Fault-Tolerant Clusters of Workstations with Single System Image

Kai Hwang
University of Southern California
Edward Chow and **Cho-Li Wang**
University of Hong Kong
Hai Jin
Huazhong University of Science and Technology
Zhiwei Xu
Chinese Academy of Sciences

Adopting a new distributed checkpointing RAID architecture, the authors develop a single I/O address space for building highly available clusters of workstations. They propose a systematic approach to achieving single system image by integrating existing middleware support with the newly developed features through cluster hardware and software experimentation.

The computing trend is moving from clustering high-end mainframes to clustering desktop computers. This trend is triggered by the widespread use of PCs, workstations, Gigabit networks, and middleware support for clustering. This paper presents new approaches to achieving *fault tolerance* and *single system image* (SSI)[1] in a workstation cluster. A *multicomputer cluster* is a collection of node computers, which are physically connected by local area networks or high-bandwidth switch networks using optical fibres. The workstations in the cluster can work collectively as an integrated computing resource, that is a SSI, or they can operate as individual computers, separately.

As shown in Fig.1, present clusters are mostly small in size and provide only limited SSI services. Future clusters are bound to move toward the upper right corner of

the design space. The common goal is to build clusters with higher size-scalability and better support for SSI and availability. The implication is that future clusters have the potential to replace the MPP, SMP, or CC-NUMA multiprocessors as compared in the sidebar under the title "Cluster as A Computer Architecture".

We focus our study on clusters with high availability through checkpointing, distributed RAID with parity checks, and SSI support. In particular, we developed a *single I/O address space* among all disks and peripheral devices attached in the cluster. This will enable remote disk accesses directly, a necessary step to implement reliable cluster of workstations or to build PC clusters with robustness.

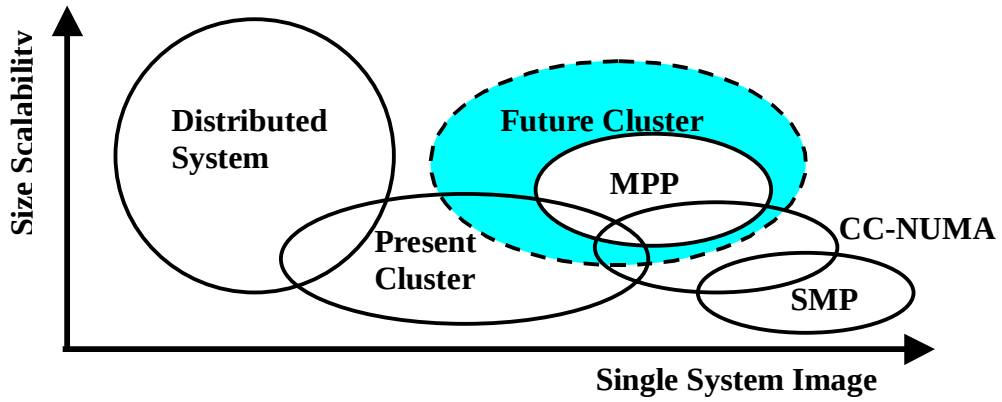


Figure 1. Design space of competing computer architectures

SSI and Availability Design Goals

In a cluster, it is desired to have SSI and availability supported by middleware between the node OS and user application environment (Fig.2). The middleware consists of essentially two layers of software, which glues all node OSs together to establish the SSI and enhanced availability. The availability infrastructure enables the cluster services of checkpointing, automatic failover, recovery from failure, and fault-tolerant operation among all cluster nodes.

The SSI layer supports collective cluster applications, which demand operational transparency and scalable performance. The clusters offer SSI at a wide range of abstraction levels. At one extreme, a cluster can function as a tightly coupled NUMA or MPP system. At the other extreme, it can behave like distributed computer systems with multiple system images. The research goals are commonly focused on *complete transparency* in resources management, *scalable performance* and *system availability* in supporting user applications[2].

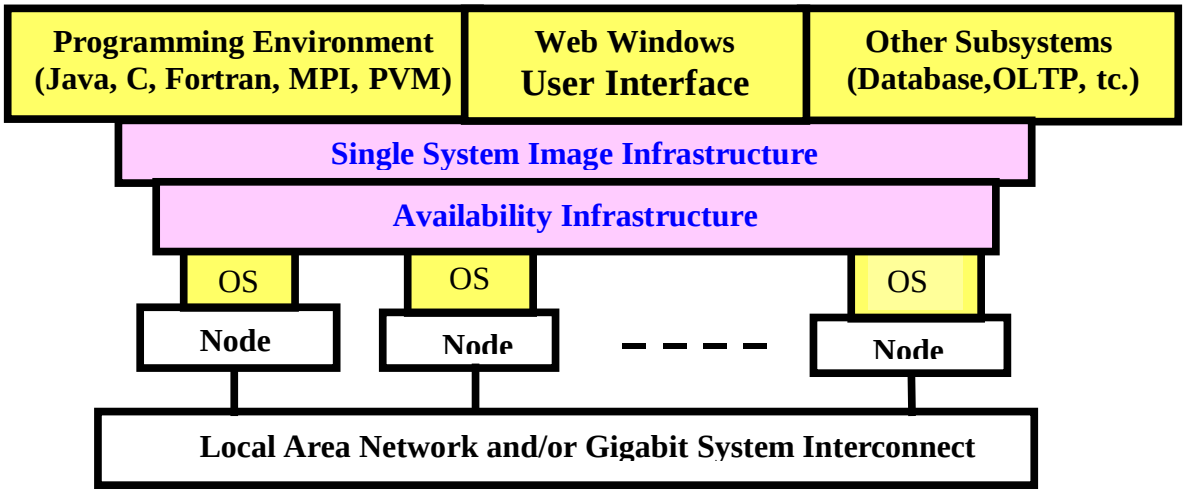


Figure 2. A workstation cluster with hardware, software, and middleware support for single system image and enhanced availability.

_____(Beginning of Sidebar A)_____

Cluster as A Computer Architecture

The term *clusters* have been called by different research groups as *clusters of workstations* (COW)[1], or *networks of workstations* (NOW)[2], or *multicomputer clusters*[3]. To compare clusters with other competing computer systems, we summarize in Table 1 four major classes of computer architectures that compete with each other for a fair market share.

The term MPP stands for *massively parallel processors*. SMP refers to *symmetric multiprocessors* with shared memory. The term CC-NUMA is associated with a scalable multiprocessor having a *cache-coherent non-uniform memory access* architecture. Distributed systems are the conventional *network of independent computers*.

Since each node runs with its own operating system, a traditional network of computers have multiple system images on different nodes. On the other hand, an SMP server must have a *single system image* (SSI) with a centralized shared memory. In a cluster, it is desired to have a SSI across all computer nodes.

As shown in Fig.1, the distributed systems and SMP are two extreme architectures with respect to the concept of SSI. The clusters, MPP, and CC-NUMA are computer architectures between the two extremes. Table 1 reveals the architectural and functional characteristics of the competing computer architectures. Interesting readers are referred to references[1, 3] for details.

Table 1 Key Characteristics of Scalable Parallel Computers[3]

Characteristic	MPP	SMP , CC-NUMA	Cluster	Distributed System
Number of Nodes	O(100) - O(1000)	O(10) - O(100)	**O(100) or less**	O(10) - O(1000)
Node Complexity	Fine or medium grain	Medium or coarse grain	**Medium grain**	Wide range
Internode Communication	Message passing or shared variables for DSM	Centralized and distributed shared memory	**Message Passing**	Shared files, RPC, message passing, IPC protocol
Job Scheduling	Single run queue at host	Single run queue mostly	**Multiple queues but coordinated**	Independent multiple Queues
SSI Support	Partially	Always in SMP and some NUMA	**Desired**	No
Node OS Copies and Type	N microkernels and 1 monolithic OS at host	One monolithic for SMP and multiple for NUMA	**N OS platforms (Homogeneous or microkernel)**	N OS platforms (Heterogeneous)
Address Space	Multiple (Single for DSM)	Single	**Multiple or single**	Multiple
Internode Security	Unnecessary	Unnecessary	**Required if exposed**	Required
Ownership	One organization	One organization	**One or More organizations**	Many organizations
Network Protocol	Nonstandard	Nonstandard	**Standard**	Standard
System Availability	Low to medium	Low for SMP and higher for NUMA	**Highly available or fault-tolerant**	Medium
Performance Metric	Throughput and Turnaround Time	Turnaround Time	**Throughput and turnaround time**	Response Time

References

1. G. F. Pfister, *In Search of Clusters: The Ongoing Battle in Lowly Parallel Computing,* , Prentice Hall PTR, Upper Saddle River, New Jersey, 07458, 1998 (Second Edition).
2. T.E. Anderson, D.E. Culler, D.A. Patterson, et al, "A Case for NOW (Networks of Workstations)", *IEEE Micro,* Feb. 1995, pp. 54-64.
3. K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, WCB / McGraw-Hill, New York, (ISBN 0-07-031798-4), 1998.

(End of Sidebar A)

COMPLETE TRANSPARENCY

The SSI layer should let the user see a single cluster system, instead of a collection of independent nodes. For example, in a SSI cluster with a *single entry point*, users can login at any node. The software installation needs to be loaded only at one node of the cluster. In a loosely

coupled network of computers, one needs to install the same software for each node. Towards complete transparency in resource allocation, de-allocation, and replication, the implementation details should be kept invisible to user processes.

To have a single entry point, when a user *telnet* and *ftp* to the cluster, the two sessions should see the same home directory. The user wants to use a *single file hierarchy*, but how the file system is physically organized in the backup storage is transparent. The file system could be mounted from a central file server; or fully distributed among many nodes. A file could be duplicated in several nodes. When a node fails, the portion of the file system could be migrated to another node.

SCALABLE PERFORMANCE

The scalability is related to the performance of a cluster. An efficient cluster should not be limited in physical size and loading pattern. When a node is added to a cluster, the protocol and API (*application programming interface*) do not have to change. The cluster performance should scale with more nodes allocated. The SSI services must include load balancing and parallel support. The workload among the nodes should be evenly distributed. For example, single entry point should distribute a login request to the most lightly loaded node. The cluster efficiency demands that the SSI services have small overheads. The time to execute the same operation on a cluster should not be much longer than that on a single workstation.

ENHANCED AVAILABILITY

In a cluster, the SSI services should be highly available all the time. Any single point of failure should be recoverable without affecting user's applications. High availability should employ checkpointing and fault tolerant technologies to enable rollback *recovery*. Fault-tolerant cluster demands the features of *hot standby*, *failover*, and *failback* services[1, 2].

While replicating resources, the consistency problem arises. For example, file system coherency and shared memory consistency. If multiple operations are performed simultaneously, the cluster system should be able to keep the duplicated data objects consistent. Critical section resources must be properly locked or protected to avoid data races. Other availability issues include security and data encryption to achieve protected access of cluster resources.

Desired SSI and Availability Services

We first identify all useful SSI services and availability functions. Then we use an example cluster configuration to illustrate the key concepts introduced.

SSI SERVICES

Ideally, a cluster should provide a variety of SSI services. Listed below are fundamental issues of SSI services in a cluster of computers. These services stretch along different dimensions of the application domain. Yet they are mutually supportive in nature.

- ***Single entry point*:** A user logins to the cluster as a single system (e.g., *telnet cluster.mycompany.com*), instead of login individual nodes as in a LAN environment (e.g., *telnet node1.cluster.mycompany.com*).
- ***Single file hierarchy* (SFH)**: Once logged in, the user sees a single hierarchy of file directories under the same root directory, just like a single file management environment for workstation users. Example SFHs include the NSF, AFS, xFS, and Solaris MC Proxy[3].
- ***Single control point***: The entire cluster is managed from a single place using a single GUI tool, much like an AIX workstation managed by the SMIT tool.
- ***Single virtual networking***: Single networking means any node can access any network connection throughout the cluster domain.
- ***Single memory space:*** This service gives users the illusion of a ***distributed shared memory*** (DSM) over local memories physically distributed over the cluster nodes. DSM enables shared-variable programming. Examples include the TreadMark, Wind Tunnel, and SHRIMP to be discussed in the sidebar titled "Middleware Packages for SSI and Availability".
- ***Single job management system* (JMS)**: Under a global job scheduler, a user job can be submitted from any node to request any number of host nodes to execute it. Concurrent job scheduling is possible either in batch, interactive, or parallel modes. Example JMSs for clusters include the GLUnix, LSF, and CODINE to be discussed in the same sidebar.
- ***Single user interface***: The users should be able to use the cluster through a single graphic interface. Such an interface is available for workstations (e.g., the *Common Desktop Environment*, or CDE) and PCs (e.g., Microsoft Windows 95). Developing a cluster GUI, one can leverage on the Web technology.

AVAILABILITY SUPPORT FUNCTIONS

Several availability support features are listed below. We will describe the concept of *single I/O space* in Section 8. The checkpointing and process migration are two mechanisms required to achieve fault tolerance and load balancing

- ***Single I/O space* (SIOS)**: This function allows any node to remotely access any I/O peripheral or disk devices without the knowledge of the physical location of the I/O devices. In this SIOS design, all distributed disks, RAIDs, and devices form a single address space.
- ***Single process space* (SPS)**: This requires mutual understanding between processes belonging to a single process space. They share a uniform process identification scheme. A process on any node can create (e.g., through a Unix fork) or communicate with (e.g., through signals, pipes, etc.) any other process on a remote node.
- ***Checkpointing and Process Migration* (CPM*)***: Checkpointing is a software mechanism to save the process state and intermediate computing results in memory or in disks

periodically. The purpose is to allow rollback recovery after a failure. Process migration dynamic load balancing among the cluster nodes. It also supports the checkpointing process.

AN ILLUSTRATIVE CLUSTER EXAMPLE

The example cluster in Fig. 3 has exactly one network connection. In two of the four nodes, each has two I/O devices attached to it. A properly designed cluster should behave like one system (the shaded area). In other words, the cluster behaves like a giant workstation with four network connections and four I/O devices attached. Any process on any node can use any network and I/O device as if it is attached to the local node.

Assume that the cluster is used as a Web server. The Web information database is distributed between the two RAIDs. An *http* daemon is started on each of the nodes to handle web requests, which come from all four-network connections. *Single I/O space* implies that any node can access the two RAIDs. Suppose most requests come from the ATM network. It would be beneficial if the functions of the http on node 3 could be distributed to all four nodes.

For *single point of control*, the system administrator should be able to configure, monitor, test, and control the entire cluster and each individual node from a single point. Many clusters aid this through a system console that is connected to all cluster nodes as shown in Fig.3. The system console is normally connected to an external LAN so that the administrators can remote login to the system console from anywhere on the LAN. Single point of control does not mean that solely the system console carries out all system administration work. The entire cluster is managed from a single place using a single GUI tool.

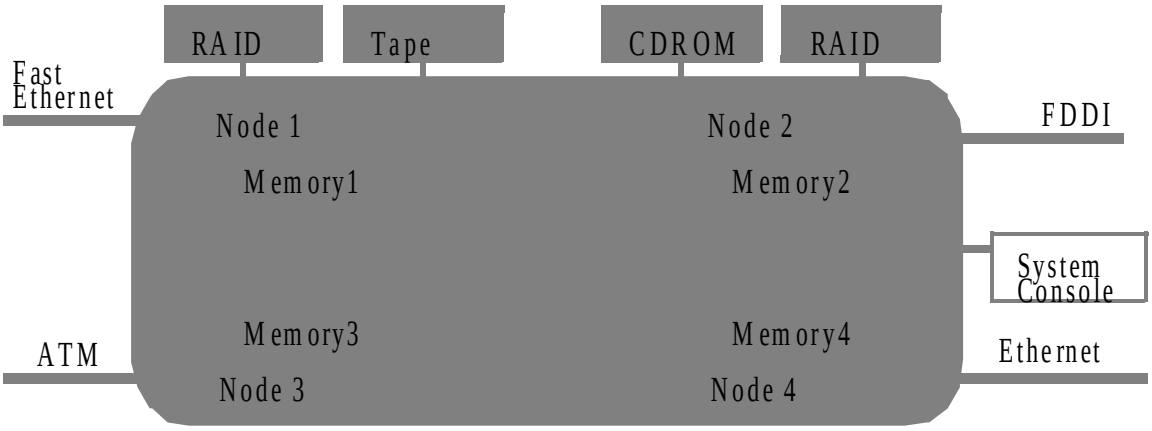


Figure 3 An example cluster of four workstations

Distributed RAID/Cluster Architectures

Three architectural design options are assessed below for enhancing the availability and fault tolerance of a cluster of workstations or PCs. The first two cluster architectures, namely the RADD and NASD, were previously proposed. The third one is newly proposed to combine the advantages of the two earlier architectures.

THE RADD ARCHITECTURE

The RADD (*Redundant Arrays of Distributed Disks*) architecture was first proposed by Stonebraker and Schloss[4] as a multicopy algorithm for distributed RAID systems. All local disks, attached to different cluster hosts, logically form the RADD subsystem. Normally, the checkpointing data blocks are stored in local disk blocks sequentially, while their parity is stored in the parity blocks residing in other local disks.

Among different nodes, the RAID-5 algorithm is applied only to handle local I/O operations, which are transparent to the higher-level RADD operations. For simplicity, one can simply apply the RAID-1 architecture on local disks. Mirroring on neighboring disks is implemented, but no parity among the distributed local disks.

THE NASD ARCHITECTURE

In NASD (*Network-attached Secure Disks*)[5], the RAIDs are directly attached to the network as a stable storage to allow shared access by all cluster nodes. Each workstation node in the cluster may or may not have local disk attached. Even with locally attached disks, they serve to buffer the data retrieved from the NASD to local nodes. NASD supports independent accesses by all nodes. Thus access conflicts must be resolved with a specially designed NASD controller.

The NASD architecture is quite different from the server-attached RAID. In the latter case, block data transfer must be done through the network server, instead directly from the network to end users at local workstations. The NASD improves in the area of scalability by removing the bottleneck problem on the network server. This is made possible by a technique called *network striping* as reported in Gibson et al[5].

THE CHECKPOINTING CLUSTER ARCHITECTURE

This architecture, originally proposed by Hwang and associates[6], is conceptually illustrated in Fig.4. The cluster nodes are either workstations or PCs. Gigabit LAN or SAN[7] connects all nodes. Local disks are attached to each workstation node. Each local disk is only accessible from its own host attached as depicted by the vertical arrows. All the local disks form a RADD.

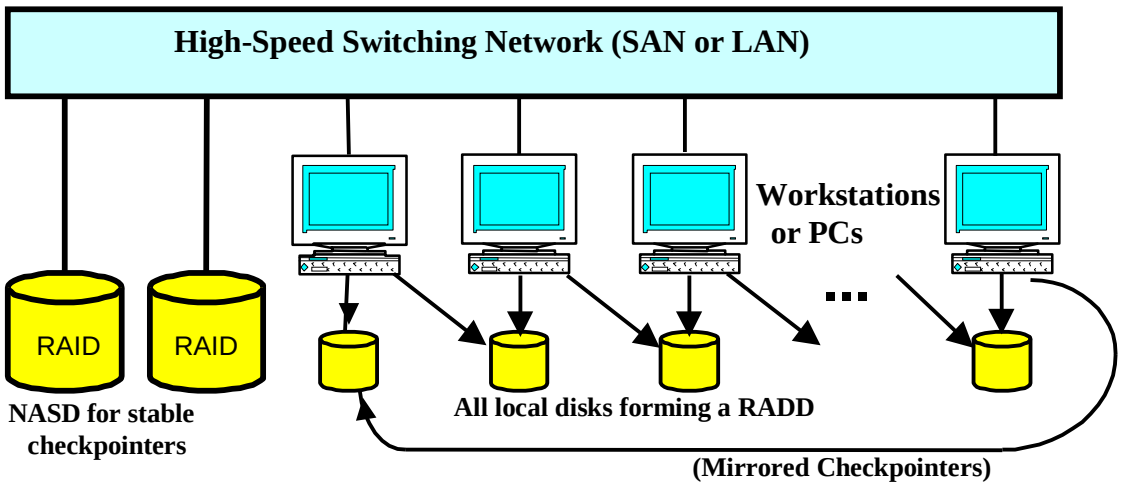


Figure 4 A fault-tolerant cluster architecture with distributed local disks (RADD) and network-attached RAIDs (NASD)

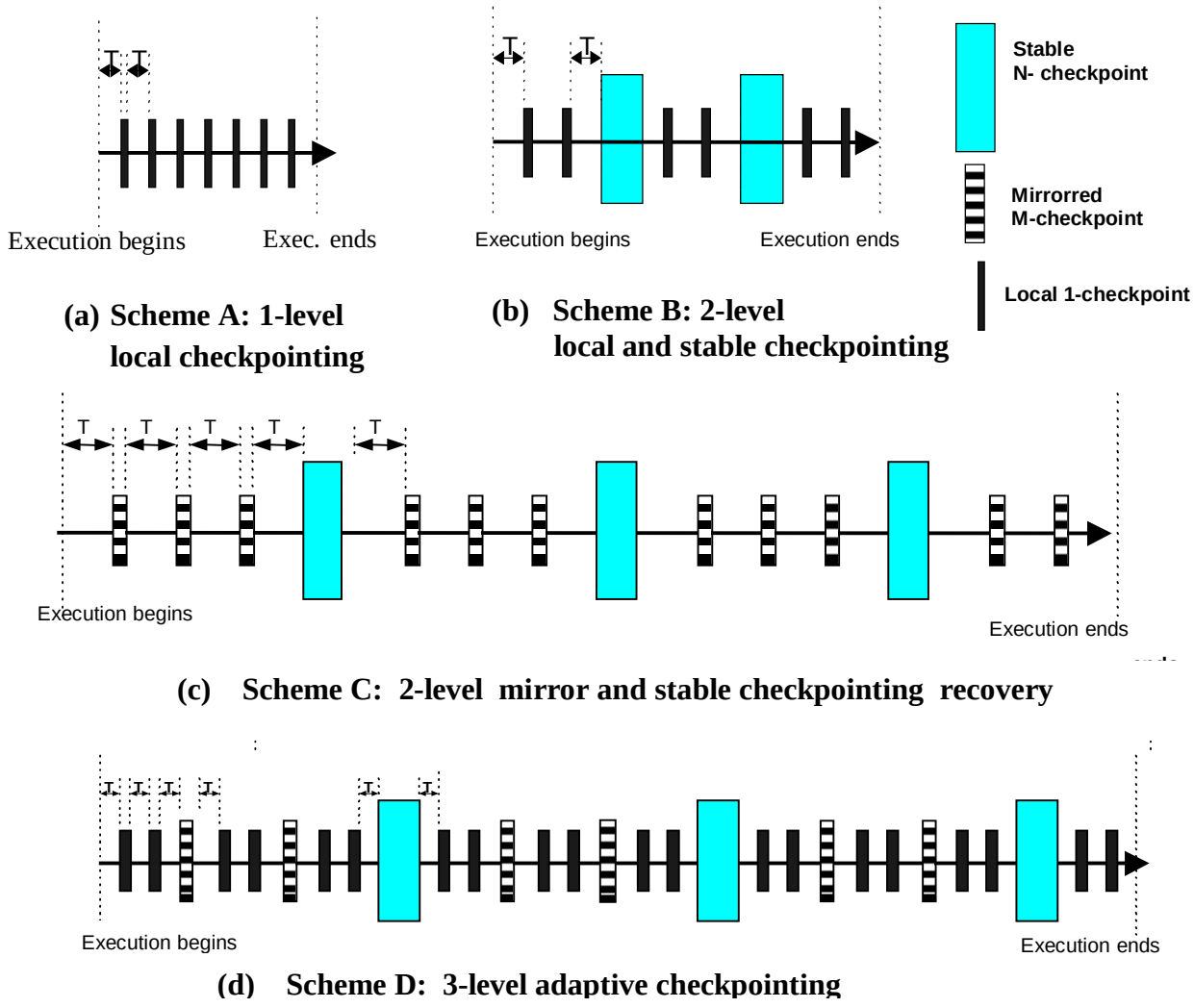
The network-attached RAIDs form a NASD as the stable storage for implementing various checkpointing schemes. We use *independent checkpointers*[6] over the coordinated checkpointers in order to reduce the checkpoint overhead and recovery latency. The uniqueness of this cluster design lies in its distributed checkpointing RAID architecture. This fault-tolerant cluster design is based on a new 3-level adaptive recovery scheme and a new single I/O space introduced in this paper.

Hierarchical Checkpointing Schemes

In general, the scenario of having a few failures in a cluster is more likely to occur than that of having a large number of simultaneous failures. Three checkpointers are suggested for designing checkpointing schemes. The timing charts of four checkpointing schemes are given in Fig.5. The checkpointer overhead corresponds to the bar width in the chart.

SCHEME A: LOCAL CHECKPOINTING

Figure 5a shows the simplest checkpointing scheme involving only the local disks. The host processor saves the 1-checkpointer periodically in the local disk. The major drawback of this scheme is its poor fault coverage. A single permanent failure will paralyze the node by losing the l-checkpointer and therefore prevent any chance of a rollback recovery.



(a) Scheme A: 1-level local checkpointing

(b) Scheme B: 2-level local and stable checkpointing

(c) Scheme C: 2-level mirror and stable checkpointing recovery

(d) Scheme D: 3-level adaptive checkpointing

Figure 5 Four hierarchical checkpointing and recovery schemes

SCHEME B: VAIDYA'S CHECKPOINTING/RECOVERY

Vaidya introduced this scheme[8] as a two-level recovery scheme (Fig.5b). It improves from Scheme A by tolerating local disk crashes, because of the allowed rollback to an N-checkpointer after a fixed number of *n* local checkpoints. Its major drawback is the large N-checkpoint overhead and recovery latency introduced. This scheme requires two levels of recovery. The recovery latency of this 2-level rollback improves sharply from the 1-level rollback in Scheme A.

SCHEME C: INTERLEAVED MIRROR AND STABLE CHECKPOINTING

This scheme saves the M-checkpoints in mirrored disks and the N-checkpoints in the stable storage (Fig. 5c). Every *m*-th consistent checkpoint is stored in the stable storage, while all other checkpoints are stored in local memories (instead of local disks) and mirrored in neighbor's

disk. This scheme rolls back to a local memory for transient failure.

For permanent failures, it rolls back to a mirrored copy of the checkpoint. The scheme rolls back to the stable storage, only when there is a permanent failure immediately following an N-checkpointer. The recovery latency reduces further from Scheme B. As a matter of fact, this scheme results in the lowest recovery latency, among the four schemes.

SCHEME D: ADAPTIVE CHECKPOINTING AND RECOVERY

Figure 5d shows that the host processor periodically saves the 1-checkpoints to its local disk. Every *m*-th checkpoint is saved as an M-checkpoint. Every *mn*-th consistent checkpoint is stored on the stable storage, while all other checkpoints on local disks. Three levels of recovery are possible under this checkpointing scheme as described in the next section. The recovery latency is slightly higher than that in Scheme C. But the latency gap diminishes as the cluster size grows larger.

Adaptive Recovery with Reduced latency

The Scheme D offers an adaptive, three-level, recovery scheme. The adaptive rollback recovery is illustrated in Fig.6. The adaptability leads to a much reduced recovery latency. The 3 levels of recovery are stated below:

- **Level-1 recovery** – Rollback to a 1-checkpoint, when a processor has a transient failure which does not immediately follow an M-checkpoint or an N-checkpoint.
- **Level-2 recovery** – Rollback to an M-checkpoint, when a node has a permanent failure, assuming no adjacent failures occur and they do not follow an N-checkpoint.
- **Level-3 recovery –** Rollback to the stable storage checkpoint (N-checkpoint), when there is a failure immediately following an N-checkpoint or when a processor or a local disk has a permanent failure and its mirrored checkpoint is lost.

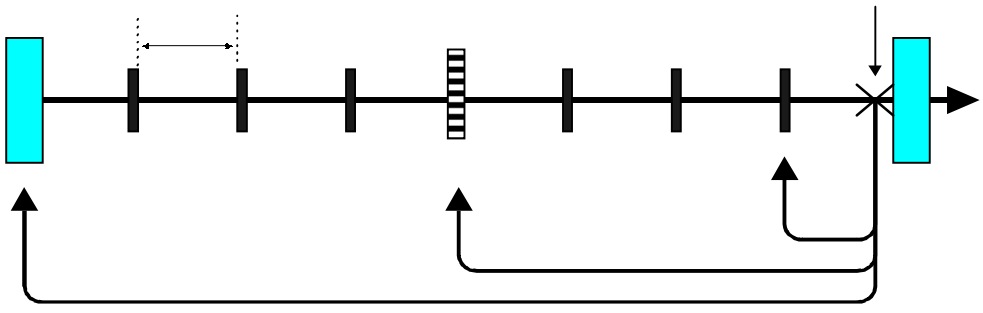


Figure 6. Adaptive rollback recovery in using Scheme D at three levels of the distributed disk hierarchy

In the worse case, the loss of useful computation for an M-recovery is *m*T, while the loss of useful computation for an N-recovery is *mn*T, where T is the checkpointing period. With much higher recovery latency due to network latency and simultaneous access of the central stable storage, the N-recovery latency is expected to be much larger than the M-recovery latency. This will reduce the probability of an N-rollback to the stable storage. The M-recovery shortens the expected recovery latency significantly.

Figure 7 compare the *expected recovery latency* of four recovery schemes in the worst case. Scheme A has the shortest checkpointing overhead but the longest recovery latency, which is independent of the cluster size. Essentially, Scheme A can only tolerate transient fault. For permanent fault, Scheme A must roll back all the way to the beginning of the job execution. The recovery time of scheme A is about 5 times higher than that of Scheme B in Fig.7.

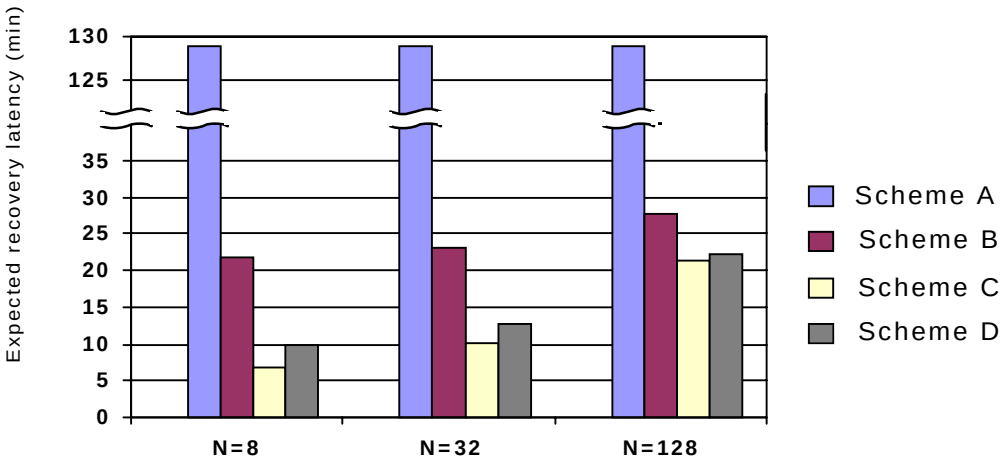


Figure 7 Expected recovery latency of four checkpointing schemes

The expected recovery latencies for Schemes B, C and D are much shorter than that of Scheme A, because they can roll back either to the image checkpoint or to the last N-checkpoint. Among them, the Scheme C has the shortest recovery latency, because rolling back to an M-checkpoint at the neighboring disk is much faster than rolling back to an N-checkpoint in the stable storage.

In Fig.7, Scheme D takes slightly longer time to recovery than Scheme C does. This is because D takes longer time to recover from an M-checkpoint. For large clusters with more than 128 nodes, both Schemes C and D perform almost equally, because the probability of rolling back to the mirrored checkpointer increase faster in Scheme D than that in Scheme C.

(Beginning of Sidebar B)

Middleware Packages for Supporting SSI and Availability in Clusters

Cluster design concerns the size scalability, enhanced availability, system manageability, fast message passing, security and encryption, and distributed computing environments. Table 2 summarizes four representative middleware packages for SSI and availability support. The GLUnix is available in the public domain and the rest are commercial products. The features in the first 7 rows support SSI services. The next eight facilitate job management and parallel programming. The remaining four are for availability and fault tolerance.

Table 2 Four Middleware Packages for SSI and Availability Services

Support Features	GLUnix	TreadMarks	Codine	LSF
Single control point	Yes	No	Yes	Yes
Single entry point	Yes	No	No	No
Single file hierarchy	Yes	Yes	Yes	Yes
Single memory space	No	Yes	No	No
Single process space	Yes	No	No	No
Single I/O space	No	No	No	No
Single networking	No	No	No	No
Batch support	Yes	No	Yes	Yes
Interactive support	Yes	Yes	Yes	Yes
Parallel support	Yes	Yes	Yes	Yes
Load balancing	Yes	No	Yes	Yes
Job monitoring	Yes	No	Yes	Yes
Suspend/Resume	No	No	Yes	Yes
Dynamic resource	Yes	No	Yes	Yes
User interface	cmd-line	cmd-line	GUI/cmd-line	GUI
Checkpointing	No	No	Yes	Yes
Process migration	No	No	Yes	Yes
Security standard	Unix	Unix	Kerboros	Kerboros
Fault tolerance	Yes	No	Yes	Yes

The *batch support* means the dedicated use of cluster resources by a single user job in a batch mode. The *interactive support* refers the time-sharing use of cluster resources by multiple users simultaneously. The *parallel support* refers to the management of parallel processes, MPI and PVM environments. So far, none of the four middleware packages has implemented the single I/O space and single networking features.

Process migration is needed to achieve dynamic load balancing among cluster nodes or to enable fail over after a detected failure. Process migration enables load distribution, fault resilience, system administration, and improved data access locality. Migrating processes from overloaded nodes to lightly loaded ones can achieve the load distribution. Migrating processes from nodes that may have partial failure can achieve the fault resilience.

Job monitoring, suspend/resume, dynamic resources, and user interfaces are useful features to provide an user-friendlier environment in program debugging, performance evaluation, resource allocation, and program optimization. The fault-tolerance and checkpointing are features to enhance the cluster system availability. To achieve transparency, the JMS should be able to dynamically reconfigure the cluster with minimal impact on the running jobs.

To make the JMS scalable, the functionality of a JMS is often distributed. For instance, a user server may reside in each host node, and the resource manager may span all over the cluster nodes. The system can migrate process from the nodes that are about to be shutdown so that the system administrator doesn't need to wait until user logout. Migrating processes towards the source of the data can also improve the data access locality.

THE GLUnix AT BERKELEY NOW PROJECT

GLUnix stands for Global Layer Unix as named in Anderson et al[1]. It can be easily downloaded to any cluster through the public domain. This global layer provides a single system image of the nodes in a cluster, so that all of the processor, memory, network capacity, and disk bandwidth can be allocated for sequential and parallel applications. The global layer is realized as a protected, user-level, operating system library that is dynamically linked to every application. The library intercepts all system calls and realizes them as procedure calls.

Distinct advantages of the GLUnix package include the following aspects. It is easy to implement and to modify the source code of GLUnix at user level. The package supports co-scheduling of parallel programs, idle resource detection, process migration, and load balancing, remote paging, and some availability support. It was deigned to port to any OS that supports interprocess communication, process signaling, and access to loading information.

THE ThreadMarks AT RICE UNIVERSITY

To enable shared-memory computing on NORMA (*no remote memory access*) and non-CC-NUMA systems, researchers have proposed the *software-coherent NUMA* (SC-NUMA) memory model, also known as the ***distributed shared-memory*** (DSM) model. Example DSM systems include the TreadMarks project at Rice University[2], the Wind Tunnel project at University of

Wisconsin[3], and the SHRIMP project at Princeton University[4]. A software-implemented DSM relies on software extensions to achieve single memory address space and consistency control.

THE CODINE DERIVED FROM THE DQS

The CODINE package is evolved from the *Distributed Queuing System* (DQS) created at Florida State University. CODINE is offered by GENIAS Software GmbH in Germany. GENIAS claims that this package has become a de facto JMS in Europe[5]. The major strength of CODINE lies in hardware and software resources management in a heterogeneous networked environment. The CODINE uses GUI tools to provide a SSI of cluster resources. Checkpointing is supported only if the kernel supports it. Kernel checkpointed jobs can be migrated. Resources can be dynamically added or deleted from a resource pool.

THE LSF FROM PLATFORM COMPUTING

Load Sharing Facility **(**LSF)[6] from Platform Computing is evolved from the Utopia system developed at University of Toronto. This package emphasizes job management and load sharing of both parallel and sequential jobs. In addition, it has support for checkpointing, availability, and load migration. Checking the entries in Table 2, CODINE and the LSF are almost equally capable of supporting the same set of SSI and availability features.

References

1. T. E. Anderson, H. M. Levy, B. N. Bershad, and E. D. Lazowska, "The Interaction of Architecture and Operating System Design", *Proc. of the 4th Int'l Conf. on Architecural Support for Programming languages and OS*, 1991, pp.108 – 120.
2. C. Amza, A. L. Cox, S. D. Warkadas, P. keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations", *IEEE Computer*, Vol.29, No. 2, 1996, pp.18-28
3. S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood, "The Wisconsin Wind Tunnel: Virtual prototyping of Parallel Computers", *Proc. of 1993 ACM SIGMETRICS Conference*, 1993, pp.48-60
4. E. W. Felten, R. D. Alpert, A. Bilas, M. A. Blumrich, D. W. Clark, S. N. Damianakis, C. Dubnicki, L. Iftode, and K. Li, "Early Experience with Message-Passing on the SHRIMP Multicomputer", *Proceedings of 23rd International Symposium on Computer Architecture*, 1996
5. GENIAS Software GmbH, "CODINE: Computing in Distributed Networked Environments", September,1995,Germany, an on-line file accessed from URL address http://www.genias.de/genias/english/codine/Welcome.html.
6. S. Zhou, "LSF: Load Sharing and Batch Queuing Software", Platform Computing Corporation, North York, Canada, 1996.

(End of sidebar B)

RELATIONSHIP AMONG MIDDLEWARE PACKAGES

In Fig. 8, we show the functional relationships among six key middleware packages. These middleware packages are used as interfaces between user applications and cluster hardware and OS platform. They support each other at the management, programming, and implementation levels.

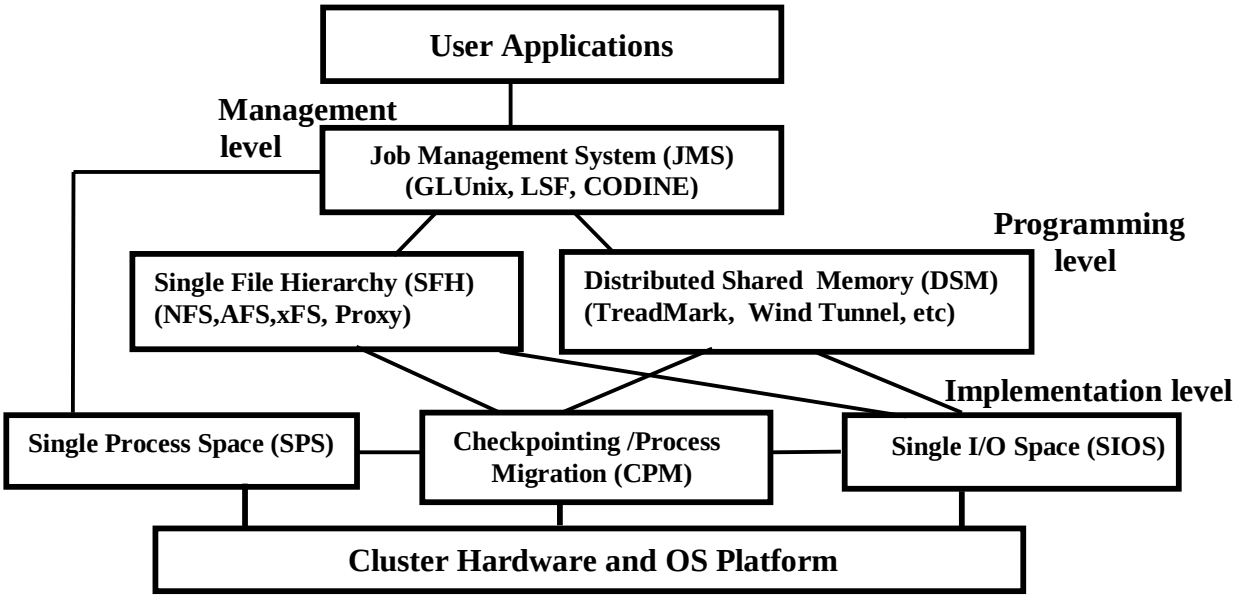


Figure 8 Relationships among supporting middleware modules

The JMS is essentially a global job scheduler. The SFH and DSM support distributed file management and shared-memory programming, respectively. The SPS, CPM, and SIOS provide support to implement the JMS, SFH, and DSM services. All middleware packages work together to support the desired availability and SSI services. This paper presents the development of the CPM mechanisms and functional design of the SIOS module.

Single I/O Space Design

In Solaris MC[3], an uniform device naming scheme was developed to achieve a *single I/O space* (SIOS) in addressing any peripheral or disk devices attached to the Unix cluster of Sun workstations. A device address consists of a node number and the device number. A process can access any device by initiating a system call at a remote processor using this uniform address. remote node. We consider the Solaris MC definition of SIOS a rather restricted one, because it only accesses remote devices as a whole, not the remote disk memory by blocks or by strips.

On the other hand, a *single address space operating system* (SASOS)[9] was developed at University of California at Berkeley. This system is primarily designed for use as a distributed

operating system for a network of independent and homogeneous computers, not for the SSI clusters in which we are interested. Therefore, SASOS is not a SIOS by our definition.

Examining the middleware modules in Fig.8, we see that the SIOS enables efficient implementations of the DSM, SFH, and CPM functions. The SIOS is a low-level construct, which supports the DSM, but cannot replace the DSM. We broaden the SIOS functions by making the remote access transparent, instead of using the system call. Our addressing scheme is initiated by local hosts, rather the remote hosts where the device is attached.

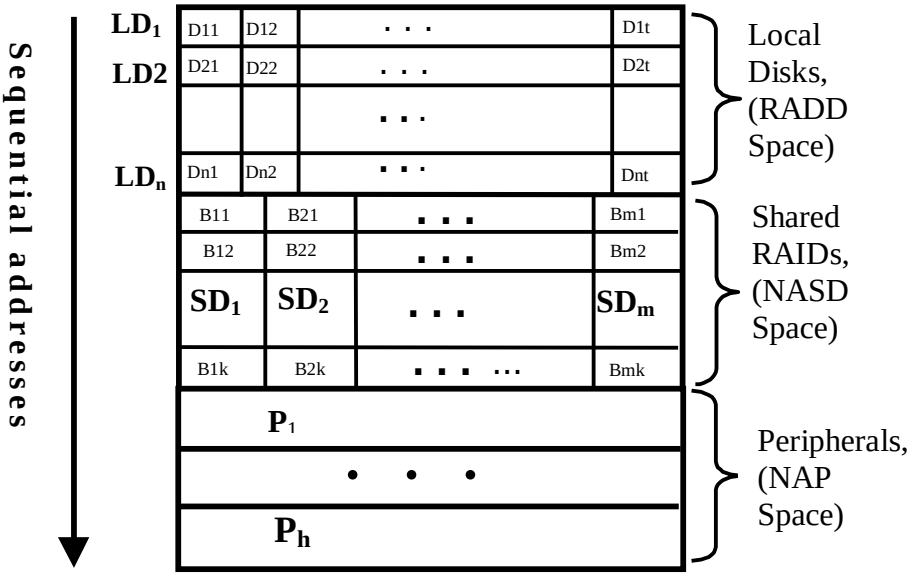
ADVANTAGES OF SINGLE I/O SPACE

The proposed SIOS is implemented primarily at user level. But some local OS system calls must be modified to enable remote disk accesses. Four distinct advantages of the extended single I/O space design are identified below.

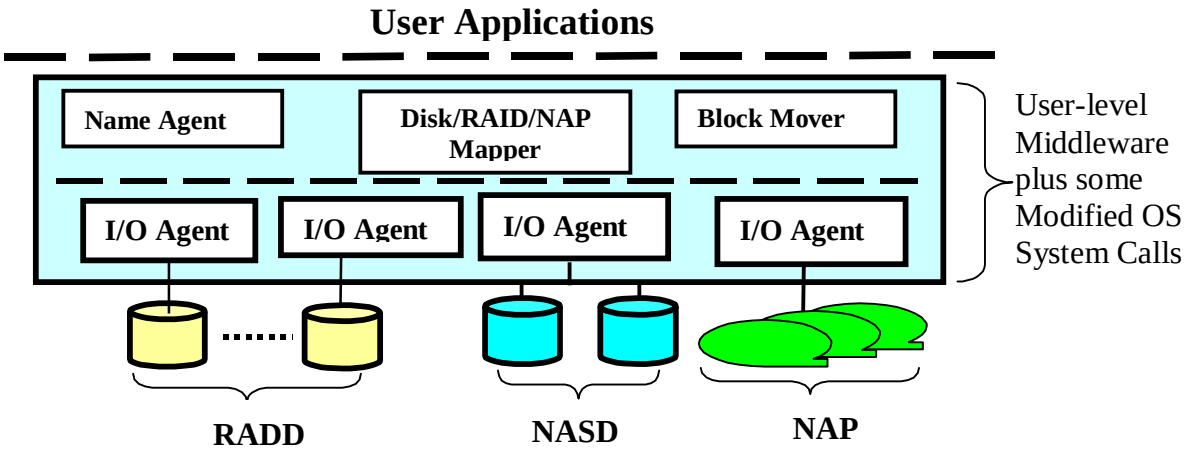
- Firstly, the SIOS addressing elimiates the gap between accessing local disk and remote disk. Remote disk access does not have to be handled as system calls from the remote host. Instead, the local host can address the remote disks directly by a modified system call.
- Secondly, the SIOS supports a persistent programming paradigm. The DSM and SFH can be more easily implemented with the SIOS. All device types and their physical locations are now transparent to all users. At present, the MPI-based I/O cannot achieve this transparency.
- Thirdly, the SIOS allows stripping on remote disks. This will accelerate parallel I/O operations, often needed in moving large data files among the cluster nodes.
- Finally, the SIOS greatly facilitates the implementation of distributed checkpointing and recovery scheme we have suggested in Sections 4 and 5. Mirroring on remote disks becomes faster and easy to control. Recovery from the shared RAIDs or NASD requires less time than current systems without the SIOS.

THE SIOS DESIGN AT USC AND HKU

This design was jointly developed at the University of Southern California and at the University of Hong Kong. As shown in Fig,9, the integrated I/O address space covers *n* local disds, *m* shared disks in the RAID, and *h* peripheral devices. The sequential address space is essentially a single, large, linear array of addresses down to the disk block level or to the device number level. For simplicity, we conder *t* blocks per local disk and *k* blocks per disk in the RAID. All NAP devices are assigned with high-order addresses in the linear array. These high-order addresses preserved the uniform naming scheme built in the Solaris MC^3.



(a) The integrated I/O address space



(b) **Addressing and mapping mechanisms**

Figure 9 Single I/O address space design (RADD: redundant arrays of distributed disks, NASD: network-connected secure disks, NAP: network-attached peripheral devices)

Data mapping onto the distributed local disks uses parity blocks, mirroring, or shadowing[6]. In order to imporve the efficiency of I/O access, user data sets are stored on local disks with ckeckpointing or parity information stored on one or more remote disks. Sequential addesses are assumed horizontally in each local disk (LD_i) in Fig.9a. Addresses in the shared RAID disks are interleaved horizontally across the vertical disks (SD_j) in the RAID array. This address scheme can be implemented by hardware, or firmware, or software.

The entire SIOS is organized as a single linear array of sequential addresses according to the following ordering among the local disks, shared RAID disks, and NAP devices:

$D_{11}, D_{12}, \ldots, D_{1t}, D_{21}, D_{22}, \ldots, D_{2t}, \ldots, D_{n1}, D_{n2}, \ldots, D_{nt},$
$B_{11}, B_{21}, \ldots, B_{m1}, B_{12}, B_{22}, \ldots, B_{m2}, \ldots, B_{1k}, B_{2k}, \ldots, B_{mk},$
$P_1, P_2, \ldots, P_h.$

Figure 9b shows the middleware functional modules needed to implement the SIOS for a cluster of workstations. The *Name Agent* maps the name known to the I/O Agent onto the unique device number known to the *Disk/RAID/NAP Mapper*. The Mapper uses the techniques of striping and replication[4, 5] to implement the address mapping.

The *Block Mover* is responsible for copying data to and from the storage media and transmiting the data from a source to a destination. Each I/O agent performs the I/O operations according to the I/O type. Multiple *I/O Agents* are used to interface local dsiks, DASD, or peripheral devices. The agents receive the I/O command from the Mapper or from the Block Mover. The agent performs low-level I/O operations under the control of the local system call.

The user-level agents and functions can be written as Java processes, which will faciliate the porting to various host platforms. The system call for remote disk access requires to change the page-fault mapping table in the kernel. In an experimental cluster construction at USC, we modify the device-relevant system calls in Linux to run on Pentium-based PC hosts. The mapper, the mover, and various agent routines form the SIOS middleware package. We aim to make the SIOS package portable to all major OS platforms in the future.

DATA MOVEMENT PROCEDURES

Figure 10 illustrates the request and return procedures of moving data blocks between distributed local disks and the shared RAID disks. User application on Node 1 requests a data block A residing on a local disk LD_2 of Node 2 or from a shared disk SD_i of the NASD.

By checking with the Name Agent and the Mapper, the I/O Agent in Node 1 sends the request to the Block Mover to copy a data block A from Node 2 to Node 1. After the I/O Agent in Node 2 receives the request from the Block Mover, it performs an I/O operation to get the data block A from its local storage. The I/O Agent then sends block A to the Block Mover and transmits it to the I/O Agent in Node 1.

Conclusions

Clusters of workstations or PC clusters are bound to become commodity products in the computer industry. However, the major difficulty in clustering lies in lack of adequate SSI software support. To build multicomputer clusters with SSI, efficient mechanisms or protection schemes are needed for global interprocess communication and for global security control without worry about access conflicts or unauthorized access of shared resources in the cluster.

The SSI and fault tolerant clusters are high goals that are nontrivial to achieve. It makes such a cluster construction a lot easier, systematic, and more efficient to use a single I/O space across all disks and I/O devices based on the proposed checkpointing RAID architecture[6]. We

need numerous SSI services, including single point of control, a single memory space, a single I/O space, single networking, a global file hierarchy, and a global job management system.

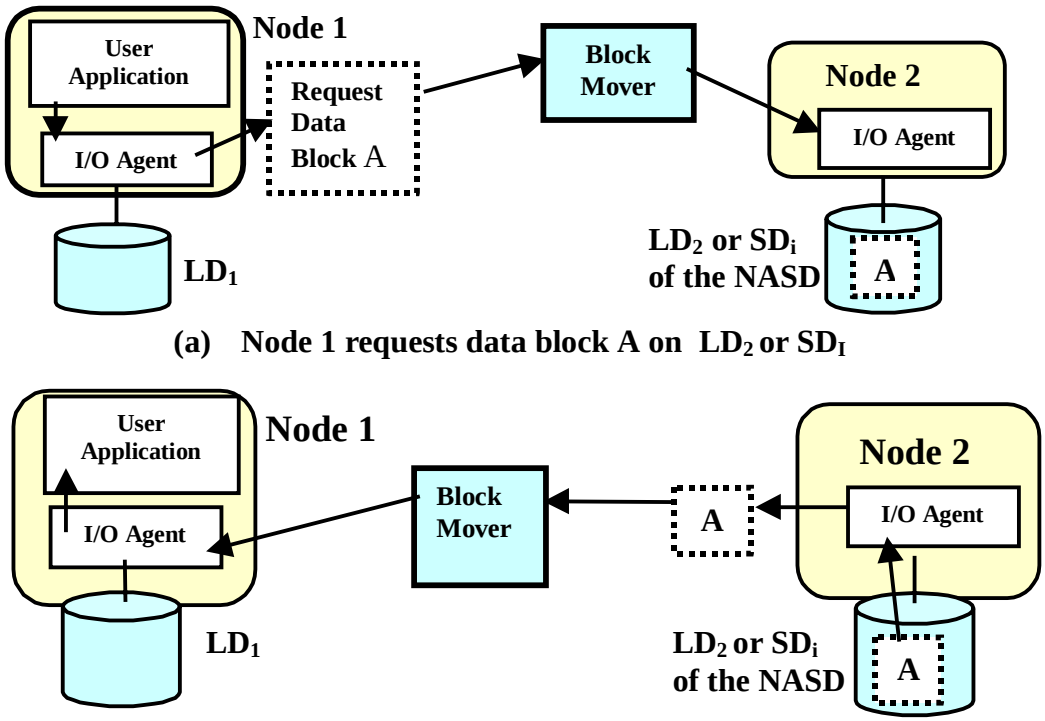


Figure 10 Moving data blocks between distributed disks and shared RAIDs

We demand total transparency not only in resources allocation and replication, but also in the control and management of concurrency and parallelism in clusters of workstations or PCs. Furthermore, checkpointing, high availability, and fault tolerance are equally important to cluster operations. The size of clustered platforms become larger and larger due to the use of commodity hardware, software, and middleware in cluster construction.

A dynamic mechanism is desired to effectively manage the fast changing cluster environment. Load balancing and process migration are necessary. This adds to another dimension of challenge in clustering of multiple computers. In the industrial track, Wolfpack for Intel-based Windows NT servers[10], NOW and Solaris MC[3] for Unix workstations are all aimed at high availability, scalability, and manageability.

On the other hand, the rapid growth in multimedia and the WWW applications has further increased the demand of clustered and network-based platforms. These applications demand higher computing power, higher communication bandwidth, more flexible control of the cluster resources, and higher availability and fault tolerance. SSI clusters or robust clusters will efficiently meet these requirements. The use of an unified or adaptive communication protocol

is in demand. In addition, when the cluster is exposed to public networks, security control becomes the most critical concern in cluster applications.

The Internet-based *metacomputing*[11] uses a large number of remote hosts from the Internet to form a supercluster. Any user sitting in front of a PC or workstation can utilize resources in the supercluster. However, metacomputing cannot be easily arranged due to the ownership problem associated with scattered workstations and PCs. Lack of software support to manage the huge Internet-connected resources makes it impractical to practice metacomputing at this time.

Java-based intelligent agents are suitable for distributed cluster computing. This applies especially to distributed financial computing[12], information retrieval in digital libraries, and electronic business areas. Furthermore, PC or workstation clusters demonstrate the potential in large-scale database search or datamining applications. Robust clusters can be more cost-effectively used in bioinformatics, telemedicine, telemarketing, data warehousing, and distance learning than using today's mainframes or supercomputers.

ACKNOWLEDGEMENTS

This research was jointly carried out at the Cluster Computing Research Laboratory of the University of Southern California (USC) and the High-Performance Computing Research Laboratory at the University of Hong Kong (HKU). The research was supported partially by the Hong Kong RGC Grants HKU 2/96C, HKU 7022/97E, and HKU 7032/98E, by the development fund of the HKU Area of Excellence in Information Technology, and by a research grant from the School of Engineering, USC.

REFERENCES

1. G. F. Pfister, "The Varieties of Single System Image", *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1993, pp.59-63
2. M.A. Baker, G.C. Fox and H.W. Yau, "Cluster Computing Review", Northeast Parallel Architectures Center, Syracuse University, Nov. 1995.
3. Y. A. Khalidi, J. M. Bernabeu, V. Matena, K. Shirriff, and M. Thadani, "Solaris MC: A Multicomputer OS", *Proc. of the 1996 USENIX Conference*.
4. M. Stonebraker and G. A. Schloss, "Distributed RAID – a New Multiple Copy Algorithm", *Proc. of the Sixth International Conference on Data Engineering*, Feb. 1990, pp.430-43
5. G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg and J. Zelenka, "File Server Scaling with Network-Attached Secure Disks", *Proceedings of the ACM Int'l Conference on Measurement and Modeling of Computer Systems (Sigmetrics '97)*, June 1997.
6. K. Hwang, E. Chow, and H. Jin, "Distributed Checkpointing RAID Architecture for fault-Tolerance in Clusters of Workstations", *Technical Report*, Dept. of Electrical and Electronics Engineering, University of Hong Kong, October 1998.
7. K. Hwang, "Gigabit Networks for Scalable Multiprocessors and Multicomputer Clusters", *Transactions of Hong Kong Institute of Engineers* , Dec. 1997, pp. 82-87.
8. N. H. Vaidya, "A Case for Two-Level Distributed Recovery Schemes", *Proceedings of the ACM Int'll Conf. on Measurement and Modeling of Computer Systems*, pp.64-73.

9. J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska, "Sharing and Protection in a Single-Address-Space Operating System", *ACM Transactions on Computer Systems*, Vol.12, No. 4, November 1994

10. M. Smith, "Closing on Clusters: Will Wolfpack Dominate the High-Volumn Windows NT Cluster Markets?", *Windows NT Magazine*, Aug. 1996

11. C. Catlett and L. Smarr, "Metacomputing," *Comm. of ACM*, Vol. 35, No. 6, June 1992, pp. 44-52.

12. J. Yen, K. Hwang, and A. Lau, "Multi-Agent Computing on Workstation Clusters for Distributed Economic Analysis", *Technical Report*, Dept. of Computer Science and Information Systems, University of Hong Kong, Nov. 1998.

Kai Hwang is a Professor of Electrical Engineering and Computer Science at the University of Southern California. He has engaged in higher education and computer research for 25 years, after earning the Ph.D. degree from the University of California at Berkeley. An IEEE Fellow, he specializes in computer architecture, digital arithmetic, and parallel processing. Dr. Hwang is the founding Editor-in-Chief of the *Journal of Parallel and Distributed Computing*. He received the Outstanding Achievement Award from the PDPTA in 1996.

He has published six books and 150 technical papers in computer architecture and parallel processing, lectured worldwide, and performed consulting and advisory work for IBM, MIT Lincoln Lab., Fujitsu and ETL in Japan, CERN Computing School in the Netherlands, and the GMD in Germany. His current research interest lies in network-based multicomputer cluster architecture and middleware development for availability and SSI in cluster applications. He can be reached by Email: **kaihwang@ceng.usc.edu** or by post mail: Dept. of Electrical Engineering-Systems, EEB Rm.106, University of Southern California, Los Angeles, CA. 90089, USA.

Edward Chow is a postgraduate student at the University of Hong Kong, where he received the B.S. degree in Electrical and Electronic Engineering in 1996. He is currently working on his M.Phil. degree at the Computer Engineering program of HKU. His research interest lies mainly in fault-tolerant computer architecture design and evaluation.

Hai Jin is an Associate Professor of Computer Science at Huazhong University of Science and Technology in China, where he received the Ph.D. degree in 1994. Presently, he visits and works at High-Performance Computing Research Laboratory, University of Hong Kong, where he participated in the HKU PeralCluster research project. His research interests cover the parallel I/O, RAID architecture design, and cluster benchmark experiments. He can be reached by Email: **hjin@eee.hku.hk**.

Cho-Li Wang received his B.S. degree in Computer Science and Information Engineering from National Taiwan University in 1985. He obtained his M.S. and Ph.D. degrees in Computer Engineering from University of Southern California in 1990 and 1995 respectively. He is an Assistant Professor of Computer Science at the University of Hong Kong. His research interests include high-speed networking, computer architectures, and software environment for distributed multimedia applications. Wang can be reached by Email: **clwang@cs.hku.hk**.

Zhiwei Xu is a Professor and Chief Architect at the National Center for Intelligent Computing Systems (NCIC), Chinese Academy of Sciences, Beijing, China. He received the Ph.D. from the University of Southern California. Presently, he leads a design group at NCIC building a series of cluster-based superservers and scalable multicomputers. His current research interest lies mainly in network-based cluster computing, parallel programming, and software environments. Xu can be reached by Email: **zxu@apple.ncic.ac.cn.**