Performance Analysis of Parallel I/O Scheduling Approaches on Cluster Computing Systems

J. H. Abawajy
School of Computer Science,
Carleton University,
Ottawa, K1S 5B6, Canada.
abawjem@scs.carleton.ca

Abstract

As computation and communication hardware performance continue to rapidly increase, I/O represents a growing fraction of application execution time. This gap between the I/O subsystem and others is expected to increase in 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 exist with respect to scheduling parallel I/O operations explicitly. In this paper, we address the problem of effective management of parallel I/O in cluster computing systems by using appropriate I/O scheduling strategies. We propose two new I/O scheduling algorithms and compare them with two existing scheduling Approaches. The preliminary results show that the proposed policies outperform existing policies substantially.

1 Introduction

As cluster computing gains popularity, it is increasingly being used for parallel and sequential applications with significant I/O requirements. However, with the tremendous advances in processor and interconnection network technologies, the lack of commensurate improvements in I/O subsystems have resulted in I/O becoming a major bottleneck for many applications in these systems. In the last few years, parallel I/O has drawn increasing attention as a promising approach to alleviating this bottleneck. Parallel I/O combines a set of storage devices and provides interfaces to utilize them in concert. For example, several file systems (e.g., Parallel Virtual File System (PVFS) [8]) and I/O runtime libraries have been developed to alleviate the I/O bottlenecks by distributing the data over several nodes and providing low-level data access mechanisms.

Although the development of parallel file systems has helped to ease the performance gap, but I/O still remains an area requiring significant performance improvement [11, 10]. This has lead to a number of different approaches. Most attention has focused on improving the performance of I/O devices using fairly low-level parallelism in techniques such as disk striping and interleaving [11]. Data distribution strategies and data layout strategies have also been researched for optimizing I/O performance [3, 1]. Also, research focusing on the *data access* strategies such as *collective I/O* [9] have been proposed as a means of optimizing I/O performance.

This paper investigates the effectiveness of parallel I/O scheduling strategies for multiprogrammed cluster computing environments. Most existing studies have not investigated the potential performance problems of handling tens to possibly hundreds of outstanding I/O requests in a multi-

0-7695-1919-9/03 $17.00 © 2003 IEEE

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

workload environment. We believe that such environments require an efficient parallel I/O scheduling techniques in order to address the I/O bottleneck problem. Parallel I/O scheduling is concerned with allocating parallel I/O operations to the most appropriate I/O server with the goal of minimizing the overall response times. Although parallel I/O scheduling can potentially provide substantial performance benefits, by far I/O scheduling strategies for parallel workload is the least studied area. A number of I/O scheduling strategies that are based on various knowledge of I/O patterns such as the sum of the service demands of is described in [4]. Unfortunately, it is very difficult if not impossible to know the pending I/O service demands of a job. Some of the I/O patterns of a job do not lend themselves to be easily explored by the parallel I/O scheduling subsystem. Scheduling of a batch of I/O operation is also discussed in [11]. To this end, we propose two new I/O scheduling algorithms and compare them with two existing scheduling Approaches. The preliminary results show that the proposed policies outperform existing policies substantially.

The rest of the paper is organized as follows. Section 2 describes the system model used in this study. In Section 3, a set of scheduling algorithms we have implemented for the purpose of this study are discussed. Section 4 presents the system and workload parameters used in the experiments whereas the discussion of results is presented in Section 5. The conclusions and future directions are given in Section 6.

2 System Model

The system of interest has **D** disks and **P** workstations that are connected by a fast interconnection network as shown in Figure 1. As in [11], we replicate application data on at least $K < D$ disks. Any data required to run a job is fetched (if it is not already present locally) before the job is run . All I/O request from a job is sent to a *data scheduler* that runs on a single central node

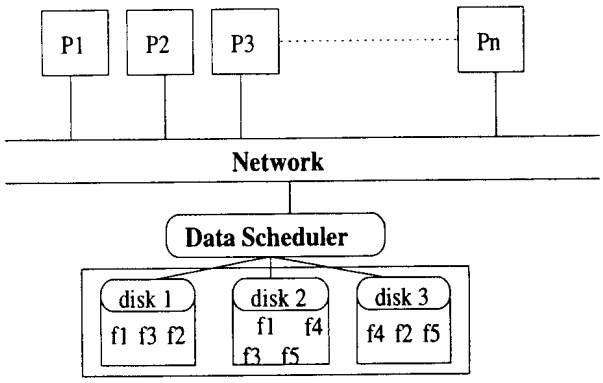

Figure 1: Cluster Computing system architecture

in the I/O subsystem. The *data scheduler* coordinates the I/O request on multiple I/O nodes for greater efficiency. 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 in the following subsections. In the following sections, we describe a number of parallel I/O scheduling strategies in detail.

3 Parallel I/O Scheduling Strategies

When application data is replicated on K disks, each I/O request can be possibly serviced by K servers. This means an I/O server may receive requests from several concurrent parallel applications, which do parallel I/O. Thus, if the I/O requests are not efficiently scheduled, some I/O servers can be overloaded while others remain idle. Moreover, the order with which the I/O servers service the I/O requests can affect the overall performance of the system. This will have a serious impact on the performance of the applications, which necessitates an efficient I/O scheduling policy. The objective of the I/O scheduler is to balance the load among the k I/O servers such that the response time of the jobs is minimized.

In this section, we propose two new schedul-

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

ing polices called the *Equi-Partition (EQUI)* I/O scheduling policy and the *Adaptive Equi-Partition (AEQUI)* I/O scheduling policy. We also describe three existing policies proposed in [11] called the *Lowest Destination Degree First (LDDF)* and *Highest Destination Degree First (HDDF)*.

3.1 Equi-Partition Policy

Figure 2 is a pseudo-code of the *Equi-Partition (EQUI)* I/O scheduling policy. In *(EQUI)* policy, at each scheduling iteration, the algorithm first determines the average number of I/O requests (i.e., *Avg*) to be handed to each I/O server as follows:

$$Avg = \frac{\text{number outstanding I/O requests}}{S} \quad (1)$$

where the parameter S is denotes number of severs that can service the outstanding I/O requests.

Let OR_k be a subset of outstanding I/O requests that can be serviced by an I/O server N_k. The algorithm selects an I/O server $N_i \in S$ with $OR_k > 0$ and the OR_k is the lowest among the I/O servers. It then removes a set of $min(Avg, OR_k)$ unassigned I/O requests from the I/O request pending queue and forwards it to server N_i. This process repeats until all pending I/O requests are assigned to the I/O servers.

Algorithm 1: Equi-Partition Policy

Let D be the number outstanding I/O requests

1. While not all pending I/O requests are assigned (i.e., $D > 0$) DO
   (a) Let N_i be an I/O server that meets the following conditions:
      i. has an outstanding requests (i.e., $OR_i > 0$);
      ii. has the lowest outstanding I/O request (i.e., min (OR_i);and
      iii. has not been assigned I/O requests in this round;
   (b) Assign I/O requests from pending I/O queue as follows:

$$target = min(OR_i, Avg)$$
$$OR_i = OR_i - target$$
$$N_i = target$$
$$D = D - target$$

Figure 2: Equi-Partition I/O Scheduling

3.2 Adaptive Equi-Partition

Figure 3 is a pseudo-code of the *Adaptive Equi-Partition (AEQU)* I/O scheduling policy. At each scheduling point, the *AEQU* first determines the average number of I/O requests (i.e., *Avg*) to be allocated to each I/O server as follows:

$$Avg = \frac{\text{Total number of I/O requests}}{S} \quad (2)$$

where

- S = total number of severs that can service the pending I/O requests;
- Total number of I/O requests = $R_{pending} + R_{backlog}$;
- $R_{pending}$ = a subset of outstanding I/O requests; and
- $R_{backlog}$ = the total number of *locally backlogged (i.e., assigned to I/O servers but not yet processed* I/O requests.

The I/O servers are then sorted in an increasing order based on the total number of locally backlogged I/O requests. The algorithm then assigns pending I/O requests to under loaded I/O servers while re-scheduling I/O requests from overloaded I/O servers onto the under loaded servers.

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

Algorithm 2: Adaptive Equi-Partition

Let:

- D = the total number I/O requests;
- $R^k_{pending}$ = a subset of outstanding I/O requests that can be serviced by server k
- $R^k_{backlog}$ = the total number of *locally backlogged* I/O requests at server k.

1. Sort the **S** servers in an increasing order based on the number of locally *bocklogged I/O requests.*
2. For each $N_k \in$ **S** DO
   (a) IF $R^k_{backlog} > Avg + 1$ THEN

$$target = R^k_{backlog} - Avg + 1$$
$$R^k_{backlog} = R^k_{backlog} - target$$
$$D = D + target$$

   (b) Otherwise, assign I/O requests from pending I/O queue as follows:

$$target = min(R^k_{pending} + R^k_{backlog}, Avg)$$
$$R_{pending} = R_{pending} - target$$
$$N_k = target$$
$$D = D - target$$

Figure 3: Adaptive Equi-Partition I/O Scheduling

3.3 Lowest Destination Degree First

An I/O scheduling policy called the *Lowest Destination Degree First (LDDF)* is proposed in [11]. In LDDF policy, I/O requests are mapped to the I/O servers in a round robin fashion. The algorithm first sorts the I/O servers in an increasing order of the pending I/O requests that they can possibly service. Starting with an I/O server that has the lowest number of pending I/O requests, the algorithm removes a pending I/O request for the I/O servers from the queue and assigns it to the server. It then picks the next I/O server with the lowest number of pending I/O requests, removes a pending I/O request from the queue and assigns it to the I/O server. This process continues until there is no more pending I/O requests.

3.4 Highest Destination Degree First

The *Highest Destination Degree First (HDDF)* [11] is similar to the LDDF policy except the algorithm sorts the I/O servers in a decreasing order of the pending I/O requests that they can possibly service. It also chooses the I/O server with the highest number of pending I/O requests first.

3.5 Random I/O Request Scheduling

In the *Random I/O Request Scheduling (RIORS)* policy, as the I/O requests arrive at the I/O scheduler, the request are assigned to a randomly selected I/O server.

4 Performance Analysis

We compared the performance of the four I/O scheduling policies discussed in the previous section using the *mean response time (MRT)* as a performance metric. All the experiments were performed on a cluster computing environment that consists of 20 pentim workstations (166 MHz) on the same 100 MBs LAN and run the Parallel Virtual Machine (PVM) under Linux operating system. In the experiments, we used 16 workstations as compute nodes while three workstations are used as I/O servers where the workstation local disk is used to store the application data. There is one I/O process on every I/O node that performs the read or write operation. Finally, we used one of the workstations as the master scheduler where the I/O requests are submitted to.

The parallel workloads of interest to us are those characterized by alternating CPU and I/O

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

phases that repeat **K** times as observed in [5]. It is a master-worker type workload (i.e., a matrix multiplication), which is characterized with non-overlapping CPU-I/O operations. However, it uses *parallel I/O technique* to read/write the application data. Specifically, the master task uses n parallel I/O operations to read in the application data from n disk. It then distributes the data among the p worker tasks. The worker tasks perform only computation using the data and at the end of the computation they return the result data to the master task, which combines the data from the n workers and writes it to n disk using n parallel I/O operations. After the data is successfully written to disk the job completes.

5 Relative Performance of the Policies

Table 1 shows the relative performance of the five I/O scheduling policies. The data shows that the proposed I/O scheduling policies are quite efficient with respect to the other three policies. The *Random* policy performs slightly better than the *Highest Destination Degree First* policy while *Lowest Destination Degree First* performs somewhat better than both *Highest Destination Degree First* and the *Random* policies.

Policy	MRT
Equi-partition	57.870
Adaptive Equi-partition	51.030
Lowest Destination Degree First	88.820
Highest Destination Degree First	95.190
Random	92.230

Table 1: Performance comparison of the five I/O scheduling policies.

The *Equi-partition* policy performs better that the *Lowest Destination Degree First*, *Highest Destination Degree First* and the *Random* policies. This is because the *Equi-partition* policy distributed the I/O requests in a balanced fashion over the I/O servers whereas the other three policies do not.

The *Adaptive Equi-partition* is the best as this policy allocates pending I/O requests as well as move unprocessed I/O requests from heavy loaded to lightly loaded I/O servers.

6 Conclusions and Future Directions

. The I/O bottleneck in cluster computing systems has recently begun receiving increasing attention. However, very little work exist with respect to scheduling parallel I/O operations in cluster computing systems. In this paper, we presented two new scheduling policies and demonstrated their effectiveness by comparing their performance with three other scheduling policies.

Acknowledgments My thanks to Meliha who has been very kind to me and helping me with many things for completing this project. Wish you good luck in Melbourne, Australia.

References

[1] Peter Kwong and Shikharesh Majumdar, "Study of Data Distribution Strategies for Parallel I/O Management", ACPC, pp. 12-23, 1996.

[2] L. Diaconescu and S. Majumdar, "Effect of Average Parallelism and CPU-I/O Overlap on the Performance of Parallel Applications", Proc. Workshop on Industrial Application of Network Computing, 2000.

[3] S. Majumdar and F. Shad, "Characterization and Management of I/O in Multiprogrammed Parallel Systems", Proc. Parallel and Distributed Processing, pp. 502-510, 1995.

[4] Peter Kwong and Shikharesh Majumdar, "Scheduling of I/O in Multiprogrammed Par-

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE

allel Systems, Informatica (Slovenia), 23 (1), pp. 104-113, 1999.

[5] Barbara K. Pasquale and George C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications", Supercomputer Conference, pp.660-669, 1994.

[6] E. Rosti et. al., "Models of Parallel Applications with Large Computation and I/O Requirements", TSE", 28 (3), pp. 286-307, 2002.

[7] E. Rosti et. al., "The Impact of I/O on Program Behavior and Parallel Scheduling", Proceedings of the SIGMETRICS Conference, pp. 56-65, 1998.

[8] R. Ross and W. Ligon, "Server-side Scheduling in Cluster Parallel I/O Systems", *Calculateurs Parallele*, 2002.

[9] Y. E. Cho et al. , "Parallel I/O for scientific applications on heterogeneous clusters: a resource-utilization approach", International Conference on Supercomputing", pp. 253-259, 1999.

[10] J. H. Abawajy, "Parallel I/O Scheduling in Multiprogrammed Cluster Computing Systems", In Proc. of the ICCS 2003, Melbourne, Australia, 2003.

[11] F. Chen and S. Majumdar, "Performance of Parallel I/O Scheduling Strategies on Networks of Workstations", *In Proc. ICPADS 2001*, pp. 157-164, 2001.

[12] R. Buyya (ed.), "High Performance Cluster Computing: Architectures and Systems", volume 1 and 2, Prentice Hall PTR, NJ, USA, 1999.

Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)
0--7695-1919-9/03 $17.00 © 2003 IEEE