This paper addresses the problem of a unified framework for resource scheduling with emerging constraints that are important in cluster computing systems. Most research in job scheduling study in multi-programmed systems has focused solely on the allocation of processors to jobs. However, since I/O is also a critical resource for many parallel jobs, the allocation of processor and I/O resources must be coordinated to allow the system to operate most effectively. To this end, we present an efficient job scheduling policy for cluster computing systems. We studied the performance of the proposed scheduling policy under various parameters. We also compared the performance of the proposed scheduling policy with a recently proposed scheduling policy. The results show that our policy performs substantially better.

Keywords: Cluster computing, Job Scheduling, Parallel I/O, Resources Scheduling, High-Throughput Computing, High-Performance Computing.

1 Introduction

Cluster computing is rapidly emerging as a viable and cost-effective high-performance computing (HPC) platform for parallel and distributed applications. As cluster computing is gaining popularity, it is increasingly being used for applications with significant I/O requirements in addition to compute-intensive applications. However, with the tremendous advances in processor and memory technology, I/O has risen to become the bottleneck in high-throughput computing as well as high performance computing for many applications.

Our focus in this paper is on scheduling algorithm suitable for a multiprogrammed cluster computing systems. Specifically, we address the problem of a unified framework for resource scheduling with emerging constraints that are important in cluster computing systems.

The motivation for studying this problem is that efficient resource utilization and minimum response time is one of the most important features to be included in cluster computing system infrastructure. Also, in a system where a number of jobs compete for resources, an efficient scheduling policy is fundamental to realize good performance for applications and higher resource utilization. The system may have powerful machines and interconnected by a high-speed network, but poor choice of a scheduling algorithm will lead to sub-optimal system performance and resource utilization. Also, in order to achieve high performance for I/O bound applications, the scheduling decisions must be based on all the required resources [1].

To this end, we propose a unified framework for resource scheduling in cluster computing systems. Our study shows that it is advantageous to schedule the system resources in a unified manner rather than scheduling each type of resource separately. We compare the proposed policy against two policies proposed in [4] and show that the proposed policy performs substantially better than the two policies proposed in [4].

The rest of the paper is organized as follows. In Section 2, the related work is presented. In Section 3, we present the system model and the proposed scheduling policy. Also, we present the two scheduling policies proposed in [4], which we used as testbeds to compare the effectiveness of our scheduling policy with. In Section 4 and Section 5, the performance analysis of the policies and results of the
experiments are discussed respectively. In Section 6, the conclusions and future directions is given.

2 Related Work

One of the major challenges in using cluster computing systems as HPC platform is to effectively utilize the available resources, which includes processors, communication network and files (i.e., storage resources) with objectives of optimizing resource utilization and response time. The current trends in job scheduling research concentrates in one of two ways. First, existing schedulers used in many distributed systems and parallel systems usually consider computational power and allocate resources based on the computational load index. The other is that many scheduling approach independently schedule processors (e.g., [7]) and I/O resources (e.g., [3]). However, separated processor and I/O resource scheduling can lead to poor utilization of resources when one type of resource is not available while others are available [1].

Recently, a framework that defines bindings between computing power and data repositories without addressing the scheduling policy issues have been proposed in [6]. We believe that aggregation of many resources is not enough to get good performance-careful scheduling of the jobs must be employed to achieve the best performance possible. Our work complements the work of [6] in that the proposed scheduling policy can be easily incorporated into their infrastructure.

There is very little work that considers the allocation of processors and I/O resources simultaneously [5]. There exist several unified framework for static environments. For example, in [1], a static scheduling approach in which the resource performance characteristics as well as full knowledge of the application service demand and arrival time is known a priori is discussed. Similarly, in [2], an approach based on a complete a priori knowledge of current loads, network conditions and topology is discussed. Also, scheduling of sequential jobs with a single CPU-I/O phase in isolation under the assumption that the jobs arrive to homogeneous dedicated system in a predictable manner is discussed in [4]. At best, the assumptions these policies are based on an ideal computing environment while the real situation in general and those of the system configuration addressed in this paper in particular are quite different from the ideal environments. For example, commodity-based clusters are more likely to be heterogeneous. This heterogeneity can cause load imbalance and reduce the performance of parallel applications. Heterogeneity may exist not only in processing power but also in I/O and network capabilities, which a serious issue for data input and output for large-scale scientific applications.

Efficient utilization of limited resources is the key to performance, and the scheduler has to deal with resource allocation and utilization adaptively. To address these problems, our algorithm allocates resources based on both the compute and the I/O load index. Our work deviates from the previous studies as we address the problem of a unified framework for resource scheduling with emerging constraints that are important in cluster computing systems. Also, the proposed policy requires no information collection as well as we make very little assumptions if any. Also, our goal is to maximize the mean response time of the jobs that compete for system resources. Unlike [1, 2, 4] in which jobs are assumed deterministic, we assume that the completion time for a job is not predictable. Also the objective in [2] is to optimize the performance of a particular application. Since multiple jobs share the system resources, optimizing the performance of an individual application may dramatically affect the completion time of other applications [1].

3 System Model and Scheduling Policies

3.1 System Architecture

The system is composed of loosely coupled S sites, each with P processors and D disks both of which are allocated, according to a particular job scheduling policy, to a set of jobs. Each site has a set of files, each of specified size, initially mapped to sites according to some distribution. As in [4], we assume that all processors have the same performance as well as can access any storage at its site.

A variety of applications that compete for the resources are submitted to the system from the community of users. These applications arrive to the system in an arbitrary manner for execution. There are several tools and services that allow users to develop and submit jobs to the system (e.g., [8]).

All I/O requests from a job is sent to the I/O
scheduler in the I/O subsystem, which coordinates the I/O requests 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. When an I/O device becomes available, the scheduler schedules a pending request from the wait queue using the FCFS policy. When the job and the I/O server are located in different clusters, we say that the I/O request is a remote request, but if they are in the same cluster we say that the I/O request is a local request.

3.2 Adaptive Distributed Scheduling Policy

We propose a distributed scheduling policy, which we refer to as Adaptive Distributed Scheduling (ADS) policy for cluster computing systems. Each site has a Job Scheduler (JS), a Site Scheduler (SS) and a Data Scheduler (DS). These three schedulers work in cooperation to optimize resource utilization as well as minimize the response time of the jobs. The general structure of the ADS is shown in Figure 1. The ADS policy works as follows:

1. Job Scheduler: The JS has several responsibilities. First, all jobs are submitted to the JS where they are originated. An arriving job declares the location and the size of the dataset it requires to the JS. Second, it keeps track of jobs that have their data located in the site but the job itself is located elsewhere. Third, it decides where an arriving job gets executed. Each of these are discussed as follows:

   (a) When a job arrives, if the dataset required by the job is present in the site, the job is sent to the SS provided that the number of jobs in SS is less than \( \Phi \). Otherwise, the job is queued in the wait queue and scheduled when resources become available.

   (b) In the case where the dataset of the job is located remotely, the following steps are performed:

      i. The JS sends information about the job and its queue length to the remote JS where the dataset is present.

      ii. When the message arrives at the remote JS, if it has less jobs than the job submission site or the size of the dataset required by the job is very large, it requests the job to be forwarded to it. Otherwise, it registers the job and its location information and then notifies the enquiring site to hold it.

   iii. Based on the response of the remote JS, the JS either forwards the job to the remote JS or queues it locally. In the later case, it also stores information about the remote JS such that it will only be contacted again if it comes back looking for its job.

2. Site Scheduler: Resources in each site are managed by a SS. No more than \( \Phi \) jobs (running + waiting) can be under the SS control at any given time. The SS works as follows:

   (a) It uses the first-come-first-served approach to allocate available processors to queued jobs.

   (b) Whenever the number of jobs in the site falls below \( \Phi \), the SS notifies the JS.

   (c) The JS responds to the request from SS as follows:

      i. If there are jobs queued locally, then the JS sends it to the SS.

      ii. Otherwise, if there are jobs with their dataset in the site waiting in remote site, the job with the largest dataset requirement is retrieved from the remote site and assigned to the SS.

      iii. If no job with its dataset in this site is found, a job from the most loaded site along its dataset are retrieved and assigned to the SS.

3. Data Scheduler: The DS manages the datasets in the local site. It can be configured as in [4] either to do nothing (NOREPLICATION) or replicate (REPLICATE) data file on remote site based on factors such as the popularity of the files to a randomly selected or onto a least loaded site.

Note that the ADS policy combines the AFFINITY and LOCAL variations presented in [4] without their drawbacks. Also note that the job and dataset are sent to the execution site while the site
was busy executing other jobs. As a result, the ADS overlaps computation and communication hiding latency quite nicely. Note also that we systematically control the frequency with which inter-site state information is collected. This substantially reduces both the number of messages exchanged between sites and the staleness of the state information.

### 3.3 Baseline Scheduling Policy

In [4] a distributed scheduling policy for GRID computing environments is discussed. Each site in the system has an External Scheduler (ES), a Local Scheduler (LS) and a Data Scheduler (DS). Jobs are submitted to the ES where they are originated. The scheduling policy works as follows:

1. When a job arrives, the ES then forwards the job to a site in which a job gets executed based on one of four alternatives: (1) a randomly selected site (RANDOM); (2) a site with the least number of jobs queued (LEASTLOADED), (3) a site where the job originated (LOCAL); and (4) a site that already has the required data (AFFINITY).

2. Once a job arrives at the designated execution site, the LS uses the first-come-first-served (FCFS) approach to allocate local resources to the jobs in the job wait queue.

3. The DS can be configured to do nothing (NOREPLICATION) or replicate the file on remote site based on the popularity of the local files. In the later case, the choice of the site for replication can be randomly selected (RANDOM) or a least loaded site (LEASTLOADED).

The authors found that under all combination of data scheduling and job scheduling, the AFFINITY and the LOCAL based policies performs as good or better than the other two approaches. Therefore, we concentrate on these two approaches in the rest of the paper.

### 4 Performance Evaluation

We used a discrete event simulation to compare the scheduling policies discussed in the previous section. Table 1 shows the default system and workload parameters used in the simulation. Jobs arrive to the Job Scheduler with Poisson inter-arrival times [3, 7]. We assume that each job requires as single file and the execution time of a job is computed as follows [4]:

\[
T_{\text{execution}} = \text{startupTime} + (T_{\text{compute}} \times \beta_{\text{CPU}}) \times T_{\text{comm}}
\]

where \(\beta_{\text{CPU}}\) and \(\beta_{\text{NET}}\) are the processor contention factor and communication contention factor respectively. We used a simple communication model expressed as follows.

\[
T_{\text{comm}} = \text{Latency} + \frac{\text{Message size}}{\text{Bandwidth}}
\]

We model network contention by keeping track of the number of simultaneously data transfers across a link and decreasing the bandwidth available for each transfer accordingly. We incur startupTime when initiating computation on multiple processors in cluster computing environments, which we accounted for in the experiments by setting the background job utilizations to 30% at most.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>disks/site</td>
<td>1</td>
</tr>
<tr>
<td>Mean service demand</td>
<td>300.0</td>
</tr>
<tr>
<td>Sites</td>
<td>4</td>
</tr>
<tr>
<td>Processors/site</td>
<td>8</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>100MBytes/sec</td>
</tr>
<tr>
<td>Latency</td>
<td>10μsec</td>
</tr>
</tbody>
</table>

### 5 Simulation Results and Discussions

Figure 2 shows the mean response time (MRT) of the three policies (y-axis) as a function of the utilization (x-axis) while all other parameters are set to their default values as shown in Table 1. From the data on the graph, we can observe that ADS performs substantially better than the AFFINITY and LOCAL policies while LOCAL performs better than the AFFINITY approach.

This trend can be explained by the fact that the ADS policy combines the approaches of both
LOCAL and AFFINITY without their drawbacks. Moreover, jobs may execute at sites that are distant from their data in both LOCAL and ADS. This leads to a remote I/O access across networks, thus the I/O cost is many times worse than a local I/O cost. However, ADS hides the communication latency by overlapping the computation and communication effectively reducing the impact of Remote I/O on the performance of the jobs. Last but not least, ADS considers both I/O and CPU load factors in making scheduling decisions showing that importance of taking into account the impact of these two factors.

We also observed ADS to have better utilizations than the AFFINITY and LOCAL policies. The reason for this is that in LOCAL policy, a major portion of the compute processors' time is wasted on waiting for I/O to complete, yielding a poor efficiency. The AFFINITY experiences the worst utilization. This is because jobs are held in a site while there are no jobs to do in other sites. In contrast, ADS moves both jobs and data before they are needed as such minimizing the idle time by the processors.

6 Conclusion and Future Directions

In this paper, we presented a unified CPU and I/O resource scheduling policy for multi-programmed cluster computing systems and compared its performance with a two recently proposed policies. We showed that the proposed policy performs substantially better than these two policies as it gives the lowest response time and the highest utilization. In this paper, we did not consider how the application data is distributed among the D disks nor did we deal with how replicated data are managed. The impact of these issues on the scheduling policy need to be addressed, which we plan to do in the future. Also, access to remote data is one of the principal challenges of cluster computing. While performing I/O, applications must be prepared for server crashes, performance variations, and exhausted resources. Also, in addition to I/O requests from the parallel job, there are background I/O requests on the disks, which we have not modeled in this paper.

References

Figure 1: System Architecture and Adaptive Distributed Scheduling Policy.

Figure 2: Relative performance of the policies as a function of Utilization.