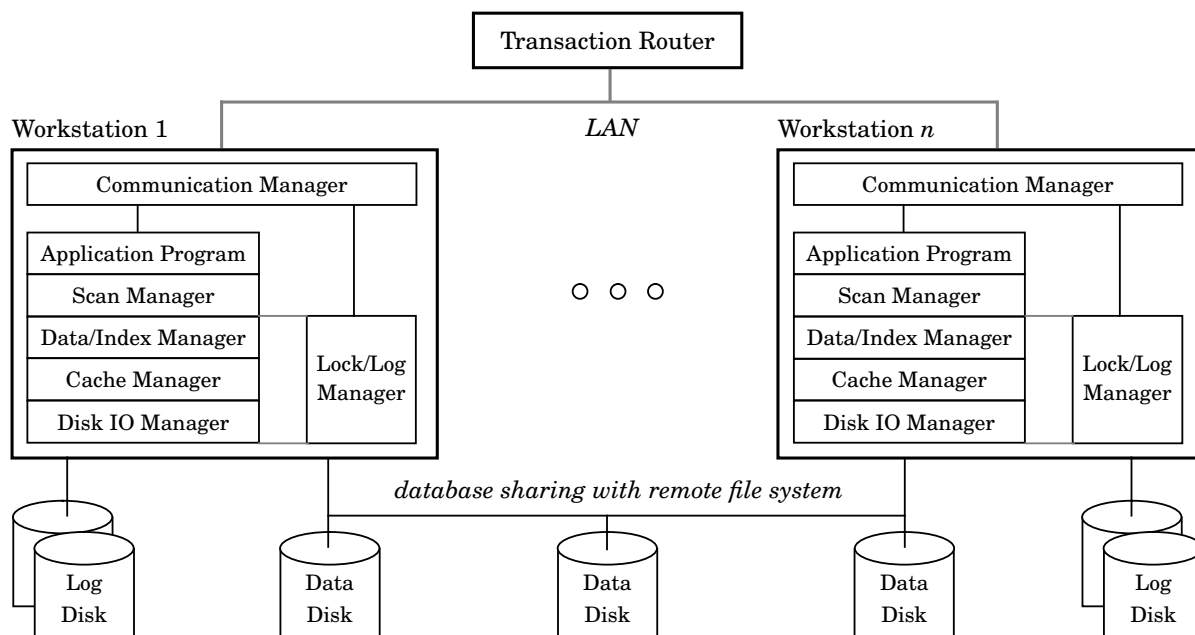


Figure 1: Architecture of PHLOX

2 Database Sharing

In a workstation cluster, each workstation can share database either with the special IO devices such as “shared SCSI”, or with the message-oriented file systems that communicate with workstations by means of message passing. While the former approach can guarantee faster disk IOs, the number of workstations that can be connected to the same disk would be limited.

The representative protocol for message-oriented file system is Sun’s NFS. However, using the NFS protocol for database sharing incurs two critical drawbacks. First, NFS writes are known to be slow. Because it is a stateless server protocol built on top of UDP, a write operation to a remotely mounted NFS file is flushed to disk before the request is acknowledged. Next, in the NFS version 3 protocol that increases write throughput by *safe asynchronous write*, it is not possible to determine when dirty data due to write operations are pushed out to the remote disk. If a database process at the NFS server is terminated abnormally due to system failure, the asynchronous write should not incur any problems since the most recent version of data can be recovered using update log records. However, the recovery process will not be performed when the NFS server is crashed after the normal termination of the database process. This means that the dirty data of the NFS server could be lost in this case.

With regard to this viewpoint, PHLOX implements its own message-oriented file system for database sharing. The basic idea is to support transparent file access similar to the NFS protocol and to implement syn-

chronous IOs with minimal overhead. Figure 2 shows the process architecture of PHLOX for implementing synchronous IOs.

For each mounted disk, the workstation allocates one or two disk IO processes to implement non-blocking IOs. There are three types of disk IO processes according to the disk type. If the disk is local, the workstation creates a *local IO process* and an *IO agent process*. A *remote IO process* is created for remote disk. The local IO process accesses local disk directly to execute read or write operations from local database applications. On the other hand, the remote IO process sends disk IO messages to the IO agent process of the remote disk’s server workstation. Then the IO agent process executes the requested disk IO operations to its local disk.

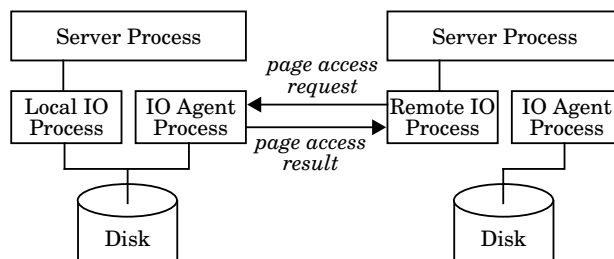


Figure 2: Supporting Synchronous IOs in PHLOX

Note that high level database applications do not need to distinguish the type of disks that they access. The same disk IO interface is provided to the applications, and thus PHLOX can support transparent file

accesses. Furthermore, by eliminating the direct access to remote disk, PHLOX can also support synchronous IOs rather cheaply. To reduce the number of disk IOs, PHLOX implements a global cache coherency scheme described in the next section.

3 Cache Coherency and Concurrency Control

Each workstation has its own buffer pool and caches database pages in the buffer. Caching may substantially reduce the number of expensive and slow disk IOs by utilizing the locality of references. However, since a particular page may be simultaneously cached in different workstations, modification of the page may invalidate copies of that page in other workstations. This necessitates the use of *cache coherency scheme* so that the workstations always see the most recent version of database pages [5, 6, 11].

The complexity of cache coherency scheme critically depends on the locking protocols used for concurrency control. There are two approaches of locking in the shared disks: centralized and distributed. In the *centralized* approach, it is assumed that there is a “global lock manager” (GLM), that not only provides global locking functions for every workstation, but also tracks which workstation has a valid copy of a page [4, 11]. While this approach is easy to implement, the GLM may be a communication bottleneck and a single failure point of the entire system. In the *distributed* approach, the database is divided into logical partitions and each workstation is assigned the synchronization responsibility, or *primary copy authority* (PCA), for one partition [15]. The distributed approach has the obvious advantage that lock requests for the local partition can be handled without communication overhead and delay. Other lock requests are sent to authorized workstation holding the PCA for the respective partition.

PHLOX adopts the following strategies for maintaining cache coherency and concurrency control.

- Maintaining cache coherency with extended ARIES/SD [11]
- Distributed locking approach with PCA

To ensure the cache coherency, PHLOX implements the extended ARIES/SD [11]. The ARIES/SD prohibits simultaneous updates on the same page by *physical lock* (P lock). To update a page, a workstation is required to become the *owner* of that page by requesting P lock with exclusive mode; when a different workstation tries to update the page or to read the recent version of the page, it must be sent the most recent copy of the page by the current owner. By permitting only one P lock per page with exclusive mode, the

physical consistency of a page can be satisfied. Furthermore, when a workstation requests an updated page cached by another workstation, the ARIES/SD supports direct page transfer between memory buffers of each workstation. Note that this way can reduce the number of disk IOs compared to the way of writing an updated page to the disk and then reading the page from the disk, which was implemented in the Oracle Parallel Server [14].

PHLOX extends the ARIES/SD in two-folds. First, PHLOX introduces the new notion of *record owner*. Specifically, PHLOX allows a workstation to receive a page only if the workstation does not cache the recent version of a *record* to be accessed; in contrast, the ARIES/SD makes a workstation be sent a page if the workstation does not cache the recent version of the *page*. The detailed algorithm of record owner can be found in [5]. Next, PHLOX supports *lock caching* so that locks can be cached by a workstation even after the requesting transaction has committed [4]. Then the workstation can handle lock requests of later transactions locally.

PHLOX supports a record-level locking to preserve serializability of transaction processing. Unlike the ARIES/SD where a GLM maintains all the lock information, PHLOX implements distributed locking approach based on the PCA to reduce message traffic for lock request and lock response. The PCA is allocated for each disk volume, and a specific workstation (PCA workstation) has a role to maintain lock information for each PCA. For the performance reason, it is recommended that a workstation should have PCAs of local disk volumes. Every workstation replicates PCA information of [volume identifier, PCA workstation of the volume]. Since lock information is distributed, distributed deadlocks can be occurred. PHLOX implements a timeout approach to resolve the distributed deadlock. Local deadlocks are handled using the cycle checking on the wait-for-graph.

Figure 3 shows a basic communication protocol for lock request and page transfer. Each component of Figure 3 is implemented as a UNIX thread. An *application thread* sends a lock request message to the corresponding PCA workstation, and it waits until the timeout. The lock request message consists of [transaction identifier, lock mode, record identifier, PageLSN of a page that includes the record].¹ The *dispatcher* of PCA workstation receives the lock request message and requests the locking operation to the PCA lock manager. If the lock is granted, the PCA lock manager sends a pair of [transaction identifier, PageLSN] to the cache manager. The *cache manager* checks whether the

¹ *PageLSN* of a page contains the log sequence number (LSN) of the log record that describes the latest update to that page [10]. By comparing PageLSNs of two pages, we can determine which page is more recent.

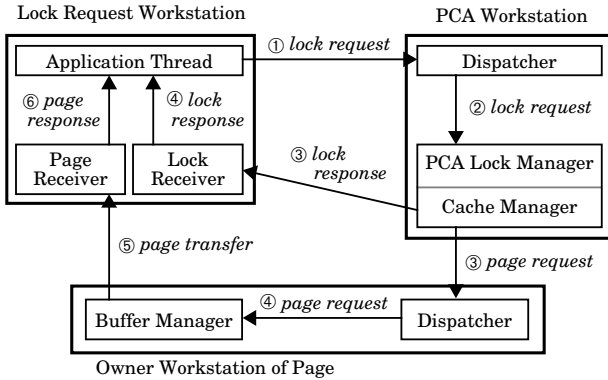


Figure 3: Communication Protocol for Locking

requesting workstation caches the most recent version of the page. According to the result of cache check, the cache manager returns the lock response message to the requesting workstation and sends a page request message to the owner workstation of the page. The lock response message consists of [lock result, PageLSN]. When a *lock receiver* receives the lock response message, it resumes the application thread by forwarding the lock response message. The application thread compares the returned PageLSN with the PageLSN that it sent in the lock request message. Suppose PageLSN_1 implies the former PageLSN and PageLSN_2 implies the latter one. Then the following three cases can be happened.

1. $\text{PageLSN}_1 = 0$: There are no workstations that cache the most recent version of the page. In this case, the application thread should access the page in the shared disk.
2. $\text{PageLSN}_1 = \text{PageLSN}_2$: The requesting workstation caches the most recent version of the page. So the application thread can access the page from the local cache.
3. $\text{PageLSN}_1 > \text{PageLSN}_2$: Other workstations cache the most recent version of the page. The application thread should wait again until the page is sent from other workstation as Figure 3 shows. The *page receiver* will resume the application thread when the requested page is available.

In case 1 and case 3, the lock request workstation sends a notification message to the PCA workstation when it caches the most recent version of the page. If the requested lock mode was exclusive, the PCA workstation registers the lock request workstation as a new owner of the page.

4 Transaction Routing

PHLOX includes its own transaction router. The role of transaction router is two-folds. First, the transaction

router monitors the execution status of every workstation in the cluster. Specifically, the transaction router checks whether each workstation is active, failed, or in performing recovery tasks. This information is used to determine the type of database recovery. We describe the details of database recovery using the transaction router in Section 5.

The next role of transaction router is to allocate workstations for processing user transactions. The goal of transaction routing in PHLOX is to support *affinity-based transaction routing* [16] that exploits the reference behavior of transaction types to assign transactions with an affinity to the same database portions to the same workstation. As a result, transactions running on different workstations should mainly access disjoint portions of the database. Note that the inter-workstation communication is occurred due to lock request, page transfer, or remote disk IOs. If each workstation has PCAs on its local disks and if the transaction router allocates transactions according to their PCA information, the inter-workstation communication should be minimized.

We now present the details of affinity-based transaction routing in PHLOX. The transaction router maintains the PCA information and the load information of each workstation. Once a user requests a transaction execution, the transaction router checks the reference behavior of the transaction and assigns it to the workstation that has majority PCAs of the transaction. We call the workstation as a *major PCA workstation*. If the major PCA workstation is not overloaded, the transaction is allocated to the workstation. PHLOX implements the “half-and-half” algorithm [3] to determine whether a workstation is overloaded. Specifically, an workstation is overloaded if more than 50% of the active transactions in the workstation are mature² but blocked due to locking. To achieve this, the transaction router monitors continuously the number of active transactions, mature transactions, and blocked transactions for each workstation.

If the major PCA workstation is overloaded, the transaction router selects a not-overloaded workstation as a *candidate* workstation. Thereafter, the new transactions on the overloaded major PCA workstation are allocated to its candidate workstation. By fixing a specific candidate workstation for each overloaded workstation, it is possible to maximize the buffering effect and the possibility of lock retention [4]. If the candidate workstation becomes also overloaded, the transaction router selects any workstation randomly for allocating new transactions.

²An active transaction is said to be *mature* after it has completed 25% of its estimated number of lock requests [3].

5 Database Recovery

PHLOX supports database recovery for various types of failures, such as transaction failure, single workstation failure, and multiple workstation failures. Since PHLOX implements ARIES [10] to recover the transaction failure due to deadlock, we will not discuss it in the paper. In this section, we first present the way of determining recovery type. Then we describe the recovery algorithm for single workstation failure and multiple workstation failures.

5.1 Determining Recovery Type

The transaction router monitors the state of each workstation by sending a monitoring message continuously. There are five states of a workstation: **not-active**, **failed**, **active**, **restarted**, and **suspended**. Figure 4 shows the state transition diagram of each workstation.

The transaction router changes the state of an **active** workstation to **failed** if the workstation does not respond for the monitoring message. When the **failed** workstation restarts, it performs the following types of database recovery.

1. There is only one **failed** workstation and others are in **active** state. When the **failed** workstation restarts, its state is changed to **restarted** and it performs *simple recovery*.
2. There are several **failed** workstations and one of them restarts. In this case, it is not possible to access log records and locking information of other **failed** workstations, and thus the database recovery cannot be performed. So the restarted workstation is in **suspended** state and wait until all of **failed** workstations restart.
3. There is only one **failed** workstation and there are some **suspended** workstations. When the **failed** workstation restarts, its state is changed to **restart** and it performs *complex recovery*. After the redo phase of complex recovery, the **suspended** workstations are changed to **restarted** and performs their own local undo.

5.2 Simple Recovery

In simple recovery, the **restarted** workstation recovers updates on database pages where it has PCA or it is registered as an owner. The simple recovery procedure consists of three phases: *analysis*, *redo*, and *undo*.

The goal of the analysis phase is (1) to construct the *dirty pages table* (DPT) and (2) to determine the *RedoLSN*, where the redo phase starts scanning the merged logs. The **restarted** workstation first initializes the DPT and the transaction table by accessing the

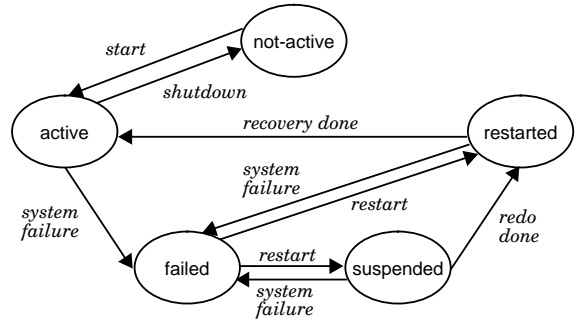


Figure 4: State Transition Diagram

most recent checkpoint log records. Then the workstation initializes the owner table by receiving information of pages where it is registered as an owner from other workstations. The RedoLSN is set to the minimum for recovery LSNs [11] of all the pages in the DPT and the owner table.

As Figure 1 shows, each workstation has separate log disks and the log disks store only the log records of the corresponding workstation. So it is required to merge log records of every log disk for redo phase. The **restarted** workstation receives sequentially N log pages that include log records of which LSN is larger than the RedoLSN from other workstations. The value of N is determined by the size of local buffer and the number of workstations. After receiving log pages for every other workstations, the **restarted** workstation constructs a *min heap* of log records according to their LSNs. Then the **restarted** workstation performs the redo phase by scanning log records sequentially from the min heap. To reduce the communication delay of log pages, the next N log pages are prefetched from other workstations during the log scan.

During the redo phase, a *transaction table* is also created. The transaction table contains a list of loser transactions to be rolled back and *UndoLSN* of each transaction. Among the transactions which has been executed at the **restarted** workstation the loser transactions are ones of which update log records are appeared in the merged logs, but the commit or the abort log records are not. The UndoLSN of each transaction is the LSN of the latest log record written by the transaction. The undo phase rolls back the loser transactions in a single sweep of the merged logs. This is done by continually taking the maximum of the UndoLSNs of loser transactions.

5.3 Complex Recovery

The analysis phase and the redo phase of the complex recovery are similar to those of the simple recovery. The difference is that the **restarted** workstation redoes updates on every database page in the complex recovery. This means that the redo phase should take

longer time than that of simple recovery.

Furthermore, `active` workstations can continue their executions in the simple recovery. However, it is not true in the complex recovery. This is because the `restarted` workstation does not perform the cache coherency procedure at the redo phase. Suppose that a same page is cached by the `restarted` workstation and the `active` workstation. Then lost updates could be happened if the `restarted` workstation pushes out the page to disk after the `active` workstation writes its updated page to disk. In simple recovery, this abnormal scenario should not be occurred since there are no pages that are accessed simultaneously by the `restarted` workstation and `active` workstations.

The undo phase is also different somewhat in the complex recovery. Unlike the simple recovery, where the loser transactions include transactions executed at the `restarted` workstation only, the loser transactions in the complex recovery might be executed at any workstation. So the undo phases should be performed *in parallel* by the individual workstations. The `restarted` workstation constructs a list of [transaction identifier, UndoLSN] for loser transactions and then transfers part of the list to each workstation. Each workstation can roll back the loser transactions similar to the way of single workstation failure.

6 Performance Experiments

To evaluate the performance of PHLOX, we have implemented a TPC-B benchmark program [8] in a workstation cluster with varying number of workstations. Each workstation has a SUN's SuperSparc CPU, and one or two disks are attached for the workstation. The total number of disks is fixed to 5. Figure 5 shows the benchmarking results of PHLOX with varying number of workstations. In the figure, the Y-axis represents the number of total transactions processed per second. The X-axis represents the number of concurrent transactions executed for each workstation.

The transaction throughput scales up in proportion to the number of workstations. This is due to the effect of both transaction routing and reference behavior of each transaction. Specifically, the results of Figure 5 come from the uniform workload, where all transactions access data uniformly throughout the entire database. Since every workstation has one or two disks respectively and each disk stores equal number of data records, most of user transactions are allocated to the major PCA workstation. As a result, inter-workstation communication due to page transfer and lock request can be minimized, and thus linear scale-up can be achieved by increasing the number of workstations.

The performance results are somewhat different in high contention workload, where all transactions have

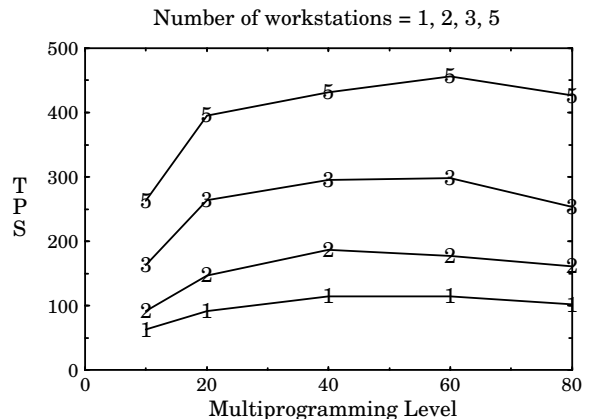


Figure 5: TPC-B Benchmark Results of PHLOX

the same data access skew and the degree of data contention is very high as a result. In this workload, performance improvements are not significant even the number of workstations increase. This is primarily due to lock request overhead to PCA workstations and page transfer overhead between workstations.

7 Related Work

In this section, we compare the distinguished features of PHLOX with other shared disks systems, even though some of them are supposed to multiprocessing architectures not workstation cluster. The previous shared disks systems include Digital's VAXcluster [7], IBM's S/390 Parallel Sysplex [13], IMS Data Sharing version (IDS) [17], and Oracle Parallel Server (OPS) [14]. Table 1 summarizes the differences of PHLOX and the previous shared disks systems.

Most of shared disks systems support the record-level locking except IDS. S/390 implements centralized locking approach by maintaining global lock information to the specialized hardware device, named *coupling facility* (CF). The MVS central processing complexes are connected to the CFs with CF channel and high-speed fiber optic links. Other systems adopt distributed locking approach. Among them, VAXcluster and OPS assume a distributed lock manager facility provided by the operating system. PHLOX and IDS implement their own distributed locking facilities. IDS supports distributed locking with the "pass-the-buck" protocol [17]. However, the cluster can connect only eight workstations due to the protocol complexity. On the other hand, since PHLOX implements distributed locking with PCA, it can connect any number of workstations.

VAXcluster, OPS, and IDS should suffer from frequent disk IOs since updated pages are transferred between workstations via shared disks. Furthermore, VAXcluster and IDS implements FORCE buffer management policy, where a transaction is not allowed to

Table 1: Comparison of Shared Disks Systems

	VAXcluster	S/390	OPS	IDS	PHLOX
lock granularity	record	record	record	page	record
global locking	distributed	centralized	distributed	distributed	distributed
page transfer	disk	shared memory	disk	disk	memory-to-memory
database sharing	message passing	shared SCSI	shared SCSI	shared SCSI	message passing
buffer management	Force	No Force	No Force	Force	No Force

commit until all of the pages modified by the transaction are pushed out to disks. S/390 supports global buffer pools in CF, which can be shared to every workstation. This is referred to as the *shared intermediate memory* architecture [16]. While it can reduce the complexity of cache coherency scheme, additional message traffic should be required to transfer updated pages from local buffer of each workstation to the global buffer pools. S/390, OPS, and IDS implement database sharing using the specialized hardware such as shared SCSI, and thus they may suffer from limited scalability as we have described in Section 2.

8 Concluding Remarks

In this paper, we have presented the design issues and implementation details of a prototypical high performance transaction processing system, which we call PHLOX. PHLOX can support parallel transaction processing with workstation clusters connecting through high speed LANs and sharing common databases.

Currently, we are in the final stage of developing a SQL processor on top of PHLOX. The SQL processor implements parallel join algorithms using the dynamic load balancing in the workstation cluster. To achieve this, we have proposed several parallel join algorithms in shared disks and evaluated the performance of proposed join algorithms using the simulation approach [12]. The performance results show that the parallel hybrid hash join algorithm outperforms other algorithms under a number of database workloads and system configurations. On the basis of performance results, we are currently implementing the parallel hybrid hash join algorithm on top of PHLOX.

References

[1] M. Abdelguerfi and K. Wong (ed.), *Parallel Database Techniques*, IEEE Computer Society Press (1998).

[2] P. Bernstein and E. Newcomer, *Principles of Transaction Processing*, Morgan Kaufmann Publishers, Inc. (1997).

[3] M. Carey, S. Krishnamurthi, and M. Livny, "Load Control for Locking: The 'Half-and-Half' Approach," in: *Proc. 9th PODS Symposium* (1990) 72-84.

[4] H. Cho, "Cache Coherency and Concurrency Control in a Multisystem Data Sharing Environment," *IEICE Trans. Inf. & Syst.* E82-D(6) (1999) 1042-1050.

[5] H. Cho and J. Park, "Maintaining Cache Coherency in a Multisystem Data Sharing Environment," *J. Syst. Architecture* 45(4) (1998) 285-303.

[6] A. Dan and P. Yu, "Performance Analysis of Buffer Coherency Policies in a Multisystem Data Sharing Environment," *IEEE Trans. on Parallel and Distributed Syst.* 4(3) (1993) 289-305.

[7] R. Davis, *VAXcluster Principles*, Digital Press (1993).

[8] J. Gray (Ed.), *The Benchmark Handbook*, Morgan Kaufmann Pub. (1993).

[9] L. Miller, A. Hurson, and S. Pakzad (ed.), *Parallel Architectures for Data/Knowledge-Based Systems*, IEEE Computer Society Press (1995).

[10] C. Mohan *et al.*, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Trans. on Database Systems* 17(1) (1992) 94-162.

[11] C. Mohan and I. Narang, "Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," in: *Proc. 17th VLDB Conf.* (1991) 193-207.

[12] A. Moon, K. Ohn, and H. Cho, "Performance of Dynamic Load Balanced Join Algorithms in Shared Disk Parallel Database Systems," in: *Proc. 7th IEEE Workshop on FTDCS* (1999) 176-181.

[13] J. Nick *et al.*, "S/390 Cluster Technology: Parallel Sysplex," *IBM Syst. J.* 36(2) (1997) 172-201.

[14] *Oracle7 Parallel Server Concepts and Administration*, Part No. A42522-1 (1996).

[15] E. Rahm, "Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems," *ACM Trans. on Database Syst.* 18(2) (1993) 333-377.

[16] P. Yu and A. Dan, "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture," *IEEE Trans. on Knowledge and Data Eng.* 6(5) (1994) 764-786.

[17] P. Yu *et al.*, "On Coupling Multi-Systems Through Data Sharing," *Proc. of the IEEE* 75(5) (1987) 573-587.