Single I/O Space for Scalable Cluster Computing

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

In this paper, we propose a novel Single I/O Space architecture for achieving a *Single System Image* (SSI) at the I/O subsystem level. This is very much desired in a scalable cluster computing environment using commodity components. Our design achieves a single address space for all blocks of data in the cluster, which can tolerate all single disk failures. While traditional approaches focused on at user-level or at distributed file subsystem level, we separate the I/O subsystem of a cluster into the file system and a set of distributed *Virtual Device Drivers* (VDDs). All SSI services are provided by the VDDs with unmodified file systems. Compared to previous approaches, our approach has higher transparency, better performance, lower implementation cost, higher availability, and application compatibility for I/O intensive cluster computing.

1. Introduction

Clusters have proven their potentials in the area of low-cost, but high-performance, parallel computing [1, 2, 5]. 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 [1, 5]. Specifically, cluster computing demands a single I/O space in distributed, I/O intensive, operations. On the other hand, while the speeds of microprocessor have been increasing dramatically in the past few decades, the speeds of I/O devices have been increased at a much slower rate [3, 4]. The I/O operations have already become a major bottleneck for those traditional out-of-core computation problems as well as some new application domains, such as multimedia database, digital libraries, etc. Due to these reasons, 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 objectives are summarized below:

1. A single address space for all data blocks in the cluster. This means the users can utilize all disk storage in a cluster without knowing the physical location of each data block or file.
2. High scalability, availability, and compatibility with current cluster architectures and applications.
3. Remote disk I/O operations should have performance at least comparable to, or even better than, that of local disk I/O operations.

In this paper, we present a novel Single I/O Space architecture. The paper is organized as follows: Section 2 introduces some basic concepts on SSI and SIOS. Section 3 describes some of the previous approaches and their pros and cons. Section 4, 5 and 6 present our detailed design and design issues of the SIOS. Section 7 presents a preliminary performance analysis on the SIOS. Finally, a conclusion is made in Section 8 with more future work.

* This research was supported by Hong Kong RGC grant 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 University of Southern California Engineering School

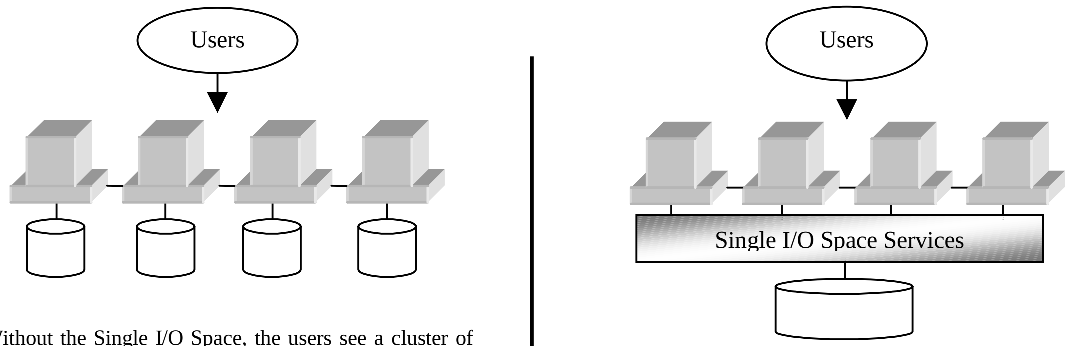
2. Single system image and SIOS

SSI services enable a cluster of PCs or workstations to be used as a single computing unit in a scalable and efficient manner. In other words, SSI services target at getting most out of the theoretical computing power of the clusters 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.

Due to the I/O bottleneck problem and the challenges of cluster computing, 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 motivated and its aim is to build a SSI for disk I/O operations in scalable cluster environments. Figure 1 illustrates the SSI service provided by the Single I/O Space from the viewpoint of cluster users.

3. Previous approaches towards SIOS

Previous research showed that SSI for disk I/O operations could be implemented in the user level [6, 7,



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

13], the file system level [8, 10, 12] and the device driver level [11]. Advantages and disadvantages of each approach are given in subsequent sections.

3.1. User level approach

Implementing the SSI services in the user level has a number of advantages. First, compared to the lower levels, user level approach has relatively lower implementation cost. Second, portability of the implemented systems will be much higher than the systems in the file system and the device driver levels. Finally, it is much easier to trace design and implementation errors if the system is in the user level. The *Parallel Virtual File System* (PVFS) at the Clemson University [6] and the *Remote I/O* (RIO) project at the Argonne National Laboratory [7] are two typical examples of designing SIOS in the user-level.

However, the user level approach does introduce two major problems. The first problem is that they cannot provide a complete SSI while users still have to use specific APIs and identifiers in order to exploit full functionality of the packages. The second problem is the performance restriction. A pure user-level design implies usage of kernel facilities including network and file I/O services, which are not intentionally designed for scalable parallel applications. Furthermore, using system calls to perform network and file I/O are expensive. What we expect in our SIOS design is a complete SSI to the users and scalable performance for cluster computing.

3.2. File system level approach

File system level implementations can provide a complete SSI to the users. Users can access remote data as if it is accessed locally. Besides, performance optimization is possible since the file system itself is specially designed for remote disk accesses. Further, the file system is able to control the distribution patterns of files for performance reasons. The *Serverless Network File Systems* (xFS) project at the University of California at Berkeley [8] and the *Solaris MC* project at the Sun Microsystems Laboratories [10] are typical examples of designing SIOS at the file system level.

On the other hand, file system level designs do have their own shortcomings. First, designing, implementing and deploying a new distributed file system require high cost. Consequently, widespread acceptance of the proposed ideas is not easy. Second, changing the file system does not guarantee strict compatibility with current applications. This directly discourages the deployment of the proposed architecture for production clusters. What we expect is to design a SIOS with unmodified file system and to achieve a low cost/performance ratio both in the implementation and the deployment phases. Furthermore, the design itself should be easily ported to existing and future clusters. Finally, we want a SIOS to be highly compatible with current applications.

3.3. Device driver level approach

Device driver level designs provide SSI not only to the users, but also to the file systems. This suggests that SSI services for disk I/O operations can still be supported without file system modifications. Besides, compared to the file system approach, limiting the design in the device driver level obviously reduces design and implementation costs. Furthermore, higher compatibility with current applications can be achieved if the file system is kept unchanged. In short, the device driver level approach solves most problems found in the above two approaches.

The major drawback of this approach is that it is difficult to control the distribution pattern of files for performance optimization purpose. An example is the *Petal* project at the Digital Equipment Corporation [11]. It enables a local file system in each cluster node to view the physically distributed disks as a collection of virtual disks. Each virtual disk can be accessed as if it is a local disk. In this way, the actual data transfer is handled in the device driver level while keeping the local file systems unmodified. The system is believed to be easier to model, design, implement, and tune. In our point of view, it is possible to improve Petal's design. Petal does provide a global name space for logical disks in the cluster. We want to extend the global name space to each data block in the cluster. In other words, Petal still requires users to use *separated disks* in the cluster, but we want to give an illusion of *a single disk* to the users.

4. Designing SIOS at device driver level

In this section, we present our design of the Single I/O Space. First, we justify our design based on the chosen implementation level. Second, we introduce the primary building block of the SIOS, the *Virtual Device Driver* (VDD).

4.1. Design objectives

We choose 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 *Virtual Device Driver* (VDD) as the disk device driver for each node in the cluster. These VDDs 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 VDD. We will discuss this in detail in Section 4.2 and section 5.

High availability features supported - High availability features can be supported by implementing some established technologies directly in the VDDs. We have implemented RAID-5 technology in our prototype, other technologies such as RAID-10 can be easily employed by modifying a specific program module in our SIOS design. The SIOS with this kind of configuration can withstand a single disk failure while keeping all the data online.

Performance and Storage size scalability - There is no central server in our design and the VDDs only maintain a *peer-to-peer* relationship with the others. So, scalability can be guaranteed by the server-less design [8]. 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.

As we will mention in Section 5.2, the size of the buffer cache in our VDD 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* [9] 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 we have limited all implementation in a single level while keeping all other parts of the systems unchanged. In particular, file system modifications can be avoided. This guarantees high compatibility with current applications.

4.2. Distributed virtual device drivers (VDDs)

The SIOS consists of *Virtual Device Drivers* (VDDs) distributed throughout the cluster nodes, each node runs a copy of the VDD as its disk device driver. A VDD is constructed in the device driver and the buffer cache levels. The block device buffer cache is included in the VDD for maintaining data consistency. The VDDs in the 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 illustrates the whole picture.

5. Addressing scheme and related issues

In this section, we first describe the addressing scheme in the SIOS design. Second, we discuss some issues, which we need to consider in designing and implementing the SIOS.

5.1. Addressing scheme

The SIOS provides a single address space for each data block in the cluster. As a consequence, each block of data on a disk has two addresses, the logical address and the physical address. The logical address is the address of a data block in the single address space while the physical address states the physical location of the data block. Besides, each partition participating in the SIOS is given a unique, global, ID for identification purpose. The physical address of each block includes the following information:

- the physical address (i.e. the offset) of that block in the storing partition
- the local partition ID of that storing partition
- the device name of the corresponding disk
- the machine name

The logical addresses are used by the upper layers while the physical addresses are hidden and are only used

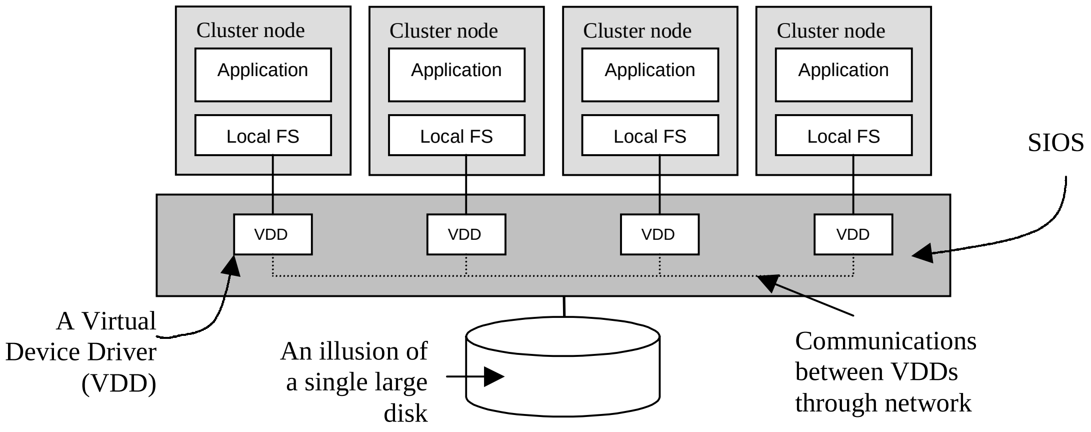

Cluster node
Application
Local FS
VDD
SIOS
A Virtual Device Driver (VDD)
An illusion of a single large disk
Communications between VDDs through network

Figure 2: Distributed virtual device drivers (VDDs)

to reference data internally among the VDDs. In our prototype, data blocks are distributed according to the RAID-5 structure. Figure 3 illustrates the single address space in the SIOS design.

5.2. Related issues in SIOS design

This section addresses some related issues when designing the SIOS for cluster computing.

Potential Bottleneck in the Buffer Cache - SIOS provides linear storage size scalability that the users can utilize extra disk spaces transparently when more nodes and disks are added into the clusters. However, an unmodified buffer cache, which is originally designed for managing limited disk space, may not be efficient enough to handle a huge, virtual, storage device provided by the SIOS. To tackle this problem, the system is subject to a certain degree of configuration. In our prototype, we increase the size of the block device buffer cache so as to obtain an observable gain in performance in using buffer cache, and to benefit from the cooperative-caching effect [10]. A balance between *better SIOS performances* and *lower memory requirements* should be obtained by analysis on the performances after the implementation.

Data Consistency - In our design, we have a mechanism to maintain data consistency within the SIOS. It ensures all data blocks would be flushed to the physical storage before the processes running in the other nodes can modify them. However, the unmodified file system does introduce some data consistency problems in the whole I/O subsystem as it has its own caches (namely the *Directory Cache* and the *Inode Cache*). Expired data stored in these caches may result in data inconsistency problems. In order to minimize the chances of data inconsistency, the lifetime of the entries stored in these caches may be lowered. This configuration may slightly affect the performances of the file system, but not significantly.

Disk Partitioning - A disk participating in the SIOS that contains the operating system must be at least divided into two partitions. The reason of this is that the disk must be accessed once the system boots, if the operating system is stored in the remote SIOS partitions, the operating system region may be inaccessible until the network services start. So, a disk participating in the SIOS has two partitions, one is the local non-SIOS partition which stores the operating system, and the other is the SIOS partition.

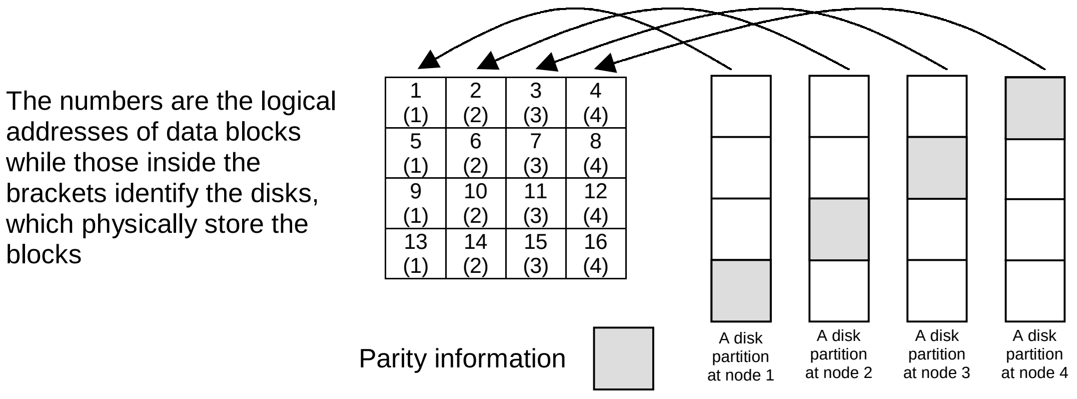

The numbers are the logical addresses of data blocks while those inside the brackets identify the disks, which physically store the blocks
1 (1)
2 (2)
3 (3)
4 (4)
5 (1)
6 (2)
7 (3)
8 (4)
9 (1)
10 (2)
11 (3)
12 (4)
13 (1)
14 (2)
15 (3)
16 (4)
Parity information
A disk partition at node 1
A disk partition at node 2
A disk partition at node 3
A disk partition at node 4

Figure 3: Addressing scheme in the SIOS

Portability - Our SIOS design supports high user application compatibility by providing an unmodified file system. However, changing the disk device driver requires modification of the system kernel. This affects the portability of the SIOS in newer versions of the kernel. To tackle this problem, we tried to implement our prototype as a loadable kernel module so that we only need to change some referenced symbols when porting the SIOS to newer versions of the kernel. Besides, a loadable kernel module also eases installation of the SIOS in other clusters.

6. The VDD architecture

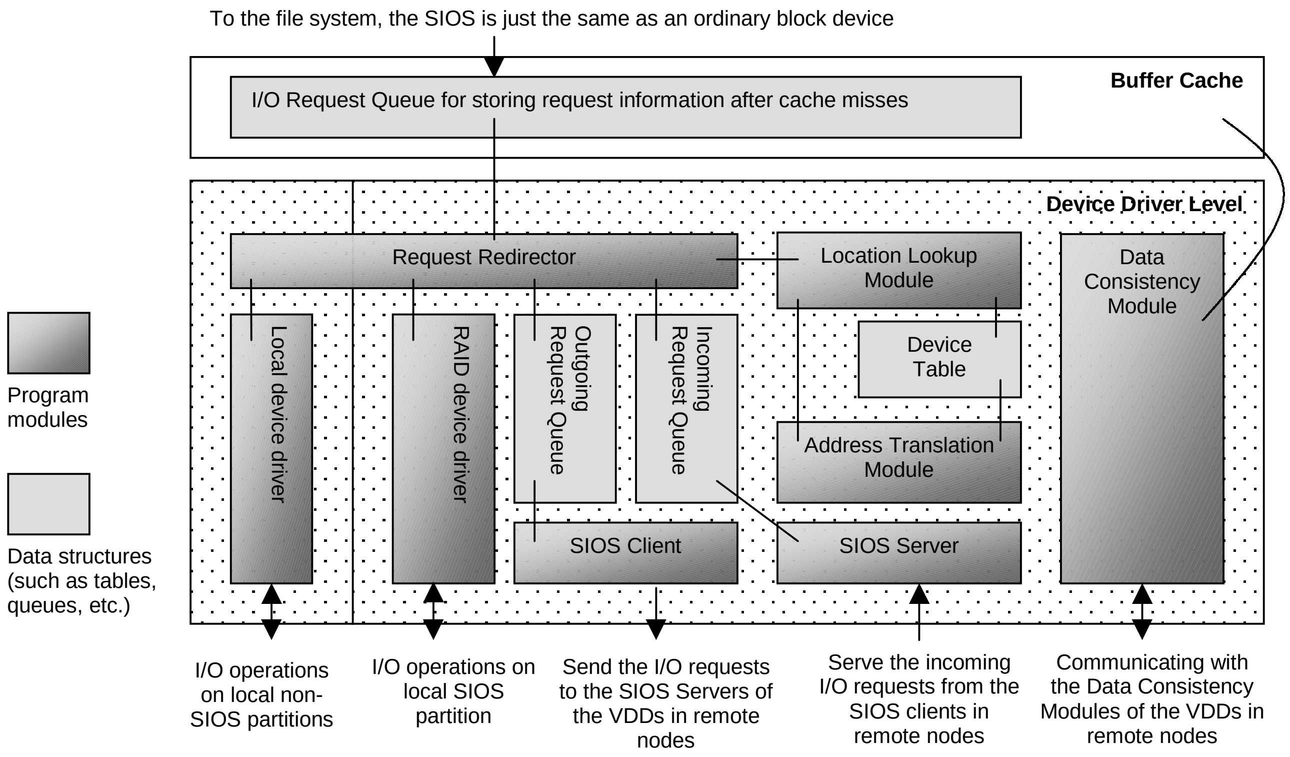
In this section, we describe the internal architecture of the VDD and the functionalities of its components. The VDD is constructed in the buffer cache and the device driver levels. The part lies in the buffer cache level is mainly responsible for data consistency while the part lies in the device driver level is responsible for processing local and remote I/O requests. Figure 4 shows the internal architecture of a VDD. The following paragraphs explain the functions of each component in the VDD program.

Request Redirector - In case of cache misses in the buffer cache, the buffer cache will make a request and store it in the I/O Request Queue. The Request Redirector determines the physical location of the requested data segment based on the information provided by the Location Lookup Module. Then, it sends the requests queued in the I/O Request Queue to the appropriate program modules. There are three possible cases:

- If the operation should be done on the local non-SIOS partitions, it will directly send the requests to the Local Device Driver
- If the data is stored in the local SIOS partition, it will ask the RAID Device Driver to serve the request
- For data stored in a remote SIOS partition, it will en-queue the request to the Outgoing Request Queue

Besides, the Request Redirector is also responsible for checking if there are any incoming requests in the Incoming Request Queue. If there are some, it will redirect the requests to the RAID Device Driver in the VDD.

Location Lookup Module - It is responsible for providing the physical location of each data block to the Request Redirector. It sends the logical address of each block to the Address Translation Module and gets the global partition ID. Then it uses the ID as a key to find the corresponding machine name, device name and local partition number from the Device Table. It returns all these information to the Request Redirector after the translation.



To the file system, the SIOS is just the same as an ordinary block device
Buffer Cache
I/O Request Queue for storing request information after cache misses
Device Driver Level
Request Redirector
Location Lookup Module
Data Consistency Module
Device Table
Address Translation Module
Local device driver
RAID device driver
Outgoing Request Queue
Incoming Request Queue
SIOS Client
SIOS Server
Program modules
Data structures (such as tables, queues, etc.)
I/O operations on local non-SIOS partitions
I/O operations on local SIOS partition
Send the I/O requests to the SIOS Servers of the VDDs in remote nodes
Serve the incoming I/O requests from the SIOS clients in remote nodes
Communicating with the Data Consistency Modules of the VDDs in remote nodes

Figure 4: Structure of a virtual device driver (VDD)

Address Translation Module - This module accepts a block's logical address as the parameter and returns the global partition ID of the partition, which stores that data block, to the calling module.

Local Device Driver - It is responsible for performing the I/O on the local, non-SIOS partition. It should be identical to an ordinary disk device driver except that it communicates with the Request Redirector, instead of the buffer cache.

RAID Device Driver - It is responsible for performing all I/O operations on the data that is stored on the local partition(s) participating in the SIOS.

SIOS Client - It is executed periodically to send the outgoing I/O requests to the remote nodes. Furthermore, it checks to see whether there are any requested data arrived at local buffers. If an outgoing I/O request has been served, the SIOS Client will move the requested data segment to the buffer cache.

SIOS Server - It is also executed periodically to listen to the incoming port to see whether there are any incoming I/O requests, if there are some, it will en-queue them to the Incoming Request Queue.

Data Consistency Module - It is responsible for maintaining the data consistency within the SIOS. It does this by communicating with other Data Consistency Modules and manage the buffer cache entries based on a simple data consistency model similar to that used in the *Frangipani* distributed file system [17].

7. Analysis of performance effects

This section presents some basic analytical results on the performance of the SIOS. Figure 4 shows a simplified abstract model for our analysis. As shown in the diagram, the cluster consists of a group of distributed nodes, each of them have its own buffer cache. The whole cluster is '*connected*' to the distributed RAID system emulated by SIOS services.

Table 1 defines the symbols used in our analysis. Several assumptions are made below.

1. All the I/O requests are evenly distributed among all the disks in the SIOS, and all distributed disks serve the same I/O request simultaneously.
2. The network latencies for sending I/O requests, consistency information, and acknowledgements are negligible. Only the network latency for transmitting data is taken into account.
3. The analysis focuses on typical transaction-processing workload, where the average size of data involved in a single I/O request is assumed to be 4 Kbytes [15].
4. For many transaction-processing workloads, a majority of requests are queries to read data, the ratio of reads to writes is typically 2 or 3 to 1 [15]. In this paper, the probabilities for read and write requests are assumed to be 0.7 and 0.3 respectively.
5. Write operation to the member disks of SIOS is full stripe write, therefore we can ignore the effect of small write problem.
6. Time used in parity calculations and memory accesses can be negligible.

In order to obtain numerical results, we set the following parameter values: the strip width and the stripe unit size of the RAID system are 5 and 1 Kbytes respectively. The block size in the buffer cache is 1 Kbytes. The size, number of cylinders, and the sector size of a single disk are 10.2GB, 16383, 512 bytes respectively (Seagate Medalist 10232 ST310232A). The hit ratio, *h*, for a single block of data in the buffer cache is 0.7. The network latency for transmitting a single block of data (1 Kbytes long), L_B, equals 0.123 us (under 65% network bandwidth utilization for kernel-level socket, through a fast ethernet network). The number of blocks of data requested in an I/O request, n, is set to 4 based on assumption (3). For analytical purposes, we assume there are 20 PCs in the cluster, each has a local disk attached.

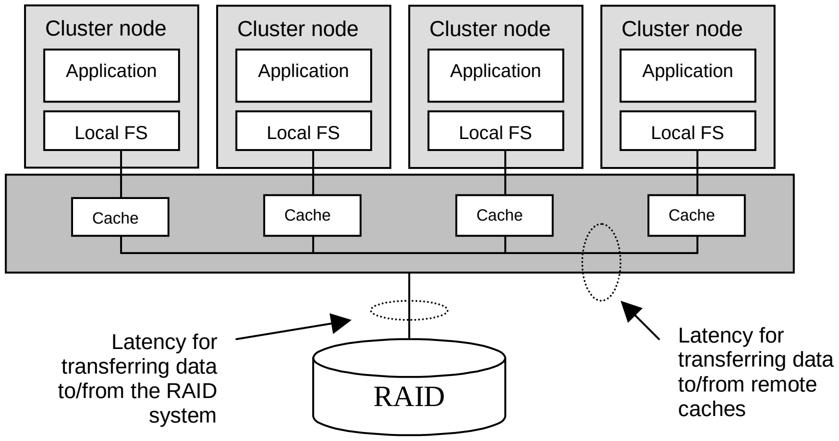


Figure 4: Simplified model of SIOS

Table 1: Definition of notations

Symbol	Definition
r_{raid}, w_{raid}	mean response time for serving a read/write request to the RAID system
R_{raw}, W_{raw}	mean response time for serving a read/write request to SIOS without cache
R_c, W_c	mean response time for serving a read/write request to cached SIOS
h	hit ratio for requesting a single block from the buffer cache
L_B	network latency for transmitting a single block of data (1 Kbytes long)
N	number of nodes in the cluster
n	number of blocks of data requested in an I/O request
α_r, α_w	latency for reading/writing *i* blocks of data from/to remote nodes
β_r, β_w	latency for reading/writing *i* blocks of data with at least one stored locally

7.1. Raw RAID response time analysis

Before we analysis the overall performance of the SIOS system, we first analyze the response time of serving read/write requests to the *raw* RAID-5 system while the caching effects are not taken into account. We combine the models proposed in [14, 15] to perform the analysis. The response time for serving a read/write request (*n* blocks of data) sent directly to the RAID system, R_{raw} and W_{raw}, are given by:

$$R_{raw} = L_B + r_{raid}, \qquad W_{raw} = 2L_B + w_{raid} \tag{1}$$

Note that the network latency is *not* nL_B, since the data blocks are transferred in parallel. For the write operations, the original data blocks and the parity blocks must be read before the new data blocks can be written, consequently the network latency involved is $2L_B$. The detailed equations for computing r_{raid} and w_{raid} can be found in [14, 15]. Figure 5 shows the mean response time for serving read/write requests under different arrival rates of I/O requests to the SIOS. From the results shown in the figure, we find that write operation in SIOS is easier to saturate than read operation. For read operation, even in the heavy workload, the system still scales quite well.

7.2. Caching effect analysis

According to Figure 4, each cluster node has its own buffer cache. In this section, we analyze the impacts of the caching effect to the overall SIOS performance. It can be shown that the mean latency for reading/writing i blocks of data from/to remote nodes, i.e. α_r and α_w, are $[L_B + (1-h^i)r_{raid}]$ and $[2L_B + (1-h^i)w_{raid}]$ respectively. When one of these i blocks is stored locally, the mean response time for read/write operations, β_r and β_w, are $\max\{r_{raid}, [L_B + (1-h^{i-1})r_{raid}]\}$ and $\max\{w_{raid}, [2L_B + (1-h^{i-1})w_{raid}]\}$ respectively. Hence, the mean response times for reading/writing *n* blocks of data under the caching effect are given by:

$$\begin{aligned} R_c &= \sum_{i=1}^{n} \binom{n}{i} (h)^{n-i} (1-h)^i \left\{ \left[\frac{N-i}{N}\alpha_r + \frac{i}{N}\beta_r \right] \right\} \\ W_c &= \sum_{i=1}^{n} \binom{n}{i} (h)^{n-i} (1-h)^i \left\{ \left[\frac{N-i}{N}\alpha_w + \frac{i}{N}\beta_w \right] \right\} \end{aligned} \tag{2}$$

Figure 6 shows the mean response time for serving read/write requests under different I/O request rate. Compared to Figure 5, it is clear that the caching and cooperative-caching effects have significantly improved the performance of the SIOS system.

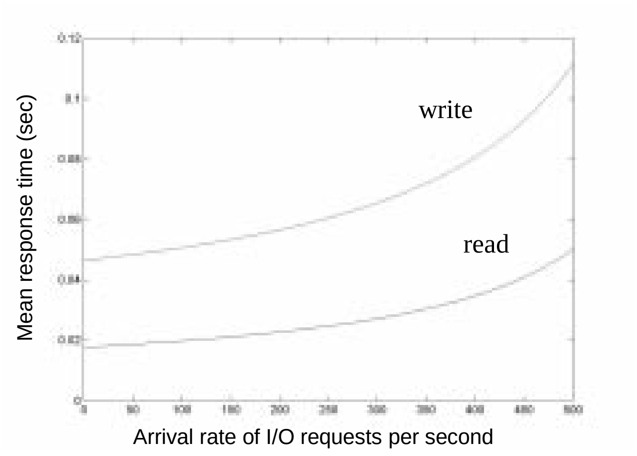


Figure 5: Mean response time of serving I/O requests under different request arrival rates

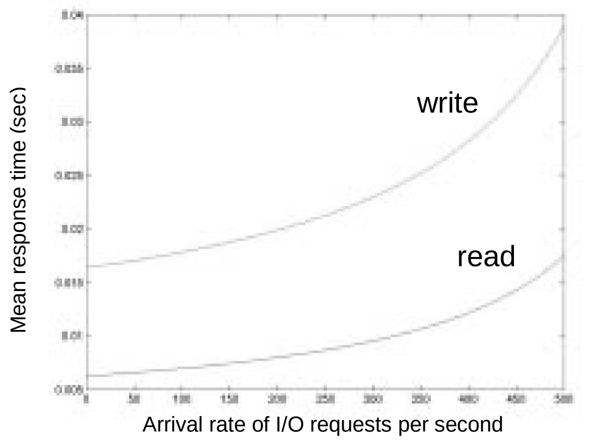


Figure 6: Mean response time of serving I/O requests (with caches) under different request arrival rates

7.3. Bandwidth analysis

In this part, we analyze the aggregate read bandwidth of the SIOS system. We use the equations derived in [16] to perform the analysis. For analytical purpose, we fix the arrival rate of I/O request at 270. This is because the distributed RAID of a small cluster (e.g. 8 nodes) will become saturated when the arrival rate is too high. Actually, a larger cluster size can obtain higher bandwidth under a higher arrival rate of I/O request. In this way, bandwidth scalability is guaranteed when more nodes are added into the cluster. Figure 7 shows the bandwidth of SIOS at different cluster size.

Figure 8 shows the aggregate bandwidth of SIOS under different I/O request rates. From the figure we can see that the bandwidth for read operation will not be saturated even under heavy workload (large I/O arrival rate for each disk). For write operation, due to RAID-5 architecture used in SIOS, each write operation involves a read-modify-write cycle. Therfore, the bandwidth for write operation can easily be saturated even in a moderate workload (400 I/O requests totally in our system). More research work should be carried out to improve the write performance of SIOS system.

8. Conclusions and further work

Our Single I/O Space design avoids most problems found in the previous approaches. Its novel architecture cleanly separates the I/O subsystem in a cluster into the file systems and a set of distributed *Virtual Device Drivers* (VDDs). All SSI services are provided by the VDDs while the file systems are kept unmodified. Its simple architecture suggests that some enabling services for cluster computing can be designed and implemented in a cost-effective manner.

Preliminary analysis shows that our SIOS design offers performance and size scalability. Especially for the read operations, SIOS scales quite well even under heavy workload. Due to the adoption of RAID-5 to leverage the reliability of SIOS, the write performance is found to be saturated when the system's workload increases. On the other hand, we can see that the cooperative caching effects in the SIOS system can greatly improve the system performance. We believe the SIOS design can improve the I/O subsystem of clusters in terms of convenience, scalability, performance and availability.

The implementation of the SIOS is still under progress. The first prototype is implemented on the Trojans cluster at University of Southern California and the cluster at the University of Hong Kong. By the time of writing this paper, the prototype design of SIOS is finished. Experiments will be carried out to test the performances of SIOS for long and short read/write operations and file manipulation operations under different cluster sizes. More extensive research work on the SIOS project will be: to analyze the access patterns of some specific applications (for example, multimedia database systems) and propose methods to optimize I/O performances in the device driver level; and to develop some SSI cluster services based on the SIOS system.

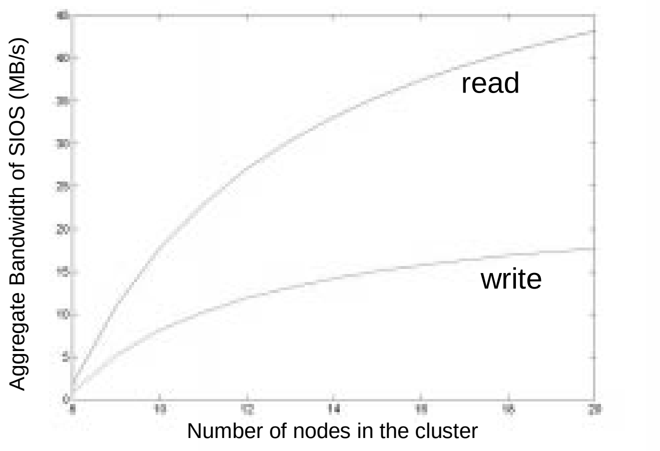


Figure 7: Aggregate bandwidth at different cluster size (I/O request arrival rate is 270)

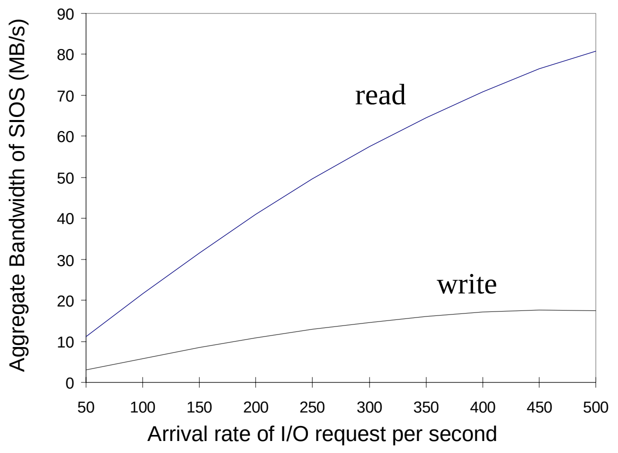


Figure 8: Aggregate bandwidth under different I/O request rates

References

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

[2] G. F. Pfister, *In Search of Clusters*. Prentice Hall, Upper Saddle River, 1995.

[3] R. H. Patterson and G. A. Gibson. "Exposing I/O Concurrency with Informed Prefetching". *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems*, 1994, pp.7-16.

[4] N. Nieuwejaar and D. Kotz. "Performance of the Gallery Parallel File System". *Proceedings of the Fourth Workshop on Input/Output in Parallel and Distributed Systems*, Philadelphia, May 1996, pp. 83-94.

[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*, January-March 1999, pp.60-69.

[6] 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.

[7] 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.

[8] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. "Serverless Network File Systems". *ACM Transactions on Computer Systems*, Vol.14, No.1, 1996, pp.41-79.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] A. Thomasian and J. Menon. "Performance Analysis of RAID5 Disk Arrays with a Vacationing Server Model for Rebuild Mode Operation". *Proceedings of the 10th International Conference on Data Engineering*, 1994, pp.111-119.

[15] A. Kuratti and W. H. Sanders. "Performance Analysis of The RAID 5 Disk Arrays". *Proceedings of International Symposiu on Computer Performance and Dependability*, 1995, pp.236 -245.

[16] E. K. Lee and R. H. Katz. "An Analytic Performance Model of Disk Arrays". *Proceedings of 1993 ACM SIGMETRICS*, 1993, pp.98-109.

[17] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. "Frangipani: A Scalable Distributed File System". *Proceedings of ACM Symposium of Operating Systems Principles*, 1997.