Parallel I/O Scheduling in Multiprogrammed Cluster Computing Systems

J.H. Abawajy

School of Computer Science,
Carleton University, Ottawa, Canada.
abawjem@scs.carleton.ca

Abstract. In this paper, we address the problem of effective management of parallel I/O in multiprogrammed cluster computing systems by using appropriate I/O scheduling strategies. The performance of I/O devices lags behind the performance of processors and network systems, resulting in I/O becoming the bottleneck in current systems. This gap is expected to increase in the future since I/O performance is limited by physical motion. Therefore, it is imperative that novel techniques for improving I/O performance be developed. Parallel I/O is a promising approach to alleviating this bottleneck. However, very little work exists with respect to scheduling parallel I/O operations explicitly. We propose new I/O scheduling algorithms and evaluate the relative performance of the proposed policies against the most commonly used approaches. Our preliminary results show that the proposed policies outperform current methods and can substantially enhance the performance of real-world scientific applications.

1 Introduction

In this paper, we focus on I/O-intensive large scale parallel applications. In addition to multiple processors, the parallelization of I/O operations and the use of multiple disk drives are required for achieving high system performance for the workloads of interest. The aggregate processing power and the storage capacity provided by cluster computing systems have lead to the widespread adoption of such systems as high performance computing platforms for various types of large-scal applications including I/O-bound applications such as digital libraries and image repositories.

Although processor and network technologies have seen impressive advances in performance, the lack of commensurate improvements in I/O devices has resulted in I/O becoming a major bottleneck in these systems. Due to physical limitations of I/O devices, significant improvements in access latency relative to processor and network speeds are unlikely. It is, therefore, imperative to find techniques that can improve the access performance of storage devices despite their high latency.

Recently, the I/O bottleneck in cluster computing systems has begun receiving increasing interest. In particular, parallel I/O has drawn increasing attention

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 223–229, 2003.

in the last few years as a promising approach to alleviating I/O bottleneck in cluster computing systems. Parallel I/O combines a set of storage devices and provides interfaces to utilize them in concert. However, it has been shown that peak performance is rarely attained from these coordinated storage devices [1]. As a result, a number of approaches for more efficiently utilizing these resources have been continuously researched and developed [3,5,10,11].

Widely applicable solutions, however, will require an integrated approach which addresses the problem at multiple system levels, including applications, systems software, and architecture [14]. We propose that within the context of such an integrated approach, scheduling parallel I/O operations will become increasingly attractive and can potentially provide substantial performance benefits. Parallel I/O scheduling is concerned with scheduling of parallel I/O operations with the goal of minimizing the overall I/O response times.

As noted in [2], comparatively a lesser amount of work exists on parallel I/O scheduling. To this end, we present a set of I/O scheduling algorithms for multiprogrammed cluster computing environments running parallel I/O workloads. We present simulation results indicating that the proposed algorithms can produce a substantial improvement over previous algorithms.

The rest of the paper is organized as follows. Section 2 discusses previous work related to scheduling of parallel I/O operations. Section 3 presents the system and workload models used in this paper. Section 4 describe the system environment used in this study along a set of I/O scheduling algorithms we have implemented. Section 5 describe the testing environment along the system and workload parameters. The results of the experiments are also discussed. The conclusions and future directions are given in Section 6.

2 Related Work

By combining large numbers of storage devices and providing the system software to utilize them in concert, parallel I/O has extended the range of problems that may be solved on high performance computing platforms. However, the conglomeration of storage systems and providing convenient access alone is not enough to achieve maximum possible performance. In [8], for instance, it was found that when two processes concurrently read or write, the throughput per process drops to far less than half of the throughput obtained when only one process is making I/O requests. Thus, blindly allowing multiple processors to access files on the same disk partition simultaneously can be detrimental to performance.

In the last few years, several techniques to improve parallel I/O performance have been proposed in the literature. These techniques span from tuning the parallel applications [13] to scheduling the I/Os. One dimension that has been pursued in attacking the I/O bottleneck problem is the development of parallel and distributed file systems (e.g., Parallel Virtual File System (PVFS) [5]) and I/O runtime libraries (e.g., [6,10]) that distribute the datasets over several nodes and providing low-level data access mechanisms. Also a number of strategies

such as prefetchin and caching as well as fairly low-level techniques such as disk striping and interleaving have been proposed in the literature.

Another class of research is those that are focusing on the *data access* strategies such as collective I/O [3]. Although these *data access* strategies perform well for applications with specific I/O characteristics, they perform poorly for applications that do not confirm with the underlying assumption on which these strategies were built on.

Although the development of parallel/distributed file systems and access strategies have helped to ease the performance gap, but I/O still remains an area requiring significant performance improvement [1]. Also, currently many of the parallel filesystems and I/O runtime libraries address maximizing the performance of individual parallel applications having intense I/O requirements [14].

In multiprogrammed environments, where a number of applications are competing for resources, an important dimension and the focus of this paper is the effective management of parallel I/O by using appropriate I/O scheduling strategies. Research in I/O scheduling strategies for parallel workload by far is the least studied area [2]. In [14], scheduling a batch of I/O operations in parallel computer systems is discussed. An approach for scheduling parallel I/O that focus on uniprogrammed systems that run a single job at a time in isolation is discussed in [2]. In some cases, these techniques have also been extended to multiprogrammed environments that execute multiple parallel jobs simultaneously under the assumption that information such as the application execution time is known *a priori* [9,2]. Unfortunately, it is very difficult if not impossible to know the pending I/O service demands of a job rendering the policies proposed in [9,2] of theoretical importance than practical one. Also, most studies have not investigated the potential performance problems of handling large outstanding I/O requests in a multi-workload environments.

3 System Model

The *cluster computing* of interest has **P** workstations of similar architecture, connected by a fast interconnection network. As in [5], the **P** workstations are divided into a set of compute node and a set of I/O servers. Data is declustered among the I/O nodes to allow parallel access to different files and parallel access to the same file. This approach increases the performance and scalability of the system. An alternative approach is instead of dedicating extra processors for I/O, some compute processors can be part-time I/O servers as suggested in [7]. However, this strategy requires solving a number of difficult problems such as selection and placement of I/O servers [7] and its merit will be considered in the future.

We assume that parallel jobs arrive to the system in a stochastic manner with requirement for both compute and I/O resources. Each parallel job is composed of several tasks that can be scheduled to executed on a single workstation. In this paper, we focus on parallel I/O scheduling and we do not address the problem of

parallel job scheduling as this problem has been addressed in [15]. Data required to run a task is fetched before the task is run if it is not already present in the workstation on which the task is assigned to run.

All I/O requests from a job is sent to the data scheduler in the I/O subsystem, which coordinates the I/O request on multiple I/O nodes for greater efficiency. Typically each tasks in a parallel application will send a request to each I/O server when performing an I/O operation. It is easy to imagine that for a large parallel application a large number of I/O requests might be in service on an I/O Scheduler at one time. This large number of I/O requests provide us with an opportunity to optimize by selecting the order in which tasks will be serviced. Therefore, the order with which the I/O device services the I/O requests can affect the overall performance of the system. Upon arrival, if an I/O request cannot be serviced immediately, it is placed into an I/O wait queue and scheduled using I/O scheduling approaches discussed next.

4 Parallel I/O Scheduling Approaches

In a multiprogrammed environments, performance can be drastically reduced when multiple processors make requests concurrently due to contention for shared resources. Concurrent access to shared resources like disks or network interface causes significant overhead and can slow down applications. That also induces a significant performance gap in both I/O and communication, depending on the number of processors concurrently accessing the resources. In this section, we discuss briefly the classical I/O scheduling policy and propose two new policies.

First come First Served (FCFS) Policy: The FCFS I/O scheduling strategy is the most commonly used approach. In FCFS scheme, the scheduler maintains a single I/O wait queue, where all pending I/O requests are kept until they are serviced. When an I/O server becomes available, the scheduler schedules a pending request from the wait queue using the FCFS policy.

I/O Bunch Policy: This is a two-level scheduling policy where at the top level the scheduler creates a set of *I/O Bunches* from the I/O requests arriving to the data scheduler while the bottom layer maps each *I/O Bunch* onto the data servers. The scheduler creates n *I/O Bunches*, $B_1, ..., B_n$, each B_i is characterized as follows:

1. composed of I/O requests for a given file;
2. Inherits the arrival time of the oldest I/O request in the bunch; and
3. At most there are f I/O requests in the bunch, where f is a tunable parameter that can be controlled either at compile-time or dynamically at run-time.

At each scheduling round, the scheduler assigns a B_i to the I/O servers based on the order of B_i arrival time.

Job-based Policy: This policy is also a two-level scheduling policy and uses the concept bunching. However, the policy harnesses knowledge of I/O characteristics of the jobs in scheduling I/O requests, which include the number of CPU-I/O phases, the type and size of the requests. Note that unlike the policy proposed in [9], we explore information that can be easily extracted from the job either at the time of submission or at the compile-time. A simplified policy job-based policy we studied considers the size of the I/O requests. Each bunch, B_i, has a set of I/O requests of size less than or equal to **S**. Each bunch is then assigned to the I/O server where the request are served.

5 Performance Analysis

We compared the performance of the three I/O scheduling policies discussed in the previous section using I/O completion times as a performance metric. The system is composed of 14 workstations where two of them are used as an I/O servers. The file sizes range from 1 MB to 16 MB. Files were written and read by making from 1 to 128 request per file. A delay was placed between each request ranging from zero to one second. This delay represents the time for which a real application would perform computation or wait for some reason other than I/O. Table 1 summarizes the default parameters and values used in the experiments.

Table 1. Default Parameters and values.

Parameters	Values
Number of CPUS	14
I/O Types	CW, W, R
File sizes	1MB to 16MB
Requests/file	1 to 128
Requests size	1 to 128
Delays	0ms, 100ms, 500ms, 1000ms

The preliminary results of the experiments show that the proposed policies perform between 13% to 22% better than the FCFS policy under all system and workload parameters we tested. The job-based policy outperforms the other two policies.

This can be explained by the fact that FCFS does not consider disk access order at all, and may result in significant disk head movement when servicing multiple requests.

Our results suggests that the naive approach to scheduling I/O results in a large number of I/O operations, each of which is often for a very small amount of data. The added network cost of performing an I/O operation across the network is often high due to latency. We also observed that on single disk I/O

servers, as I/O queues are filled with large requests spanning many files, there is the potential for high seek penalties that can lead to variability in request time, lower I/O performance and throughput. Our results also suggests that using knowledge of the application I/O patterns, it is possible to achieve much more efficient I/O than more general solution.

These results are based on limited performance analysis and should be considered as a preliminary. We are undertaking rigorous testing to verify the results.

6 Conclusions and Future Directions

Parallel I/O is a promising approach to alleviating the problem of I/O bottleneck in cluster computing systems. We have been investigating an effective management of parallel I/O by using appropriate I/O scheduling strategies. Although parallel I/O scheduling can improve the performance, very little work exist on scheduling parallel I/O operations explicitly. We proposed new efficient I/O scheduling algorithms and evaluated the relative performance of the policies against the most commonly used I/O scheduling approach. These new I/O scheduling approaches have been shown, through simulation, to have superior performance to existing schemes and can substantially enhance the performance of real-world scientific applications. We are currently investigating their performance through experimentation with I/O traces taken from the scientific and non-scientific application domains, including web-servers and interactive applications. We are also looking at implementing and testing the proposed scheduling policies on .

References

1. R. Ross and W. Ligon, "Server-Side Scheduling in Cluster Parallel I/O Systems," *Special Issue of Calculateurs Paralleles,* 2003.
2. Fangyu Chen, Shikharesh Majumdar, "Performance of Parallel I/O Scheduling Strategies on a Network of Workstations," *ICPADS 2001,* pages: 157–164, 2001.
3. Phillip M. Dickens and Rajeev Thakur, "Evaluation of Collective I/O Implementations on Parallel Architectures," *Journal of Parallel and Distributed Computing,* **61**(8),(2001):1052–1076.
4. R. Ross, D. Nurmi, A. Cheng, and M. Zingale, "A Case Study in Application I/O on Linux Clusters," *Proceedings of SC2001, Denver, CO, November,*(2001).
5. P. Carns, W. Ligon III, R. Ross, and R. Thakur, "PVFS: A Parallel File System For Linux Clusters," *Proceedings of the 4th Annual Linux Showcase and Conference,* (Atlanta, GA, October 2000,) pp. 317–327.
6. Rajeev Thakur, William Gropp, and Ewing Lusk, "On Implementing MPI-IO Portably and with High Performance," *Proc. of the Sixth Workshop on I/O in Parallel and Distributed Systems,*(May 1999), pp. 23–32.
7. Y. Cho, M. Winslett, S. Kuo, Y. Chen, J. Lee, and K. Motukuri, "Parallel I/O on Networks of Workstations: Performance Improvement by Careful Placement of I/O Servers," *In Proceedings of the HiPer'98, High Performance Computing on Hewlett-Packard Systems (Annual conference of the HP2EUG),Switzerland, October 1998,* pages 104–111, Zurich.

8. Y. Cho, M. Winslett, S. Kuo, J. Lee, and Y. Chen, "Parallel I/O for Scientific Applications on Heterogeneous Clusters: A Resource-utilization Approach," *Proceedings of the 13th ACM International Conference on Supercomputing,Rhodes, Greece,* June 1999.
9. P. Kwong, S. Majumdar, "Scheduling of I/O in Multiprogrammed Parallel Systems," *Informatica,* **23**(1),(April 1999):67–76.
10. Kent E. Seamons. "Panda: Fast Access to Persistent Arrays Using High Level Interfaces and Server Directed Input/Output," *PhD thesis,* Dept. of Computer Science, University of Illinois at Urbana-Champaign, May 1996.
11. R. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. Culler, J. Hellerstein, D. Patterson, and K. Yelick. "Cluster I/O with River: Making the Fast Case Common," *In Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems, Atlanta, Georgia*, 1999.
12. Sandra Johnson Baylor, Caroline B. Benveniste, and Y. Hsu. "Performance evaluation of a parallel I/O architecture," *Technical Report RC 20049, IBM T. J. Watson Research Center,* May 1995.
13. A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, and A. Sussman. "Tuning the Performance of I/O-Intensive Parallel Applications," *In Proceedings of the Fourth Annual Workshop on I/O in Parallel and Distributed Systems,* pages 15–27, May 1996.
14. Ravi Jain, Kiran Somalwar, John Werth, J.c. Browne, "Heuristics for Scheduling I/O Operations," *IEEE Transactions on Parallel and Distributed Systems,* March 1997 (Vol. 8, No. 3), pp. 310–320.
15. J. H. Abawajy, "An integrated resource scheduling approach on cluster computing systems," *In Proceedings of the PDSECA/IPDPS-03), Nice, France,* April 22–26, 2003.