Dynamic Load Balancing for I/O-Intensive Tasks on Heterogeneous Clusters

Xiao Qin, Hong Jiang, Yifeng Zhu, David R. Swanson

Department of Computer Science and Engineering
University of Nebraska - Lincoln, Lincoln, NE, USA.
Email: {xqin, jiang, yzhu, dswanson}@cse.unl.edu

Abstract. [1] Since I/O-intensive tasks running on a heterogeneous cluster need a highly effective usage of global I/O resources, previous CPU- or memory-centric load balancing schemes suffer significant performance drop under I/O-intensive workload due to the imbalance of I/O load. To solve this problem, we develop two I/O-aware load-balancing schemes, which consider system heterogeneity and migrate more I/O-intensive tasks from a node with high I/O utilization to those with low I/O utilization. If the workload is memory-intensive in nature, the new method applies a memory-based load balancing policy to assign the tasks. Likewise, when the workload becomes CPU-intensive, our scheme leverages a CPU-based policy as an efficient means to balance the system load. In doing so, the proposed approach maintains the same level of performance as the existing schemes when I/O load is low or well balanced. Results from a trace-driven simulation study show that, when a workload is I/O-intensive, the proposed schemes improve the performance with respect to mean slowdown over the existing schemes by up to a factor of 8. In addition, the slowdowns of almost all the policies increase consistently with the system heterogeneity.

1 Introduction

Dynamic load balancing schemes are widely recognized as important techniques for the efficient utilization of resources in networks of workstations or clusters. Many load balancing polices achieve high system performance by increasing the utilization of CPU [1], memory [2], or a combination of CPU and memory [3]. However, these load-balancing policies are less effective when the workload comprises a large number of I/O-intensive tasks and I/O resources exhibit imbalanced load.

We have proposed two I/O-aware load-balancing schemes to improve overall performance of a distributed system with a general and practical workload including I/O activities [4][5]. However, it is assumed in [4][5] that the system is homogeneous in nature. There is a strong likelihood that upgraded clusters or

[1] This paper appears in the proceedings of the 2003 International Conference on High Performance Computing (HiPC03).

networked clusters are heterogeneous, and heterogeneity in disks tends to induce more significant performance degradation when coupled with imbalanced load of memory and I/O resources. Therefore, it becomes imperative that heterogeneous clusters be capable of hiding the heterogeneity of resources, especially that of I/O resources, by judiciously balancing I/O work across all the nodes in a cluster. This paper studies two dynamic load balancing policies, which are shown to be effective for improving the utilization of disk I/O resources in heterogeneous clusters.

There is a large body of literature concerning load balancing in disk systems. Scheuermann et al. [6] have studied the issues of striping and load balancing in parallel disk systems. To minimize total I/O time in heterogeneous cluster environments, Cho et al. [7] have developed heuristics to choose the number of I/O servers and place them on physical processors. In [8][9], we have studied dynamic scheduling algorithms to improve the read and write performance of a parallel file system by balancing the global workload. The above techniques can improve system performance by fully utilizing the available hard drives. However, these approaches become less effective under a complex workload where I/O-intensive tasks share resources with many memory- and CPU-intensive tasks.

Many researchers have shown that I/O cache and buffer are useful mechanisms to optimize storage systems. Ma et al. [10] have implemented active buffering to alleviate the I/O burden by using local idle memory and overlapping I/O with computation. We have developed a feedback control mechanism to improve the performance of a cluster by manipulating the I/O buffer size [11]. Forney et al. [12] have investigated storage-aware caching algorithms in heterogeneous clusters. Although we focus solely on balancing disk I/O load in this paper, the approach proposed here is capable of improving the buffer utilization of each node. The scheme presented in this paper is complementary to the existing caching/buffering techniques, thereby providing additional performance improvement when combined with active buffering, storage-aware caching, and a feedback control mechanism.

The rest of the paper is organized as follows. Section 2 describes the system model. Section 3, we propose two I/O-aware load-balancing policies. Section 4 evaluates the performance. Section 5 concludes the paper.

2 Workload and System Model

In this paper, we consider a cluster as a collection of heterogeneous nodes connected by a high-speed network. Tasks arrive at each node dynamically and independently, and share resources available there. Each node maintains its individual task queue where newly arrived tasks may be transmitted to other nodes or executed in the local node, depending on a load balancing policy employed in the system. Each node keeps track of reasonably up-to-date global load information by periodically exchanging load status with other nodes.

We address heterogeneity with respect to a diverse set of disks, CPUs, and memories. We characterize each node i by its CPU speed C_i, memory capacity

M_i, and disk performance D_i. Let B_i^{disk}, S_i, and R_i denote the disk bandwidth, average seek time, and average rotation time of the disk in node i, then the disk performance can be approximately measured as the following equation: $D_i = \frac{1}{S_i + R_i + d/B_i^{disk}}$ where d is the average size of data stored or retrieved by I/O requests.

The weight of a disk performance W_i^{disk} is defined as a ratio between its performance and that of the fastest disk in the cluster. Thus, we have $W_i^{disk} = \frac{D_i}{max_{j=1}^n(D_j)}$. The disk heterogeneity level, referred to as H_D, can be quantitatively measured by the standard deviation of disk weights. Thus, H_D is expressed as:

$$H_D = \sqrt{\frac{\sum_{i=1}^n (W_{avg}^{disk} - W_i^{disk})^2}{n}} \tag{1}$$

where W_{avg}^{disk} is the average disk weight. Likewise, the CPU and memory heterogeneity levels are defined as follows:

$$H_C = \sqrt{\frac{\sum_{i=1}^n (W_{avg}^{CPU} - W_i^{CPU})^2}{n}}, H_M = \sqrt{\frac{\sum_{i=1}^n (W_{avg}^{mem} - W_i^{mem})^2}{n}} \tag{2}$$

where W_i^{CPU} and W_i^{mem} are the CPU and memory weights, and W_{avg}^{CPU} and W_{avg}^{mem} are the average weights of CPU and memory resources [3].

We also assume that the network provides full connectivity in the sense that any two nodes are connected through either a physical link or a virtual link. We assume that arriving tasks are either sequential or otherwise parallel applications can be broken into a number of tasks with individual and independent resource requirements.

Each task is assumed to read or write data locally, and this amount of data, referred to as initial data, is required to be shipped along with a migrated task. This assumption is conservative in the sense that it makes our approach less effective, because the migration overhead imposed by the initial data can be potentially avoided by replicating it across all the nodes. The memory burden of migrating data is not considered in our model, because data movement can be handled by storage and network interface controllers without local CPU's intervention and I/O buffer [13].

For simplicity, we assume that all I/O operations issued by tasks are blocking. This simplification is conservative in the sense that it underestimates the performance benefits from the proposed scheme because this assumption causes a number of undesired migrations with negative impact.

3 Load Balancing in Heterogeneous Clusters

3.1 Existing Load Balancing Policies

In principle, the load of a node can be balanced by migrating either a newly arrived job or a currently running job preemptively to another node if needed.

While the first approach is referred to as Remote Execution, the second one is called preemptive migration [1][5]. Since the focuses of this study are effective usage of I/O resources and coping with system heterogeneity, we only consider the technique of remote execution in this paper. In what follows, we briefly review two existing load-balancing policies.

(1) CPU-based Load Balancing (CPU-RE) [3][14]. We consider a simple and effective policy, which is based on a heuristic. The CPU load of node i, $load_{CPU}(i)$, is measured by the following expression [3]: $load_{CPU}(i) = L_i \times (\frac{max_{j=1}^{n} C_j}{C_i})$ where L_i is the number of tasks on node i.

If $load_{CPU}(i)$ is greater than a certain threshold, node i is consider overloaded with respect to CPU resource. The CPU-based policy transfers the newly arrived tasks on the overloaded node i to the remote node with the lightest CPU load. This policy is capable of resulting in a balanced CPU load distribution for systems with uneven CPU load distribution [14]. Note that the CPU threshold is a key parameter that depends on both workload and transfer cost. In the experiments reported in Section 4, the value of CPU threshold is set to four [3].

(2) CPU-memory-based Load Balancing (CM-RE) [3]. This policy takes both CPU and memory resources into account. The memory load of node i, $load_{mem}(i)$, is the sum of the memory space allocated to the tasks running on node i. Thus, we have $load_{mem}(i) = \sum_{j \in N_i} mem(j)$ where $mem(j)$ represents the memory requirement of task j, and N_i denotes the set of tasks running on node i. If the memory space of a node is greater than or equal to $load_{mem}(i)$, CM-RE adopts the above CPU-RE policy to make load-balancing decisions. When $load_{mem}(i)$ exceeds the amount of available memory space, the CM-RE policy transfers the newly arrived tasks on the overloaded node to the remote nodes with the lightest memory load. Zhang et al. [3] showed that CM-RE is superiors to CPU-RE under a memory-intensive workload.

3.2 IO-aware Load Balancing in Heterogeneous Clusters

We now turn our attention to an I/O-aware load balancing policy (IO-RE), which is heuristic and greedy in nature. Instead of using CPU and memory load indices, the IO-RE policy relies on an I/O load index to measure two types of I/O access: the implicit I/O load induced by page faults and the explicit I/O requests resulting from tasks accessing disks. Let $page(i, j)$ be the implicit I/O load of task j on node i, and $IO(j)$ be the explicit I/O requirement of task j. Thus, node i's I/O load index is:

$$load_{IO}(i) = \sum_{j \in N_i} page(i, j) + \sum_{j \in N_i} IO(j) \tag{3}$$

An I/O threshold, $threshold_{IO}(i)$, is introduced to identify whether node i's I/O resource is overloaded. Node i's I/O resource is considered overloaded if $load_{IO}(i)$ is higher than $threshold_{IO}(i)$. Specifically, $threshold_{IO}(i)$, which reflects the I/O processing capability of node i, is expressed as:

$$threshold_{IO}(i) = \frac{D_i}{\sum_{j=1}^{n} D_j} \times \sum_{j=1}^{n} load_{IO}(j) \tag{4}$$

where the first term on the right hand side of the above equation corresponds to the fraction of the total I/O processing power on node i, and the second term gives the accumulative I/O load imposed by the running tasks in the heterogeneous cluster. The I/O threshold associated with node i can be calculated using equations 3 to substitute for $load_{IO}(j)$ in equation 4.

For a task j arriving at a local node i, the IO-RE scheme attempts to balance I/O resources in the following four main steps. First, the I/O load of node i is updated by adding task j's explicit and implicit I/O load. Second, the I/O threshold of node i is computed based on Equation 4. Third, if node i's I/O resource is underloaded, task j will be executed locally on node i. When the node is overloaded with respect to I/O resource, IO-RE judiciously chooses a remote node k as task j's destination node, subject to the following two conditions: (1) The I/O resource is not overloaded. (2) The I/O load discrepancy between node i and k is greater than the I/O load induced by task j, to avoid useless migrations. If such a remote node is not available, task j has to be executed locally on node i. Otherwise and finally, task j is transferred to the remote node k, and the I/O load of nodes i and k is updated in accordance with j's load.

Since tasks' implicit and explicit I/O load will be used in Equation 4 and the above conditions, it is essential to derive the two types of I/O load. When the available memory space M_i is unable to fulfill the accumulative memory requirements of all tasks running on the node ($load_{mem}(i) > M_i$), the node may encounter a large number of page faults. Implicit I/O load depends on three factors: M_i, $load_{mem}(i)$, and the page fault rate pr_i. Thus, $page(i, j)$ can be defined as follows:

$$page(i, j) = \begin{cases} 0 & \text{if } load_{mem}(i) \leq M_i, \\ \frac{pr_i \times load_{mem}(i)}{M_i} & \text{otherwise.} \end{cases} \tag{5}$$

Explicit I/O load $IO(i, j)$ is proportional to I/O access rate $ar(j)$ and inversely proportional to I/O buffer hit rate $hr(i, j)$. The buffer hit rate for task j on node i is approximated by the following formula:

$$hr(i, j) = \begin{cases} \frac{r}{r+1} & \text{if } buf(i, j) \geq d(j), \\ \frac{r \times buf(i,j)}{(r+1) \times d(j)} & \text{otherwise,} \end{cases} \tag{6}$$

where r is the data re-access rate (defined to be the number of times the same data is accessed by a task), $buf(i, j)$ denotes the buffer size allocated to task j, and $d(j)$ is the amount of data accessed by task j, given a buffer with infinite size. The buffer size a task can obtain at run time heavily depends on the total available buffer size in the node and the tasks' access patterns. $d(j)$ is linearly proportional to access rate, computation time and average data size of

I/O accesses, and $d(j)$ is inversely proportional to r. In some cases, where the initial data of a task j is not initially available at the remote node, the data migration overhead can be approximately estimated as $T_d(j) = d_{init}(j)/b_{net}$, where $d_{init}(j)$ and b_{net} represent the initial data size and the available network bandwidth, respectively.

3.3 IOCM-RE: A Comprehensive Load Balancing Policy

Since the main target of the IO-RE policy is exclusively I/O-intensive workload, IO-RE is unable to maintain a high performance when the workload tends to be CPU- or memory-intensive. To overcome this limitation of IO-RE, a new approach, referred to as IOCM-RE, attempts to achieve the effective usage of CPU and memory in addition to I/O resources in heterogeneous clusters.

More specifically, when the explicit I/O load of a node is greater than zero, the I/O-based policy will be leveraged by IOCM-RE as an efficient means to make load-balancing decisions. When the node exhibits no explicit I/O load, either the memory-based or the CPU-based policy will be utilized to balance the system load. In other words, if the node has implicit I/O load due to page faults, load-balancing decisions are made by the memory-based policy. On the other hand, the CPU-based policy is used when the node is able to fulfill the accumulative memory requirements of all tasks running on it. A pseudo code of the IOCM-RE scheme is presented in Figure 1 below.

Algorithm: IO-CPU-Memory based load balancing (IOCM-RE):
/* Assume that a task j newly arrives at node i */
if $IO(j) + \sum_{j \in N_i} IO(k) > 0$ then
  The IO-RE policy is used to balance the system node; /* see Section 3.2 */
else if $page(i,j) + \sum_{j \in N_i} page(i,k) > 0$ then /* see Section 3.1(2) */
    The memory-based policy is utilized for load balancing;
  else   /* see Section 3.1(1) */
    The CPU-based policy makes the load balancing decision;

Fig. 1. Pseudocode of the IO-CPU-Memory based load balancing

4 Performance Evaluation

In this section, we experimentally compare the performance of IOCM-RE against that of three other schemes: CPU-RE [14][15], CM-RE [3], and IO-RE (Section 3.2). The performance metric by which we judge system performance is the mean slowdown. Formally, the mean slowdown of all the tasks in trace T is given below, where w_i and $l_C(i)$ are wall-clock execution time and computation load of task

i. The implicit and explicit disk I/O load of task i are denoted as $l_{page}(i)$ and $l_{IO}(i)$, respectively.

$$slowdown(T) = \frac{\sum_{i \in T} w_i}{\sum_{i \in T}\left(\frac{n l_c(i)}{\sum_{j=1}^{n} C_j} + \frac{n(l_{page}(i)+l_{IO}(i))}{\sum_{j=1}^{n} D_j}\right)} \tag{7}$$

Note that the numerator is the summation of all the tasks' wall-clock execution time while sharing resources, and the denominator is the summation of all tasks' time spent running on CPU or accessing I/O without sharing resources with other tasks. Since the clusters studied in the simulation experiments are heterogeneous in nature, we use the average values of CPU power, memory space, and the disk performance to calculate the execution time of each task exclusively running on a node.

4.1 Simulator and Simulation Parameters

Harchol-Balter and Downey [1] have implemented a simulator of a distributed system with six nodes. Zhang et. al [3] extended the simulator by incorporating memory recourses. Compared to these two simulators, ours possesses four new features. First, the IOCM-RE and IO-RE schemes are implemented. Second, a fully connected network is simulated. Third, a simple disk model is added into the simulator. Last, an I/O buffer is implemented on top of the disk model in each node. In all experiments, we used the simulated system with the configuration parameters listed in Table 1. The parameters are chosen in such a way that they resemble workstations such as the Sun SPARC-20.

Parameters	**Value**	**Parameters**	**Value**
CPU Speed	100-400 MIPS	Page Fault Service Time	8.1 ms
RAM Size	32-256 MByte	Mean page fault rate	0.01No./MI
Buffer Size	64MByte	Data re-access rate,r	5
Context switch time	0.1 ms	Network Bandwidth	100Mbps

Table 1. System Parameters. CPU speed and page fault rate are measured by Millions Instruction Per Second (MIPS) and No./Million Instructions (No./MI), respectively.

Disk I/O operations issued by each task are modeled as a Poisson Process with a mean arrival rate λ, which is set to 2.0 No./MI in our experiments. The service time of each I/O access is the summation of seek time, rotation time, and transfer time. The transfer time is equal to the size of accessed data divided by the disk bandwidth. Data sizes are randomly generated based on a Gamma distribution with the mean size of 256KByte.

The configuration of disks used in our simulated environment is based on the assumption of device aging and performance-fault injection. Specifically, IBM 9LZX is chosen as a base disk and its performance is aged over years to generate a variety of disk characteristics [12], which is shown in Table 2.

Age	sk time	R time	Bandwidth	Age	sk time	R time	Bandwidth
1 year	5.3 ms	3.00ms	20.0MB/s	4 year	7.27ms	4.11ms	7.29MB/s
2 year	5.89ms	3.33ms	14.3MB/s	5 year	8.08ms	4.56ms	5.21MB/s
3 year	6.54ms	3.69ms	10.2MB/s	6 year	8.98ms	5.07ms	3.72MB/s

Table 2. Characteristics of Disk Systems. sk time: seek time, R time: Rotation time

Table 3. Characteristics of Five Heterogeneous Clusters. CPU and memory are measured by MIPS and MByte. Disks are characterized by bandwidth measured in MByte/S. HL-Heterogeneity Level

Node	system A			system B			system C			system D			system E		
	cpu	*mem*	*disk*	*cpu*	*mem*	*disk*	*cpu*	*mem*	*disk*	*cpu*	*mem*	*disk*	*cpu*	*mem*	*disk*
1	100	480	20	100	480	20	100	480	10.2	50	320	10.2	50	320	5.21
2	100	480	20	150	640	20	150	640	20	200	800	20	200	800	14.3
3	100	480	20	150	640	20	150	640	20	200	800	20	200	800	20
4	100	480	20	50	320	20	50	320	10.2	50	320	14.3	50	320	5.21
5	100	480	20	100	480	20	100	480	20	50	320	14.3	50	320	7.29
6	100	480	20	50	320	20	50	320	10.2	50	320	10.2	50	320	3.72
HL	0	0	0	0.27	0.20	0	0.27	0.20	0.25	0.35	0.28	0.20	0.35	0.28	0.30

4.2 Overall Performance Comparisons

In this experiment, the CPU power and the memory size of each node are set to 200MIPS and 256MByte. Figure 2 plots the mean slowdown as a function of disk age. Disks are configured such that five fast disks are one year old, and a sixth, slower disk assumed an age ranging from 1 to 6 years.

Figure 2 shows that the mean slowdowns of four policies increase considerably as one of the disks ages. This is because aging one slow disk gives rise to longer I/O processing time. A second observation from Figure 2 is that IO-RE and IOCM-RE perform the best out of the four policies, and they improve the performance over the other two policies by up to a factor of 8. The performance improvement of IO-RE and IOCM-RE relies on the technique that balances I/O load by migrating I/O-intensive tasks from overloaded nodes to underloaded ones.

4.3 Impact of Heterogeneity on the Performance of Load-balancing Policies

In this section, we turn our attention to the impact of system heterogeneity on the performance of the proposed policies. The five configurations of increasing heterogeneity of a heterogeneous cluster with 6 nodes are summarized in Table 3. As can be seen from Figure 3, IO-RE and IOCM-RE significantly outperform the other two policies. For example, IOCM-RE improves the performance over CPU-RE and CM-RE by up to a factor of 5 and 3, respectively.

Importantly, Figure 3 shows that the mean slowdowns of almost all policies increase consistently as the system heterogeneity increases. An interesting obser-

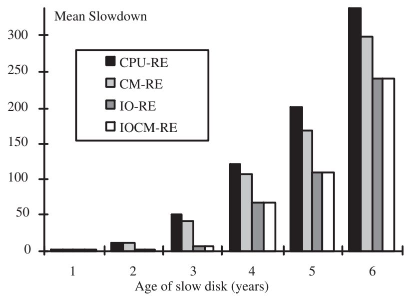

Fig. 2. Mean slowdown as a function of the age of a single disk

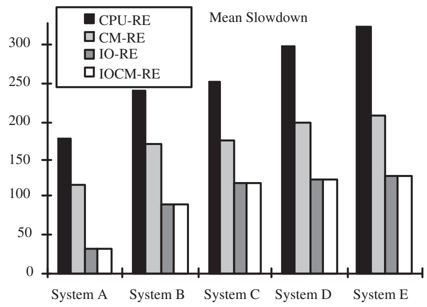

Fig. 3. Mean slowdown on five heterogeneous systems.

vation from this experiment is that the mean slowdowns of IO-RE and IOCM-RE are more sensitive to changes in CPU and memory heterogeneity than the other two policies. Recall that system B's CPU and memory heterogeneities are higher than those of system A. Compared the performance of system A with that of B, the mean slowdowns of IO-RE and IOCM-RE are increased by 196.4%, whereas the slowdowns of CPU-RE and CM-RE are increased approximately by 34.7% and 47.9%, respectively. The reason is that I/O-aware policies ignore the heterogeneity in CPU resources. When the heterogeneity of CPU and memory remain unchanged, IO-RE and IOCM-RE is less sensitive to the change in disk I/O heterogeneity than the other three policies. This is because both IO-RE and IOCM-RE consider disk heterogeneity as well as the effective usage of I/O resources.

5 Conclusion

In this paper, we have studied two I/O-aware load-balancing policies, IO-RE (I/O-based policy) and IOCM-RE (load balancing for I/O, CPU, and Memory), for heterogenous clusters executing applications that represent a diverse workload conditions, including I/O-intensive and memory-intensive applications. IOCM-RE considers both explicit and implicit I/O load, in addition to CPU and memory utilizations. To evaluate the effectiveness of our approaches, we have compared the performance of the proposed policies against two existing approaches: CPU-based policy (CPU-RE) and CPU-Memory-based policy (CM-RE). IOCM-RE is more general than the existing approaches in the sense that it can maintain high performance under diverse workload conditions. A trace-driven simulation provides us with empirical results to draw three conclusions: (1) When a workload is I/O-intensive, the proposed scheme improves the performance with respect to mean slowdown over the existing schemes by up to a factor of 8. (2) The slowdowns of the four policies considerably increase as one of the disks ages. (3) The slowdowns of almost all the policies increase consistently with the system heterogeneity.

In this paper, we assumed that parallel applications can be broken into a number of individual tasks and, therefore, future research will deal with parallel jobs with inter-dependent tasks. We also plan to evaluate the performance of the proposed schemes using a set of real I/O-intensive application traces.

6 Acknowledgments

This work was partially supported by an NSF grant (EPS-0091900), a Nebraska University Foundation grant (26-0511-0019), and a UNL Academic Program Priorities Grant. Work was completed using the Research Computing Facility at University of Nebraska-Lincoln.

References

1. Harchol-Balter, M., Downey, A.: Exploiting process lifetime distributions for load balancing. ACM Transactions on Computer Systems **15** (1997) 253–285
2. Acharva, A., Setia, S.: Availability and utility of idle memory in workstation clusters. In: Proceedings of the ACM SIGMETRICS Conf. on Measuring and Modeling of Computer Systems. (1999)
3. Xiao, L., Zhang, X., Qu, Y.: Effective load sharing on heterogeneous networks of workstations. In: Proc. of International Symposium on Parallel and Distributed Processing. (2000)
4. Qin, X., Jiang, H., Zhu, Y., Swanson, D.: A dynamic load balancing scheme for I/O-intensive applications in distributed systems. In: Proceedings of the 32nd International Conference on Parallel Processing Workshops. (2003)
5. Qin, X., Jiang, H., Zhu, Y., Swanson, D.: Boosting performance for I/O-intensive workload by preemptive job migrations in a cluster system. In: Proc. of the 15th Symp. on Computer Architecture and High Performance Computing, Brazil (2003)
6. Scheuermann, P., Weikum, G., Zabback, P.: Data partitioning and load balancing in parallel disk systems. The VLDB Journal (1998) 48–66
7. Cho, Y., Winslett, M., S. Kuo, J.L., Chen, Y.: Parallel I/O for scientific applications on heterogeneous clusters: A resource-utilization approach. In: Proceedings of Supercomputing. (1999)
8. Zhu, Y., Jiang, H., Qin, X., Feng, D., Swanson, D.: Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS). In: the Fourth LCI International Conference on Linux Clusters. (2003)
9. Zhu, Y., Jiang, H., Qin, X., Feng, D., Swanson, D.: Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (ceft-pvfs). In: Proc. of the 3rd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid. (2003)
10. Ma, X., Winslett, M., Lee, J., Yu, S.: Faster collective output through active buffering. In: Proceedings of the International Symposium on Parallel and Distributed Processing. (2002)
11. Qin, X., Jiang, H., Zhu, Y., Swanson, D.: Dynamic load balancing for I/O- and memory-intensive workload in clusters using a feedback control mechanism. In: Proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003), Klagenfurt, Austria (2003)

12. Forney, B., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Storage-aware caching: Revisiting caching for heterogeneous storage systems. In: Proceedings of the 1st Symposium on File and Storage Technology, Monterey, California, USA (2002)
13. Geoffray, P.: Opiom: Off-processor I/O with myrinet. Future Generation Computer Systems **18** (2002) 491–499
14. Franklin, M., Govindan, V.: A general matrix iterative model for dynamic load balancing. Parallel Computing **33** (1996)
15. Eager, D., Lazowska, E., Zahorjan, J.: Adaptive load sharing in homogeneous distributed systems. IEEE Trans. on Software Eng. **12** (1986) 662–675