Design and Analysis of Clusters with Single I/O Space*

Roy S. C. Ho[1], Kai Hwang[1, 2], and Hai Jin[1,2]
The University of Hong Kong[1] and University of Southern California[2]
Email: scho@csis.hku.hk, kaihwang@usc.edu, hjin@eee.hku.hk

Abstract - Support of Single System Image (SSI) services is the main approach that enables better utilization of PC/workstation clusters. Some SSI services can be easily built with the support of other low-level, elementary, SSI services. In this paper, we describe a Single I/O Space architecture for achieving a SSI at the I/O subsystem level. Furthermore, we demonstrate how the Single I/O Space can facilitate the development of other key SSI services. Typical SSI services which can benefit from the Single I/O Space include single file hierarchy, single memory space, checkpointing systems and single process space with process migration facilities. Benchmark performance results show that our design achieves both performance and storage size scalabilities that are essential to building I/O-intensive clusters.

Keywords: Cluster computing, Single system image, parallel I/O, distributed RAID, and single I/O space.

1. Introduction

Clusters have proven their potentials in the area of low-cost, but high-performance, parallel computing. Support of *Single System Image* (SSI) services is the main approach that enables better utilization of clusters in terms of convenience, performance, scalability, and reliability [5][6]. Specifically, cluster computing demands a single I/O space in distributed, I/O intensive, operations. The construction of an efficient I/O framework with the SSI services provided is essential for supporting all kinds of I/O intensive application in scalable cluster environments.

The *Single I/O Space* (SIOS) project is initiated and its aim is to build a SSI with high availability services for disk I/O operations in scalable cluster environments using commodity hardware and software. Our main objective is to obtain a single address space for all data blocks in the cluster with high performance scalability, availability and compatibility with current cluster architectures and applications. Figure 1 illustrates the SSI service provided by the SIOS from the viewpoint of cluster users.

Besides, some important SSI services can be easily built with the support of other low-level, elementary SSI services. It can be shown that the SIOS facilitates the development of a number of key SSI services. In this aspect, SIOS is a basic, but powerful, infrastructure to achieving a SSI cluster.

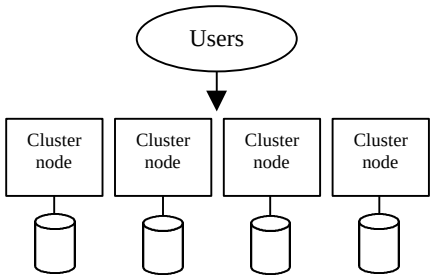
SSI services enable a cluster of PCs or workstations to be used as a single computing unit in a scalable and efficient manner. In other words, SSI services target at getting most out of the theoretical computing power of the cluster nodes, while hiding the physically distributed architectures. Furthermore, the design of SSI services should provide scalable performance when more nodes and components are added into the clusters.

In this paper, we present the SIOS architecture and its impacts on cluster computing. The paper is organized as follows: Section 2 introduces some basic concepts about the SIOS. Section 3 describes the detailed design of the SIOS. Section 4 discusses how the SIOS facilitates the construction of a highly available SSI cluster. Section 5 presents a set of benchmark results on the developed prototype of the SIOS. Finally, a conclusion is made in Section 6 with more future work.

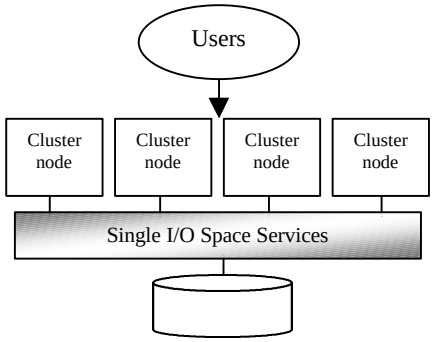
2. Designing SIOS at Device Driver Level

Previous research showed that a SSI for disk I/O operations could be implemented in the user level [4][9][15], the file system level [1][3][7] and the device driver level [8][14]. User level designs are believed to have higher portability and lower implementation cost. However, they usually cannot provide a complete SSI while users still have to use specific application programming interfaces (APIs) and identifiers in order to exploit full functionality of the packages. Furthermore, using system calls and kernel facilities to perform file and network I/O may decrease the performance. File system level designs can have full control in data distribution while providing a complete SSI to the user. However, changing the file system does not guarantee strict compatibility with current applications and the required

* This research was supported by Hong Kong RGC grants HKU 2/96C and HKU 7022/97E, by the Area of Excellence (AOE) information technology development fund at HKU, and by a special research grant from the USC Engineering School.



Without the Single I/O Space, the users see a cluster of workstations, each with their own local I/O devices. Remote I/O can only be done through established services like NFS, with poor scalability and performance degradations.



With the support of Single I/O Space, the users perceive an illusion that they are using a single, large storage. Remote I/O can be done as if the devices are attached locally.

Figure 1: Concept of SIOS Services

development and deployment costs are relatively high. Device driver level designs avoid most problems found in the other levels. They can provide a complete SSI to the file systems and the users while minimizing the file systems modifications. The major drawback of this approach is that it is difficult to control the distribution pattern of files for performance optimization purposes. We chose to design and implement our SIOS prototype in the device driver level.

In this section, we present our design of the SIOS. First, we outline our design objectives. Second, we introduce the primary building block of the SIOS, the Cooperative Device Driver.

2.1. Design Objectives

We chose to design our prototype in the device driver level. Our design has the following objectives.

A single address space for all data blocks in the cluster - We provide the single address space by constructing a *Cooperative Device Driver* (CDD) as the disk device driver for each node in the cluster. These CDDs communicate and cooperate to form the single address space for each data block in the cluster. The file system in each node will perceive an illusion that it is using a large disk, and this illusion is provided by the CDD. We will discuss this in details in Section 2.2 and Section 4.1.

High availability features supported - High availability features can be supported by implementing some established technologies directly in the CDDs. We have implemented RAID-5 and RAID-1 architectures in our prototype, other architectures such as RAID-10 can be easily deployed by modifying a specific program module in our SIOS design. For instance, the SIOS with the RAID-5 configuration can withstand a single disk failure while keeping all the data online.

Performance and Size Scalability - There is no central server in our design and the CDDs only maintain a peer-to-peer relationship with the others. So, scalability can be guaranteed by the server-less design as proposed in [1]. Further, we eliminate the potential bottleneck by striping the data across the distributed RAID disks. We do expect performance gain by parallel accesses of striped files.

The size of the buffer cache in our CDD design will be increased so that we are able to get a remote data block from remote memory, instead of remote disk, if that data block has been cached-in. In this way, all the buffer caches cooperate to form a large cache for the large storage provided by the SIOS. With a fast network, performance gain can be achieved by the cooperative-caching effect [2] and remote disk I/O operations should have the performance at least comparable, or even better than, that of local disk I/O operations.

High compatibility with current cluster applications - This is a "clean" design in the sense that we have limited all implementation in a single level while avoiding the changes in the other parts of the systems. In particular, file system modifications can be minimized. This guarantees high compatibility with current applications and reduces implementation costs.

2.2. Distributed Cooperative Device Drivers

The SIOS consists of CDDs distributed throughout the cluster nodes, each node runs a copy of the CDD as its disk device driver. A CDD is constructed in the device driver and the buffer cache levels. The block device buffer cache is included in the CDD for maintaining data consistency. The CDDs in a cluster cooperate to obtain a single address space for each data block in the cluster, thus providing an illusion of a single, large disk to the file systems above. In this way, remote data retrievals are transparent to upper layers and the operations performed should be the same as if the data is accessed locally.

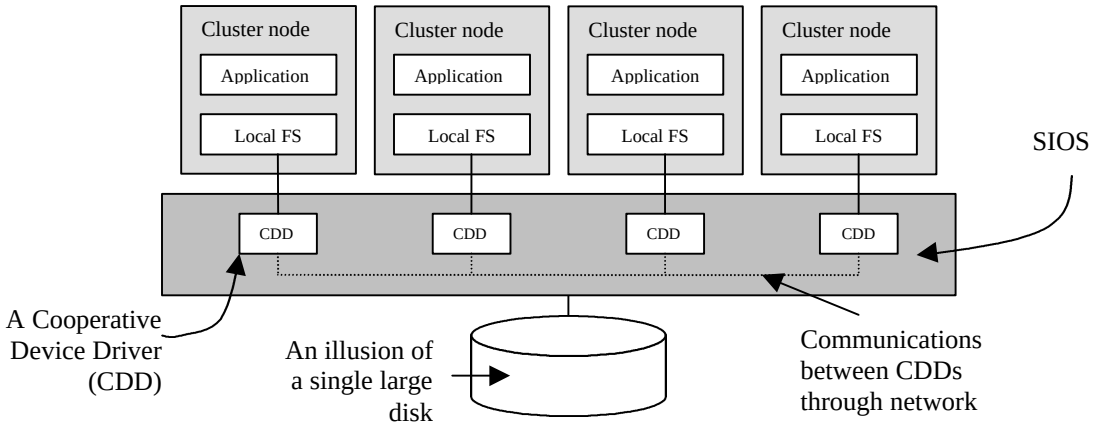


Figure 2: Distributed Cooperative Device Drivers (CDDs)

Figure 2 illustrates the whole picture.

3. CDD Architecture

In this section, we first explain the internal design of a CDD and introduce the device masquerading technique as the enabling mechanism to build the SIOS. Second, the data consistency issues among the distributed disks are discussed.

3.1. Device Masquerading Technique and the Internal Design of a CDD

The *device masquerading technique* is the key concept to design CDDs. The idea is to redirect all I/O requests to the remote disks. The results of the requests, including the requested data, are transferred back to the originating nodes. This mechanism gives an illusion to the operating systems that the remote disks are attached locally.

Fig. 3a illustrates the device masquerading technique. As shown in the figure, each cluster node has only one physical disk attached. The CDDs run cooperatively to redirect I/O requests to the remote disks. Each node perceives the illusion that it has two physical disks attached locally. Fig. 3b shows the internal design of a CDD. The disk manager receives and processes the I/O requests from the remote CDD client modules. The CDD client module redirects the local I/O requests to remote disk managers. The consistency module is responsible for maintaining data consistency among the distributed disks.

A CDD can be configured to run as a disk manager or a CDD client, or both at the same time. This means that there are three possible statuses for a cluster node: (1) a disk manager which contributes its local disk storage to the other nodes, (2) a client which accesses the remote disks donated by the other disk managers, and (3) both of the above.

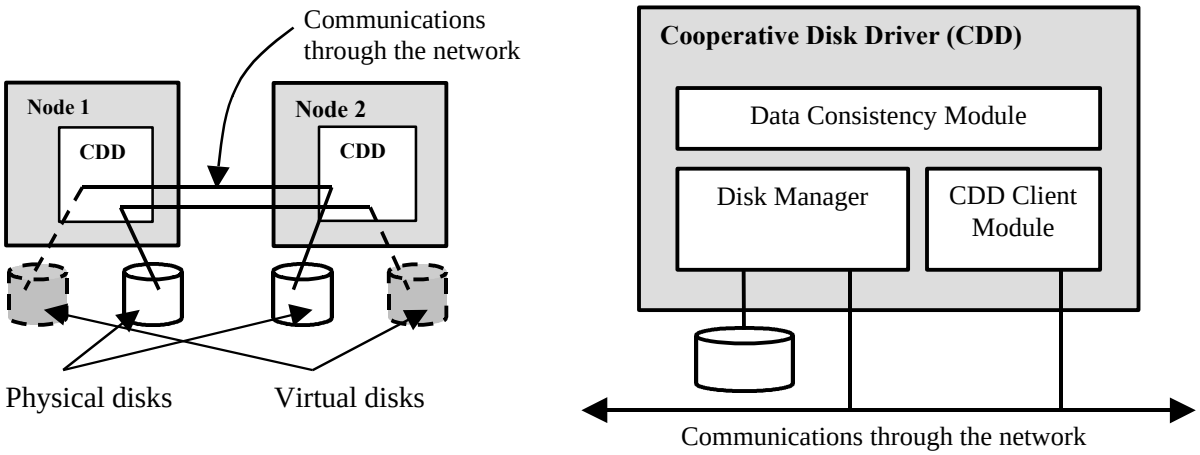
Software RAID architectures can be implemented easily with the support of CDDs to form distributed RAID architectures. All the data transfers and the data consistency issues are handled transparently within the CDDs and the RAID drivers only need to maintain the data distribution and the related policies. Combining the CDD and the RAID concepts shows three advantages. First, RAID architectures add the fault tolerance capability to the I/O subsystem. Second, RAID creates a single address space across all distributed disks. Third, the CDDs support transparent data transfers and data consistency management that make the constructions of distributed RAID architectures much easier.

3.2. Data Consistency Issues

Data consistency problems arise when multiple cluster nodes have cached copies of the same set of disk data blocks. Previous approaches [1][16] addressed this issue in the file system level. In our design, data consistency checking is supported in the disk driver level. This approach simplifies the design and the implementation of distributed file management services. Data consistency among the CDDs is maintained in block level.

A data consistency model similar to that used in the Frangipani file system [16] is used in our CDD design. Multiple-reader/single-writer locks are used for synchronization among the cluster nodes. A write lock of a data block permits a CDD client to read, modify and to cache modified copy in its own memory. A read lock of a data block permits a CDD client to cache a read-only copy of this data block.

A CDD client must get the appropriate lock before it read/write a data block. If a data block is not locked, any CDD client can get the read or write lock of that block immediately. Only one CDD client can hold the write lock of a data block at one time, while read locks may be granted to multiple CDD clients. If one CDD client holds



(a) Device masquerading **(b) CDD architecture**

Figure 3: Architecture Design of the Cooperative Disk Drivers (CDDs)

a write lock while another client is requesting a read/write lock of the same data block, the first client needs to flush its cache entries to the disks. It will release the lock if a write lock is being requested; otherwise the lock will be downgraded to a read lock. If one CDD client holds a read lock while another client is requesting a write lock, the first client will invalidate its cache entries and release the lock.

We introduced a special lock-group table for developing distributed file management services. Each record in this table corresponds to a group of data blocks that have been granted to a specific CDD client with read or write permissions. The locks in each record are granted and released atomically. This lock-group table is replicated among the Data Consistency Modules in the CDDs. A set of query functions is also supported for checking purposes. Based on these facilities, the CDDs guarantee certain file management operations, e.g. creating directories, can be performed in an atomic manner. Distributed file systems developed on top of CDDs can design their own concurrent file access policies, e.g. per-file locking, without taking care of the remote communications involved. In our prototype, the implemented policy is sufficient to carry out the benchmark experiments.

4. Designing SSI Clusters with SIOS

SSI services enable a cluster of PCs or workstations to be used as a single computing unit in a scalable and efficient manner. In this connection, SIOS plays an important role in developing and supporting different SSI services. [5][6][11] give comprehensive descriptions on different SSI services for cluster computing. In the following paragraphs, we explain how some of these SSI services can be supported by, and benefit from, the SIOS service. The services that we discuss include the single file hierarchy, single memory space, checkpointing facilities, single process space, and process migration support.

4.1. Single File Hierarchy

Single file hierarchy is an illusion of a single, huge file system image that transparently integrates distributed disks. In other words, all files in a cluster are stored in a single hierarchy, and they can be accessed through ordinary calls such as open, read, etc. [6]. Apart from the conveniences it offer, the Single File Hierarchy is particularly useful for process migration, where the migrated process can still access the opened files at the same locations in a directory tree.

The SIOS not only gives an illusion of a single disk to the users, but to the local file systems too. This means each local file system will perceive the illusion that it is using a single disk as if the disk is attached locally. Since each file system in the cluster is using the same disk storage provided by the SIOS, a complete, single file hierarchy is formed automatically.

The functionalities of single file hierarchy have already been partially provided by existing distributed file systems [6]. Typical examples include the Network File System (NFS) [13] and the Andrew File System (AFS) [10]. However, their performances usually degrade when the number of clients is getting large. The SIOS provides scalable performance through its server-less design while supporting a complete SSI to the users.

4.2. Single Memory Space

Single memory address space gives the users an illusion of a big, centralized main memory, which in reality may be a set of distributed local memories [6]. Several approaches have attempted to achieve single memory address space, or Distributed Shared Memory (DSM), on clusters [12]. However, most of them mainly emphasized on distributed shared *physical* memory. SIOS can extend this idea to Distributed Shared Virtual Memory (DSVM) for all large-scale, out-of-core, cluster applications, by providing an efficient I/O storage to the current DSM packages. It will be shown in Section 5 that the I/O bandwidth delivered by the SIOS is much higher than a single disk, thus providing an efficient storage for swapping memory pages. Figure 4 shows how DSVM can be implemented on top of SIOS.

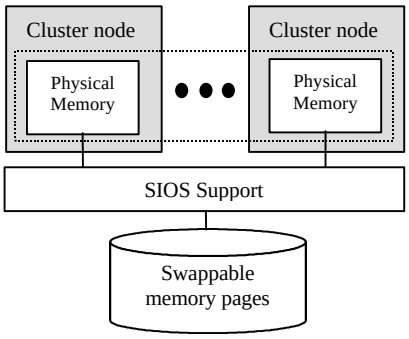


Figure 4: Distributed Shared Virtual Memory

DSVM can utilize the efficient storage provided by the SIOS easily because SIOS provides an illusion of a single disk to each cluster node so that the current DSM packages can be extended to support DSVM by using the established paging mechanisms in the operating systems. Further, the I/O overhead for swapping memory pages can be balanced among all the cluster nodes, while each node's local memory can access the shared virtual memory as if it is on a single-node architecture. It should be emphasized that a DSM package developed for a reliable cluster should have fault-tolerant features so that if one or more nodes fail, the whole memory address space will not collapse. In this connection, the fault-tolerant features of the DSVM can be partially supported by the SIOS.

4.3. Checkpointing Facilities

Checkpointing facilities record each process's state on some permanent storage periodically so that if there are any software or hardware failures, the process can be restarted at the most updated, recorded state, instead from the very beginning. Implementing a checkpointing system requires a checkpointing scheme design and a stable storage, which can tolerates disk access failures. SIOS can provide the stable storage since it tolerates disk failures. As a result, with the support of SIOS, the implementation of a checkpointing system can focus on the checkpointing scheme, rather than on the stable storage.

One example is the distributed checkpointing scheme proposed in [5]. Different levels of checkpointers are stored independently on local and remote disks. In this case, a SIOS based on a mirroring architecture hides the physical distribution of data so that the checkpointing system only needs to store the checkpointers "locally" while the data is actually saved in the remote disks. If there are any failures and the processes need to be restarted, the corresponding checkpointers can be retrieved from the other disks. In this connection, different configurations of the SIOS (e.g. RAID-1, RAID-5, RAID-10, etc.) can support different data distribution patterns required in various checkpointing systems.

4.4. Single Process Space and Process Migration

A single process space means all user processes, no matter on which nodes they reside, share a uniform process identification scheme [6]. All processes should be able to communicate to the others as if they are all run in a single node. SIOS facilitates the implementation of DSVM, which indirectly supports the transparency for inter-process communication through shared-memory.

There are several pieces of information that have to be maintained for a process to migrate to another node

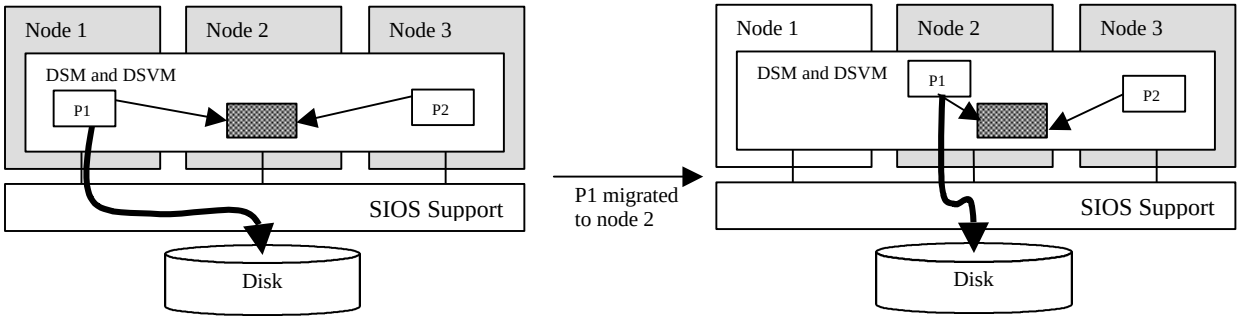


Figure 5: Process Migration Supported by SIOS

successfully: accesses to files, communications with the other processes, and shared memory segments with the other processes. SIOS helps to maintain the accesses to files as the storage can be accessed in any node of the cluster. Furthermore, shared memory segments can also be maintained by the support of DSVM. However, a single network space is needed in order to restore communication channels with the other processes after a process is migrated. Figure 5 shows how process migration can be supported by the SIOS.

As shown in Figure 5, Process 1 (P1) is originally running in Node 1. It has a shared memory region with the other process P2 and it opens several files in the SIOS storage. If P1 is migrated to Node 2, the accesses to the opened files can be restored automatically while the access to the shared memory segment is maintained by the DSM and the DSVM systems.

5. Benchmark Performance Results

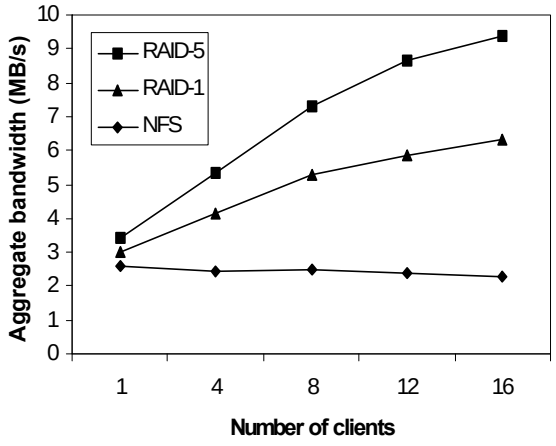
In this section, we present the experimental results obtained in raw disk I/O experiments. The experiments were designed to test the scalability of the CDD architecture. The prototype cluster was built with sixteen 400 MHz Pentium II PCs running the Linux kernel version 2.2.5. These PC nodes are connected by a 100 Mbps Fast Ethernet switch. At present, each node is attached with one 10-GB disk. With 16 nodes, the total capacity of the disk array is 160 GB. All 16 disks form a single I/O space. To test the cooperative operations among the distributed nodes, RAID-5 and RAID-1 architectures were ported with the support of CDDs. The NFS is used as a baseline for comparison purposes.

5.1. Bandwidth Results With Increasing Clients

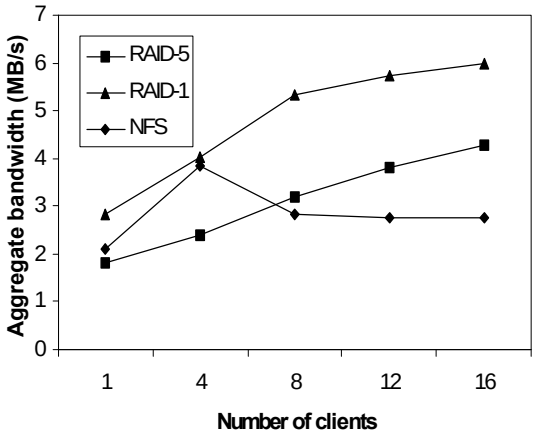
Figure 6 shows the performance of the RAID-5, RAID-1 and the NFS architectures. The results on large read are given in Fig. 6a. In this test, each client reads a 10MB-long file from all the disks. Therefore, the test is truly focused on the parallel I/O capability of the disk array. All the files are set to be uncached and each client only reads its own private file. All read operations are performed simultaneously, with the help of an MPI_Barrier() call. The bandwidth results of small read are not reported as they are very close to that of large read in all numbers of clients.

The NFS throughput is limited at 2.6 MB/s regardless of the number of clients, due to the fact that sequential I/O is performed by the NFS on a central server. As the number of client increases, the NFS becoming the bottleneck shows a declining performance. The two RAID architectures, in contrast, show nearly linear scalability in bandwidth. The RAID-5 architecture scales up to a bandwidth of 9.5 MB/s for 16 clients. RAID-1 lags behind with a show of 6.33 MB/s for 16 clients.

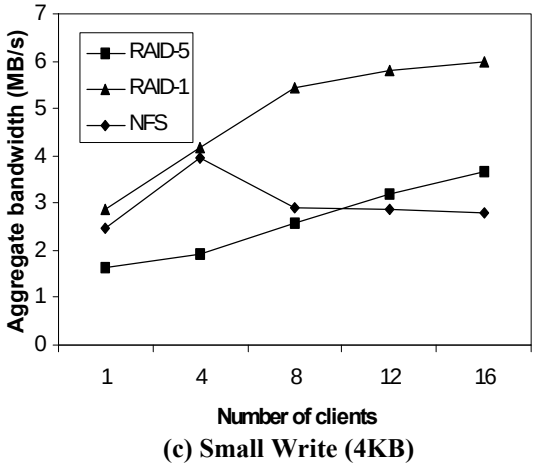
Fig. 6b shows the large write bandwidths of the three I/O subsystems. In this test, each client writes a 10MB-



(a) Large Read (10MB)



(b) Large Write (10MB)



(c) Small Write (4KB)

Figure 6: Aggregate I/O Bandwidth with Increasing Clients

long file to the cache and issues a special sync() call to flush the data blocks to the disks. Fig. 6c shows the small write bandwidths of the three I/O subsystems. In this test, each client writes a 4KB-long file to the I/O subsystem. All write operations among the clients are also synchronized in these experiments. In both experiments, the NFS scales in performance up to four clients, even higher than that of RAID-5, due the caching effect at the NFS server. When the number of client exceed four, the NFS bandwidth drops to a low 2.77 MB/s.

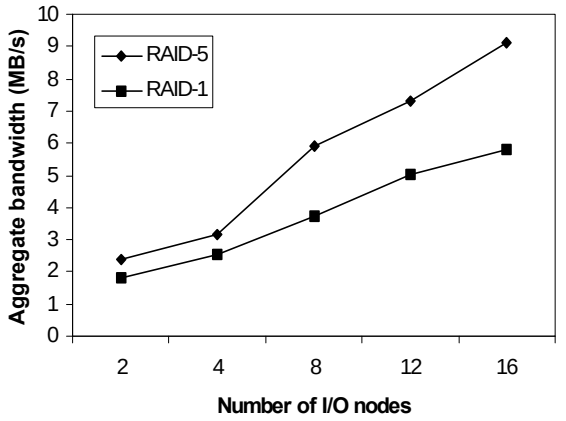
For parallel writes of either a large file (Fig. 6b) or a small block (Fig. 6c), RAID-5 scales slowly due to the heavy overhead involved in parity calculations. For small writes, the RAID-5 bandwidth becomes even worse. RAID-5 shows better performance in large write than small write because full-stripe writes can be performed in large write in which old parities and data blocks need not to be read before the write operations. In contrast, old data and parity blocks must be read in advance in the case of small write. In both cases, a considerable amount of CPU time has been spent in calculating parity information, and this accounts for the poor I/O performance in RAID-5. RAID-1 scales even better than RAID-5, but its bandwidth saturates early to a 5.95 MB/s in both large and small write experiments, due to the fact that only half of the available bandwidth is used for transferring data. However, both RAID architectures show increasing bandwidth with the number of clients, when compared to the declining performance of the NFS.

5.2. Bandwidth Results With Increasing Disk Array Sizes

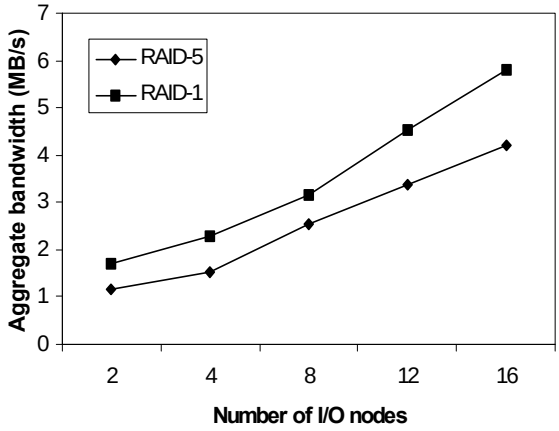
Bandwidth results are plotted in Fig.7 against the disk array size. The results are shown for the two RAID architectures. The number of clients is kept at 16, while all caches are bypassed in the experiments.

Figure 7a shows the performance in long read operations in which each client writes a 10MB-long file to the I/O subsystem synchronously. Again, the results of small read are not shown as they are very close to the long read results. As indicated in the diagram, both RAID architectures show high bandwidth scalability when the disk array size increases. RAID-1 scales up to 5.95 MB/s while RAID-5 scales up to 9.1 MB/s. Figure 7b shows the benchmark results for large writes. The bandwidths of RAID-1 and RAID-5 are 5.72MB/s and 4.29MB/s, respectively. Again, the parity calculation overhead in RAID-5 accounts for its poor performance. Figure 7c shows that the performance of RAID-5 drops further to 3.08MB/s due to the small write overheads. However, in all cases, the two RAID architectures show increasing performance when the disk array size is getting large.

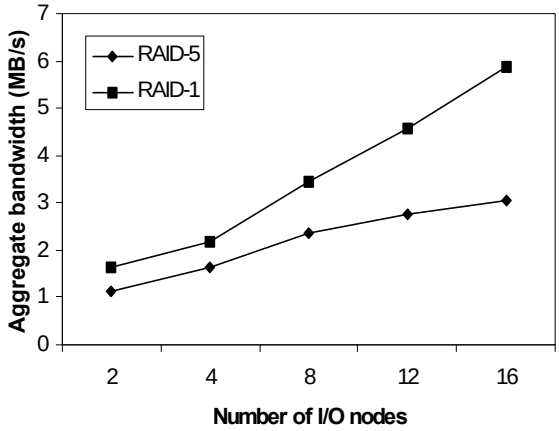
Based on these experiments, the CDD platform is proved to be a good alternative to the NFS for I/O-intensive cluster computing. Both RAID-1 and RAID-5 architectures, with the support of CDDs, show high scalability in bandwidth when compared to the results of NFS.



(a) Large Read (10MB)



(b)Large Write (10MB)



(c) Small Write (4KB)

Figure 7: Aggregate I/O Bandwidth with Increasing Disk Number

6. Related Work and Conclusions

The development of the SIOS was inspired by several research projects. The xFS and the Tertiary Disk projects at Berkeley [1][14], and the Petal/Frangipani project at Compaq Digital [8][16], all have influenced our design philosophy. The main difference between our approach and these projects is that we include data consistency checking in the device driver level. The CDDs work cooperatively to hide the details involved in transferring the data and maintaining the consistency. With the support of CDDs, the design of a distributed file system can be focused on the concurrent file access policies and the related performance considerations. In this case, the implementation cost and the complexity of a distributed file system can be reduced.

Our SIOS architecture cleanly separates the I/O subsystem in a cluster into the file systems and a set of distributed CDDs. All SSI services for I/O operations are provided by the CDDs while the file system modifications are minimized. Its simple architecture suggests that some enabling services for cluster computing can be designed and implemented in a cost-effective manner. Furthermore, it has been shown that some desired SSI services for cluster computing could be built on top of the SIOS. In this aspect, the SIOS is a basic, but powerful, infrastructure to achieving a complete SSI cluster. Benchmark performance results show that our SIOS design offers performance and size storage size scalabilities. We do believe the SIOS design can improve the I/O subsystem of clusters in terms of convenience, scalability, performance and availability.

The first prototype is implemented on the Trojans cluster at University of Southern California and the cluster at the University of Hong Kong. More extensive research work on the SIOS project will be carried out to address the load-balancing issues raised in I/O-intensive cluster environments.

References

[1] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. "Serverless Network File Systems". *ACM Trans. on Computer Systems*, Jan. 1996, pp.41-79.

[2] M. Dahlin, R. Wang, T. Anderson, D. Patterson. "Cooperative Caching: Using Remote Client Memory to Improve File System Performance". *Proceedings of Operating System Design and Implementation*, 1994.

[3] D. G. Feitelson, P. F. Corbett, J. P. Prost, and S. J.Baylor. "Parallel Access to Files in the Vesta File System". *Proceedings of the conference on Supercomputing '93* , 1993, Page 472.

[4] I. Foster, D. Kohr, Jr., R. Krishnaiyer, and J. Mogill. "Remote I/O: Fast Access to Distant Storage". *Proceedings of the Fifth Annual Workshop on I/O in Parallel and Distributed Systems*, November 1997, pp.14-25.

[5] K. Hwang, H. Jin, E. Chow, C.L. Wang, and Z. Xu. "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space". *IEEE Concurrency Magazine*, March 1999, pp.60-69.

[6] K. Hwang and Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill, New York, 1998.

[7] Y. A. Khalidi, J. M. Bernabeu, V. Matena, K. Shirriff and M. Thadani. "Solaris MC: A Multi-Computer OS". *Proceedings of 1996 USENIX Conference*, 1996.

[8] E. K. Lee and C. A. Thekkath. "Petal: Distributed Virtual Disks". *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, October 1996, pp.84-92.

[9] W. B. Ligon III and R. B. Ross. "An Overview of the Parallel Virtual File System". *Proceedings of the 1999 Extreme Linux Workshop*, June, 1999.

[10] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. "Andrew: A Distrubuted Personal Computing Environment". Communications of the ACM, Vol. 29, No. 3, March 1986, pp.184-201.

[11] G. F. Pfister. "The Varieties of Single System Image", *Proceedings of IEEE Workshop on Advances in Parallel and Distributed System*, IEEE CS Press, 1993, pp.59-63.

[12] J. Protic, M. Tomasevic, and V. Milutinovic. Distributed Shared Memory, Concepts and Systems. IEEE Computer Society, Los Alamitos, California, 1998.

[13] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem", *Proc. of the USENIX Conference*, June 1985, pp.119-130.

[14] N. Talagala, S. Asami, D. Patterson, and K. Lutz. "Tertiary Disk: Large Scale Distributed Storage", *UCB Technical Report No. UCB//CSD-98-989*.

[15] R. Thakur, W. Gropp, and E. Lusk. "An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces". *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation*, October 1996, pp.180-187.

[16] C. A. Thekkath, T. Mann, and E. K. Lee. "Frangipani: A Scalable Distributed File System". *Proceedings of ACM Symposium of Operating Systems Principles*, Oct. 1997, pp.224-237.