An Integrated Approach to Task Scheduling and File Replication[*]

Gaurav Khanna[†], Umit Catalyurek[‡],
Tahsin Kurc[‡], P. Sadayappan[†], Joel Saltz[‡]
[†] Dept. of Computer Science and Engineering, [‡] Dept. of Biomedical Informatics
The Ohio State University

Abstract

In this paper we investigate task scheduling and data replication for execution of a batch of data-intensive tasks with batch-shared I/O behavior on a system consisting of coupled storage and compute clusters. We propose a strategy that formulates the problem of scheduling the batch of tasks in conjunction with data replication as a 0-1 integer programming problem. The goal is to minimize the batch execution time by achieving load balance across compute nodes and minimizing the overall data transfer cost. We compare our approach with other previously proposed strategies, and demonstrate its effectiveness through both simulations as well as experimental studies on a cluster.

1 Introduction

The ability of researchers in engineering, biomedicine, and other fields of science to collect and generate scientific data sets has improved significantly with the help of advanced instruments and the increased computational capacity of high-end computer systems. Most scientific applications store datasets in collections of files. A request for the region of interest specifies a subset of data files, which make up the dataset, or segments of the files – either as part of the parameters of the request or after an index lookup, which finds the files and file segments that can address the request. The data of interest is retrieved from the storage system and transformed into a data product, which is more suitable for examination by the scientist.

In this paper, we propose an integrated scheduling and replication strategy to efficiently schedule a batch of data-intensive tasks exhibiting batch-shared I/O behavior [14] on coupled storage and compute clusters. The goal is to minimize the execution time of the batch. In a coupled storage-compute cluster system, a number of disks attached to low power PCs can form a storage pool. This storage pool is connected to a powerful cluster over local area networks. In our model, when a task is scheduled to a processing node, the files accessed by the task are staged to the processing node before the task is executed.

In an earlier work [9], we modeled the sharing of files among tasks as a hypergraph and employed hypergraph partitioning to obtain a partitioning of tasks onto compute nodes that computationally balanced the workload and reduced remote I/O operations for file transfers. Our earlier

[*]This research was supported in part by the National Science Foundation under Grants #CCF-0342615, #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ANI-0330612, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497), NIH NIBIB BISTI #P20EB000591, Ohio Board of Regents BRTTC #BRTT02-0003.

work assumed that a compute node had enough disk space to hold all of the files staged to that node (i.e., we assumed infinite disk cache space on each compute node) and did not take into account replicas of files on the compute nodes. In contrast to our earlier work, we allow explicit replication of files on compute nodes based on their popularity. We present a 0-1 Integer programming based formulation that solves for the mapping of tasks to nodes, sources and destinations for all replication operations, and the destination nodes for all remote transfer operations in an integrated fashion. The algorithm also handles the case in which the storage space on the compute cluster may be limited and may not be sufficient to store all the files at once. We experimentally evaluate the proposed approach using simulations and on real machines with application emulators from three application domains; analysis of remotely-sensed data, biomedical imaging, and particle physics data analysis.

2 Problem Definition

We target batches which consist of independent sequential programs. Each task requests a subset of files in the environment and can be executed on any of the nodes in the compute cluster. Data files required by a task should be staged from the storage cluster to the compute cluster for the task to execute correctly. A data file is the unit of I/O transfer from the storage cluster to the compute cluster. The tasks in the batch may share a number of files. If a file is required for processing by one or more tasks on a particular node, it may be retrieved either from the remote storage system or from another compute node which already has the file. The decision to replicate a file is dependent on the scheduling of tasks which require that file and the decision to schedule a task on a node depends upon the replication of files and remote transfers since it essentially determines whether the task will be able to get its required data locally if its allocated to that node.

Our objective is, given a batch of tasks and a set of files required by these tasks, 1) to schedule the tasks in an efficient manner, 2) to decide which files need to be remotely transferred and their respective destination nodes, and 3) to determine which files need to be replicated and their source and destination node information, so as to minimize the batch execution time. Figure 1 depicts an illustration of this problem. Each task in the batch is represented by a computation weight, a list of input files, and their file sizes.

3 Strategies for Coordinated Scheduling and Replication

We compare our approach against two other algorithms. The first algorithm is based on the traditional MinMin scheduling with implicit replication of files. The second algorithm is the batch mode version of the one proposed by Ranganathan et al. [13]. The algorithm combines a scheduling scheme, called *Job Data Present* with a replication heuristic, referred to as *Data Least Loaded* in a decoupled fashion. These algorithms are described below.

MinMin with Implicit Replication. This algorithm computes the expected minimum completion time (MCT) of each task on each node in the system. Among the unscheduled tasks in the batch, it chooses the task that can complete the earliest and assigns it to the node that can execute that task fastest. When computing the expected MCT of a task on a node, MinMin takes into account the files already available on the node and files already available on other compute nodes which can therefore act as alternate sources for creating file replicas other than the remote storage system. The replication policy in MinMin is implicit. When a task is scheduled on a node, all of its files are

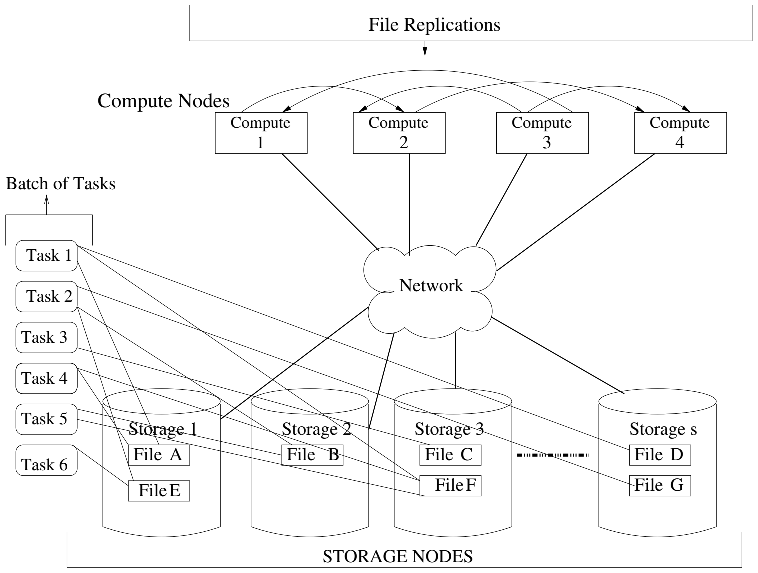


Figure 1: Scheduling problem.

staged to the corresponding node. This leads to an implicit replication policy as multiple copies of files may be created on different nodes of the compute cluster. Each file required for a task is staged from one of the replicas or from the storage cluster such that the time to transfer the file is minimized.

Batch mode variant of Job Data Present with Data Least Loaded. In the algorithm presented in [13], a single file per task is employed which means that either a compute node stores the file required by a task or it does not store the file. The algorithm incorporates a notion of eligible nodes for each task, which are the set of nodes that store the file required by the task. It works by picking a task from a FIFO queue and assigning it to the node that already has the required data. If more than one compute nodes are eligible candidates, then it chooses the least loaded node. In our case, we allow multiple files per task which means that there may exist compute nodes which store subsets of the files required by a job. This essentially amounts to allocating a job to a node such that the expected data transfer time to stage in the set of files required by a task is minimized. The replication mechanism *Data Least Loaded* is decoupled from the scheduling policy. The replication mechanism keeps track of the popularity of files, and when the popularity of a file exceeds a threshold, then the file is replicated to the least loaded node in the compute cluster.

4 0-1 Integer Programming-based Approach

We assume that each node on the compute server has a local disk, which can be used as disk cache for files staged from storage nodes. We first present the IP formulation for the *unlimited disk cache space* case. In this case, each compute node has enough space to store at least one copy of each file requested by the tasks in the batch. We then describe an extension to handle when disk cache space is limited. In the *limited disk cache space* case, each compute node has sufficient space to store all the files for at least one task in the batch.

4.1 Unlimited Disk Cache Space

In the following discussion we will use subscripts i and j for compute nodes, k for task and ℓ for file. For each task t_k the set of files accessed by that task is denoted by $Access_k$. The set of tasks accessing a particular file f_ℓ is denoted by $Require_\ell$. Let $X_{\ell i}$ be a binary variable where $X_{\ell i}$=1 if file f_ℓ is stored at node c_i, 0 otherwise. Let $Y_{ij\ell}$ be a binary variable where $Y_{ij\ell}$=1 if node c_i replicates file f_ℓ on node c_j, 0 otherwise. Let $R_{\ell i}$ be a binary variable where $R_{\ell i}$=1 if file f_ℓ is remotely transferred to node c_i, 0 otherwise. Let T_{ki} be a binary variable where T_{ki}=1 if task t_k is allocated to node c_i, 0 otherwise.

The objective function is the minimization of the overall batch execution time under a set of constraints. The constraints are defined as follows:

A compute node can only replicate a file on another compute node if the former has the file present locally.

$$(\forall i)(\forall j, j \neq i)(\forall \ell) Y_{ij\ell} <= X_{\ell i} \tag{1}$$

A replication operation of a file to a destination node is only required if there is some task allocated to the that node which requires the file.

$$(\forall i)(\forall j, j \neq i)(\forall \ell) Y_{ij\ell} \leq \sum_{k \in \ Require_\ell} T_{kj} \tag{2}$$

For a particular node c_i and a file f_ℓ, There is at most only one other node c_j which can replicate the file on the node c_i.

$$(\forall i)(\forall \ell) \sum_{\forall j, j \neq i} Y_{ji\ell} \leq 1 \tag{3}$$

Each task is allocated to only a single node in the system.

$$(\forall k) \sum_{\forall i} T_{ki} = 1 \tag{4}$$

The storage of a file on a node is either the result of a remote transfer or a replication of the file on that particular node.

$$(\forall i)(\forall \ell) X_{\ell i} = R_{\ell i} + \sum_{\forall j, j \neq i} Y_{ji\ell} \tag{5}$$

Both a remote transfer operation and a replication operation for a particular file on a particular destination node is not allowed.

$$(\forall i)(\forall \ell) R_{\ell i} + \sum_{\forall j, j \neq i} Y_{ji\ell} \leq 1 \tag{6}$$

The allocation of a task to a node entails the staging of all the files required by the task onto the node.

$$(\forall i)(\forall k)(\forall \ell \in \ Access_k) T_{ki} \leq X_{\ell i} \tag{7}$$

And last, every file should have at least one remote transfer.

$$(\forall \ell) \sum_{\forall i} R_{\ell i} \geq 1 \tag{8}$$

The overall batch execution time $Batch\_Exec\_Time$ is the maximum of the execution time of each node. The goal is to minimize $Batch\_Exec\_Time$ subject to the above mentioned constraints. The overall execution time of a node c_i is defined as $Exec_i$. It is composed of three additive components: the overall replication cost associated with that node ($Replication_i$), the computation cost of tasks allocated to that node ($Computation_i$), and the remote transfer cost of files to that node ($Remote_i$). Let t_{rep} be the replica creation cost per byte, t_{rem} the per byte remote transfer time, and $Comp_k$ the computation cost of task t_k.

$$\begin{aligned}
Replication_i &= \sum_{(\forall \ell)(\forall j, j \neq i)} (Y_{ji\ell} + Y_{ij\ell}) \times t_{rep} && (9)\\
Computation_i &= \sum_{\forall k} Comp_k \times T_{ki} && (10)\\
Remote_i &= \sum_{\forall \ell} R_{\ell i} \times t_{rem} && (11)\\
Exec_i &= Replication_i + Remote_i + Computation_i && (12)\\
Batch\_Exec\_Time &= \max_{\forall i}\{Exec_i\} && (13)
\end{aligned}$$

The solution to the proposed IP formulation yields complete information regarding the allocation of tasks to compute nodes, source and destination nodes for all replication operations, and destination nodes for all remote transfer operations. It provides a one-step solution which comprises of both the schedule as well as data placement by efficiently exploiting the global task-file sharing information. A drawback is that it does not model disk space constraints on the compute cluster. To address the problem of limited disk space, we propose a 2-stage Integer Programming formulation.

4.2 Limited Disk Cache Space: A two-stage 0-1 IP Formulation

The limited disk cache space approach assumes that there is enough space on each compute node to store files required for at least a single task. Since the available disk space on the compute cluster is not sufficient to stage in all the files required by the batch under consideration, a disk file eviction mechanism is needed in conjunction with the IP based integrated scheduling and replication approach. The file eviction mechanism is discussed in more detail in Section 4.3.

The first stage of the proposed algorithm accepts as input a set of tasks and yields a subset of the tasks (referred to here as a sub-batch), which can execute on the compute cluster without violating the disk space constraints. The second stage accepts this sub-batch as an input and applies the 0-1 IP formulation for the unlimited disk space with an additional constraint. The additional constraint is required since even though the storage requirements of the sub-batch do not exceed available disk space on compute nodes, still the space on a particular compute node may not be sufficient to stage in all the files required by the sub-batch. Each sub-batch execution phase is followed by an intelligent file eviction phase which deletes unwanted files and less popular files. The two stage solution is then repeatedly applied on the remaining set of pending tasks. Subsequent iterations of this two stage solution also model the fact that copies of some files have already been created on the compute cluster due to previous sub-batch executions.

First Stage: Sub-batch Selection. The key idea is to divide a batch of tasks into maximally sized subsets of tasks such that the tasks in a subset can execute on the compute cluster without the need of any file eviction mechanism. Choosing a maximally sized subset of tasks which can fit into the available disk space essentially amounts to choosing a subset of tasks which have high degree of file sharing among themselves. In addition, allocation of the tasks in the sub-batch across compute nodes should be computationally balanced. We now describe the IP formulation for sub-batch selection. We assume we have the same set of input parameters as explained before. The constraints are defined as follows.

The allocation of a task to a node entails the staging of all the files required by the task onto the node.

$$(\forall i)(\forall k)(\forall \ell \in \ Access_k) T_{ki} \leq X_{\ell i} \tag{14}$$

The total storage space taken by files stored on a particular node should not exceed the disk space available on that node.

$$(\forall i) \sum_{\forall \ell} X_{\ell i} \times filesize(f_\ell) \leq DiskSpace_i \tag{15}$$

A task cannot be allocated to more than one node in the system. Note that for sub-batch selection, we do not enforce the constraint that all tasks be allocated in the system since we cannot do so with limited disk space.

$$(\forall k) \sum_{\forall i} T_{ki} \leq 1 \tag{16}$$

To have a load balanced mapping of tasks in a sub-batch to compute nodes, we enforce a constraint that the computation time on any node should be within a certain tolerance $Thresh$ of the average computation time over all the nodes. Essentially what it means is that the sub-batch should not be chosen in such a way that the eventual sub-batch allocation in the second stage leads to a load imbalanced solution. Let $Average\_Computation\_time$ denote the average of the computation times over all the nodes.

$$\begin{aligned}
(\forall i) Computation_i &<= Average\_Computation\_Time \times (1 + Thresh) && (17)\\
Computation_i &= \sum_{\forall k} Comp_k \times T_{ki} && (18)\\
Average\_Computation\_Time &= \frac{1}{C} \times \sum_{\forall i} Computation_i && (19)
\end{aligned}$$

where C is the number of compute nodes. The objective function is to choose a load balanced maximally sized subset of tasks which does not violate disk space constraints.

$$Objective\_Function = \max_{(\forall i)(\forall j)} \sum T_{ij} \tag{20}$$

Second Stage: Sub-batch Allocation Optimized for Overall Execution Time. The sub-batch selection phase yields a subset of tasks and their allocations. However, the allocation need not be the best allocation in terms of minimizing the execution time, because the sub-batch selection stage does not try to minimize the execution time by exploiting the differential that exists between fetching a file from a nearby compute node as opposed to fetching it from the remote system. To achieve the best possible allocation for the chosen sub-batch, we input the set of tasks chosen in the sub-batch to the 0-1 IP formulation given in Section 4.1 with an additional constraint for disk space. The total storage space taken by files allocated to a particular node should not exceed the disk space available on that node.

$$(\forall i) \sum_{\forall \ell} X_{\ell i} \times filesize(f_\ell) \leq DiskSpace_i \tag{21}$$

This constitutes the stage two of the algorithm and yields the schedule as well as data placement information for the sub-batch. Since the sub-batch allocation obtained after the first phase is a solution which does not violate disk space constraints, we are always guaranteed of a feasible solution after the second stage. In the case where the second stage cannot improve upon the solution obtained after stage one, we will get the same solution for task allocation in addition to the replication and remote transfer information.

4.3 File Eviction Policy

Once a sub-batch finishes execution, a disk file eviction mechanism is invoked which deletes files in the increasing order of their popularity. The amount of data to be evicted on each compute node is chosen in such a way that at the end of eviction, each node has at least as much amount of space required to execute a single task. This is followed by the 2-stage process explained before with input being the set of remaining pending tasks. We incorporate a notion of popularity of access with each file. The popularity of file f_ℓ, $Popularity_\ell$ is calculated as follows.

$$Popularity_\ell = \frac{Access\_Freq_\ell \times filesize(f_\ell)}{Numcopies_\ell} \tag{22}$$

$Access\_Freq_\ell$ represents the number of pending requests to the file. This information can be easily obtained from the original batch and the set of tasks which have already finished execution. $filesize(f_\ell)$ represents the size of the file f_ℓ. $Numcopies_\ell$ represents the number of copies of file f_ℓ in the compute cluster. The idea behind incorporating number of copies is that if two files have the same probability of access and the same size, then the file with fewer copies should get a higher popularity and thereby reduce its chances of eviction. The fact that popularity is directly proportional to the access frequency is fairly obvious while the intuition behind putting in the file size is that greater the size of the file, greater will be the cost of getting the file back to a node from which it has been evicted. Therefore, it makes more sense to evict smaller sized files since the cost of staging such files again in future will be less.

We have integrated this file eviction mechanism into our proposed approach as well as *MinMin* with *Implicit Replication* for the purpose of performance comparison. For the algorithm *Job Data Present* with *Data Least Loaded*, we employ an LRU based eviction mechanism [13].

4.4 Hierarchical IP-based Approach

The performance of the IP based formulation in terms of the time it takes for the solver to yield a solution is sensitive to the problem size and the number of compute nodes in the system. The proposed IP based approach may not be suitable if applied directly for problems which involve large number of compute nodes. To address this issue, we propose a Hierarchical IP-based solution which basically uses the proposed IP formulations as building blocks and applies them in a hierarchical fashion.

Let C be the set of compute nodes in the system. A two level hierarchy on a compute cluster can be imposed by abstracting the C nodes as a collection of $C_{toplevel}$ virtual nodes each of which is composed of a disjoint subset of nodes from C. Each virtual node thus forms a virtual cluster. A multi-level hierarchy can be imposed in a similar manner such that a flat configuration consisting of C compute nodes can be abstracted as a tree where each non-leaf level consists of virtual nodes at that level and the leaf level represents the actual compute nodes. The amount of disk space on a virtual node is the total amount of space across the leaf level nodes which are descendants of this particular virtual node. Given a multi-level hierarchy of a C node compute cluster, we apply the first phase of sub-batch selection successively at each level of the tree. This is followed by applying the second phase only at the leaf level of the tree. For example, given a two level hierarchy consisting of $C_{toplevel}$ virtual nodes at the top level and C nodes at the leaf level, we first apply the phase 1 of the algorithm over the $C_{toplevel}$ virtual nodes. The output of the first phase gives a sub-batch for each virtual node. This is followed by applying the phase 1 again on each of the virtual nodes where each virtual node consists of a disjoint set of leaf nodes. Finally, we apply the phase 2 for each virtual node. The output of this is an allocation of tasks to the leaf nodes. This is followed by a task execution phase and then the entire hierarchical process repeats on the set of pending tasks. The benefit of the hierarchical scheme is that it partitions the overall problem into smaller subproblems which are run on smaller subsets of compute nodes and thus should benefit considerably in terms of overall solver time. The drawback is the sub-optimality introduced due to abstracting the problem as a set of disjoint subproblems instead of solving the single large problem in a global fashion.

5 Experimental Results

We evaluated the scheduling algorithms through real experiments and simulations, against three application classes: satellite data processing, biomedical image analysis and particle physics analysis.

5.1 Application Workloads

To generate datasets for the satellite data processing application (referred to here as **SAT**), we used the emulator developed in [15]. The application [6] operates on data chunks that are formed by grouping subsets of sensor readings that are close to each other in spatial and temporal dimensions. These chunks can be organized into multiple files. In our emulation, we assigned one data chunk per file. A satellite data analysis task specifies the data of interest via a spatio-temporal window. When multiple scientists access these datasets, there will likely be overlaps among the set of files requested because of ”hot spots” such as a particular region or time period that scientists may want to study.

For the image analysis application (referred to here as **IMAGE**), we implemented a program to emulate studies that involve analysis on images obtained from MRI and CT scans (captured on multiple days as follow-up studies). An image dataset consists of a series of 2D images obtained for a patient and is associated with meta-data describing patient and study related information (in our case, we used patient id and study id as the meta-data). Each image in a dataset is associated with an imaging modality and the date of image acquisition and stored in a separate file. An image analysis program can select a subset of images based on a set of patient ids and study ids, image modality, and a date range. In a research study, systematic assessment of image analysis techniques requires an ability to efficiently invoke candidate image quantification methods on large collections of image data. A researcher may, for instance, apply several image analysis methods on overlapping subsets of image datasets containing thousands of 2D and 3D images in order to assess their ability to predict outcome of a treatment across patient groups.

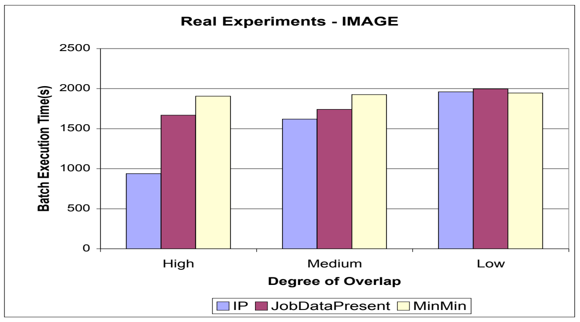
The particle physics analysis application involves analysis of a large number of particle collision events. For this application (referred to here as **PPA**), we implemented a program to emulate recording of information about collision events between particle beams. The particle physics dataset consists of a set of files where each file represents information about a set of such collision events. Collision events can be analyzed by multiple jobs. Each job is assigned a single file. Multiple jobs may use the same file.

We evaluated the system for three different types of workloads; *high overlap*, *medium overlap*, and *low overlap*, each of which represents different amounts of file sharing among tasks in a batch. For SAT, we simulated queries directed to geographically distant parts of the world. Four sets of queries were generated representing the queries directed to 4 hot spot regions. Across the sets, there is no overlap between the queries, and in each set, queries are adjusted such that for high overlap workload, they resulted in a 85% overlap, on average, in terms of files requested by different tasks in the batch. Similarly, we generated medium and low overlap workloads with 40% and 10% overlap, respectively. For IMAGE, different degrees of overlap is achieved by varying the values of patient and time attributes across requests by different tasks. We generated workloads with 85%, 40%, and 0% overlap for high, medium, and low overlap cases. For PPA, we generated workloads with single file per task. This is the characteristic of most particle physics data analysis applications in that files accessed by a single job are all independent of each other so that a job that requires n files can be split into n sub-jobs each requiring a single file. We employed a high file sharing pattern similar to the high overlap workloads explained above to govern the file sharing behavior across tasks in PPA.

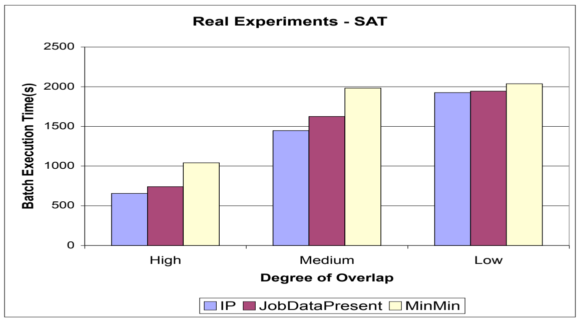
We generated 20 days worth of data, about 25 GB for SAT. The data was distributed across the storage nodes using a Hilbert-curve based declustering method [8]. Each file in the dataset was 50 MB. In the high overlap case, each task accessed on an average 8 files. In the medium and low overlap cases, each task accessed on an average 14 files. For IMAGE, the dataset generated by the emulator corresponded to a dataset of 2000 patients and images acquired over several days from MRI and CT scans. Each task on an average accessed 6 files. The sizes of images were 4 MB and 64 MB for MRI and CT scans, respectively. The overall size of the dataset was around 128 GB. Images for each patient were distributed among all the storage nodes in a round robin fashion. For PPA, the overall size of the dataset was about 50 GB. The sizes of files in the dataset was 4 MB and 64 MB. The files were distributed among the storage nodes in a round robin fashion.

In order to create data intensive workloads which are targeted in this paper, we set the processing time for each task to be 0.001 seconds per Megabyte of data.

The proposed integer program used a publicly available solver called **lp_solve** [2]. lp_solve is written in ANSI C and can be compiled on many different platforms. lp_solve is suited to solving large scale problems involving large number of variables and constraints.



(a)



(b)

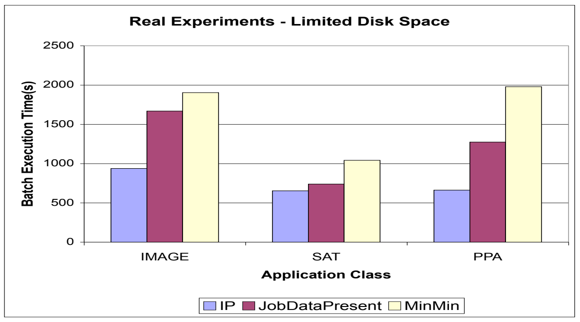
Figure 2: Batch execution time achieved by different algorithms on (a) IMAGE and (b) SAT.

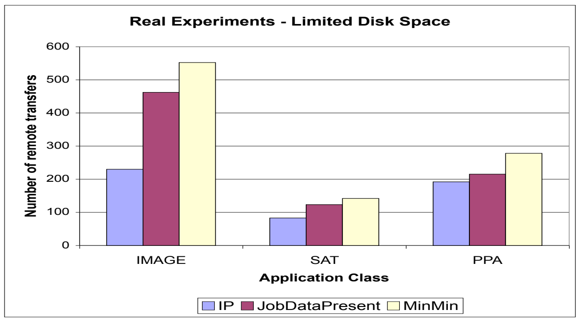
5.2 Performance Evaluation on Real Machines

Our experiments were carried out using a compute cluster and a single storage cluster as described below. The compute cluster being employed (**OSC**) is a compute cluster at the Ohio Supercomputer Center. The compute cluster consists of dual-processor nodes equipped with dual 2.4 GHz Intel P4 Xeon processors with hyper threading, resulting in 4 virtual CPUs per node. Each node has 4 GB of memory, 62 GB of local scratch space, interconnected by an 8 Gbps Infiniband switch. The storage cluster is a cluster of Pentium III 933 MHz nodes (**OSUMED**). Each node of this cluster has 300 GB disk space and 512 MB of memory. The disk bandwidth available on these storage nodes varies from 18 MB/sec to around 25 MB/sec. Using micro benchmarks, we measured the bandwidth of the shared link between the OSUMED cluster and the OSC cluster to be around 100 Mbps.

Figure 2 shows the relative performance of the various scheduling/replication schemes on workloads with different degrees of shared I/O among tasks. These experiments were conducted using 4 compute nodes and 4 storage nodes. These experiments were done with workloads from two different application classes IMAGE and SAT. The IMAGE workload consisted of 100 task batch where the average amount of data accessed by a task in the batch was around 300MB. The titan workload consisted of 100 tasks with an average per task data requirement of around 600MB. The experiments were representative of a limited disk capacity on the compute cluster. Specifically, The disk capacity on each node in the cluster was 500MB for the IMAGE experiment and 1500MB for SAT. As is seen from the figures, the IP based strategy performs better than the other algorithms for all cases. This is because the IP formulation is able to leverage the global task-file affinity information by incorporating it into its goal function of minimizing the batch execution time. In addition, while minimizing the networking and I/O overheads, The IP approach also maintains computational load balance across the nodes. The benefit of the IP based approach is maximum for the high overlap workload and reduces as the degree of overlap decreases, as expected. The base schemes do not perform as well as the IP based approach because they are greedy heuristics and hence make local decisions without exploiting the task-file affinities. Among the base schemes, *Job Data Present* coupled with *Data Least Loaded* does better than *MinMin* with *Implicit Replication*. This is because the scheduling scheme *Job Data Present* favors data locality and hence is able to make good use of the dynamic data replication of most popular datasets performed by the replication algorithm *Data Least Loaded*.

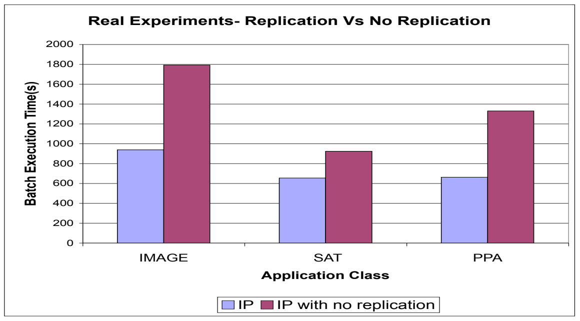
Figure 3(a) shows the the relative batch execution times obtained by applying the schemes on

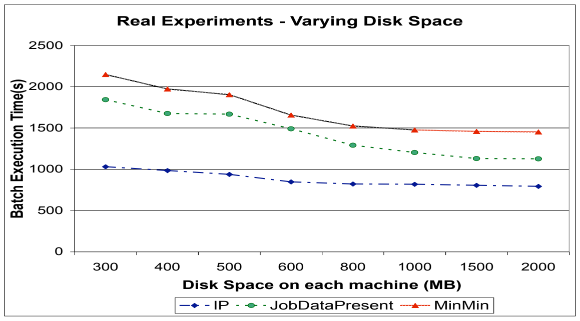




(a) (b)

Figure 3: The performance of the scheduling strategies in the high overlap case for all the application classes (a) Batch execution time. (b) The number of files accessed remotely from the storage cluster.





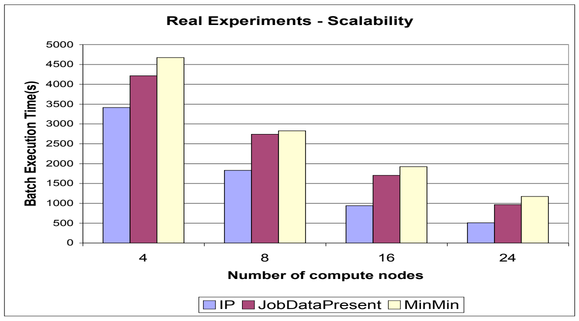
(a) (b)

Figure 4: (a) Benefit of compute node to compute node data replication over no replication. (b) Variation of the batch execution time with respect to disk space on the compute cluster.

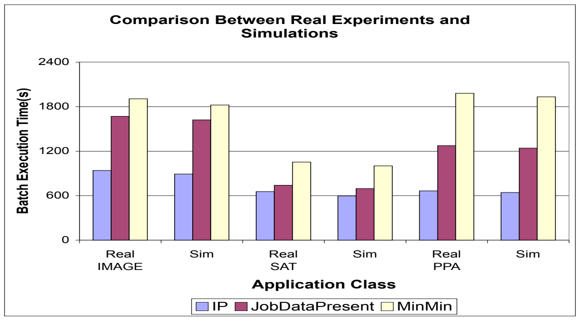
workloads belonging to various application classes. These experiments were conducted using high overlap workloads for all the application classes on 4 compute nodes and 4 storage nodes. The experiments were run with limited disk configurations as explained above. The workloads used for IMAGE and SAT are the same as used in the previous set of experiments. For PPA, we employ a 500 task high overlap workload with 500MB space on each compute node. The results show that the IP based strategy performs much better as compared to the other two schemes in terms of the batch execution time. The IP based approach also minimizes the total number of remote file transfers over the network between the compute and the storage cluster as shown in Figure 3(b). *Job Data Present* does second best since it favors data locality and simultaneously exploits runtime replication of popular datasets.

Figure 4(a) quantifies the benefit which can be obtained through data replication. The *IP with no replication* scheme here refers to the IP based strategy with no replication. This is done by suitably disabling certain constraints and setting certain variables to 0 in the IP formulation. The results show that replication gives significant speedup over an efficient scheme which performs no data replication in an environment where the remote file transfer cost is significantly greater than the compute node to compute node transfer cost. Thus, replication here is indeed justified.

Figure 4(b) demonstrates how the proposed scheme and the base schemes perform with respect to variation in disk space on the compute cluster. These workload used for this experiment was



(a)



(b)

Figure 5: (a) Performance of different algorithms for IMAGE with varying number of compute nodes, (b) Comparison of real experiments and simulations for different algorithms for all application classes

a 100 task IMAGE workload. The disk space on each machine on the compute side is varied from 300MB to 2GB. The results show that as the disk space increases, the base schemes show a greater decrease in the overall batch execution time. However, The IP based scheme still performs the best. This is an expected result, since the base schemes suffer a lot of file evictions on the compute cluster when the disk space is limited. The IP approach, on the other hand makes efficient allocations of tasks and files and therefore, suffers considerably lesser evictions. As the disk space increases, the base schemes are thus able to get more benefit out of it than the IP based strategy.

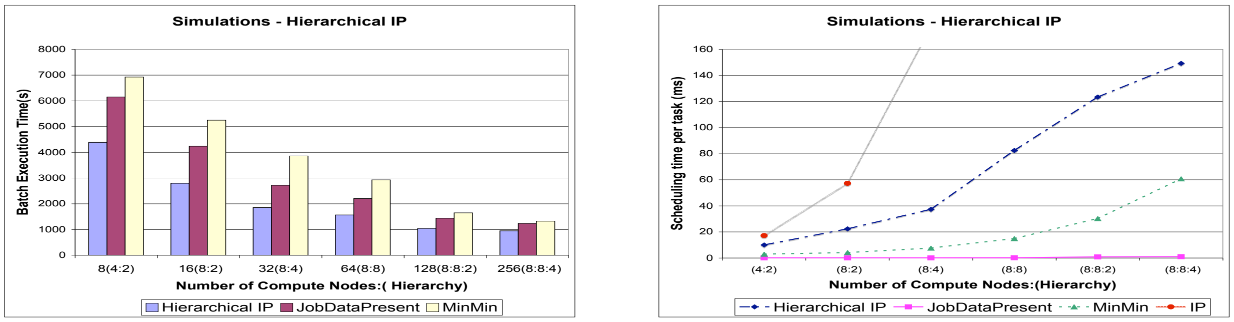
To analyze the performance of our proposed scheme with respect to varying the number of compute nodes in the system, we ran experiments of high overlap workload consisting of 400 IMAGE tasks. These experiments were run with a limited disk configuration where the space on each node of the compute cluster is 500MB. Figure 5(a) shows the results with varying number of compute nodes. As is seen from the figure, the IP based strategy achieves better performance than the other strategies in all configurations.

5.3 Performance Evaluation through Simulations

We used simulations to understand the performance of the various scheduling schemes on larger systems. We ran our simulations using the **Simgrid Toolkit** [3, 11]. In our simulations, we used version 2.18.5 of this toolkit. This toolkit implements event-driven simulation of applications on heterogeneous distributed systems. Since Simgrid does not provide an abstraction for disk, we modeled the disk as a shared link (with bandwidth equal to disk bandwidth) which is time-sliced.

For the purpose of validating the simulations, we simulated a hardware configuration similar to the experimental setup for the real experiments. Nodes within the cluster are homogeneous in terms of processing capability and local disk bandwidth. The network between compute cluster and the storage nodes is simulated as a 100 Mbps links. Figure 5(b) shows the comparison between the real experiments and the simulated results for all the application domains. We see that the relative trends of the simulated results closely follow those of the real experiments even though the absolute values vary slightly.

Figure 6(a) shows the batch execution time for a 2000 task PPA workload with varying number of compute nodes. These simulations were conducted with the hierarchical IP approach as explained earlier along with the other schemes. The hierarchy chosen for a particular number of



(a) (b)

Figure 6: The performance of the hierarchical scheme with varying number of compute nodes (a) Batch execution time. (b) Scheduling overhead.

compute nodes is shown in the graph. For example, 8(4:2) in Figure 6(a) says that the hierarchical version over 8 compute nodes was run using a (4:2) configuration with 4 virtual nodes at the top level and 2 real nodes as children of each virtual node. The results show that the multi-level hierarchical scheme does better than the base schemes in all the cases. However, the relative performance improvement decreases. This is due to the fact that as the number of compute node increases, the extent of sub-optimality introduced by the hierarchical approach increases. Since the primary motivation behind the hierarchical approach was to lower the scheduling overhead, therefore we also measured the scheduling overhead associated with the scheme relative to that of other schemes. Figure 6(b) shows the corresponding per job scheduling times (in milliseconds). The graphs show that the IP based approach becomes considerably faster relative to the flat IP scheme and shows that the feasibility of the scheme over higher number of compute nodes. The scheduling time of *MinMin* shows an increasing trend with respect to the number of compute nodes. This is because, the scheme is inherently quadratic. The *Job Data Present* scheme as expected, performs the best in terms of scheduling time. This is because, its a dynamic scheme and at each step chooses the next set of tasks to execute which is linear with respect to the number of compute nodes. It does not have to iterate over all task-host pairs as in *MinMin*.

6 Related Work

Relatively little research has so far addressed the co-scheduling of task execution and data replication. Ranganathan et. al. [13] proposed a decoupled approach to scheduling of computations and data for data-intensive applications in a grid environment, and demonstrated its effectiveness via simulation studies. In contrast, our approach seeks an integrated formulation of both scheduling and data replication for a batch scheduling context - where a set of independent jobs is to be collectively scheduled and inter-job file affinity information among the batch of jobs can be analyzed and utilized. While the decoupled approach of Ranganathan et. al. [13] is well suited to an online environment where tasks arrive over time and there is a lack of knowledge about file access patterns of future jobs, our integrated approach to scheduling and data replication is more effective in the batch context we consider. Casanova et al. [4] modified the MinMin, MaxMin, and Sufferage job scheduling heuristics to take into account the cost of inter-site file access, in the context of scheduling parameter sweep applications in a Grid environment. Our work targets an environment with a compute cluster and storage cluster, and explicitly models the effect of file replication.

Desprez et al. [7] proposed an algorithm that combines data management and scheduling using a steady state approach. In their model, it is assumed that the aggregate available disk space over all compute nodes is adequate to hold at least one copy of all the files needed by the set of jobs to be scheduled. However, our formulation takes into account the fact that the disk space on the compute servers may be not be sufficient to concurrently store at least one copy of each file. To address the issue of limited space, we employ a 0-1 integer programming based batch-partitioning strategy, coupled with an intelligent file eviction mechanism that evicts files based on a notion of popularity of access. While their work targets the placement of data, the cost of creating a replica from one of many possible sources of data is not a focus. Our work addresses both the data placement as well as the choice among alternate sources of data from which a replica is to be created.

The work of Bent et al. [1] also focuses on the problem of coordination of data movement and computation scheduling in a cluster environment and formulates it as a network flow problem. However, their formulation is based on the assumption that a task accessing multiple files can be split into a set of subtasks accessing a single file each, that can be allocated and scheduled independently. Moreover, their formulation implicitly assumes that all files are of the same size in their network flow formulation. In our work, we allow the flexibility of single/multiple files per data-analysis task and also allow variable file sizes. Kosar et al. [10] propose a specialized scheduler for data placement activities on the Grid. The scheduler allows check-pointing and monitoring of data transfers as well as use of DAG schedulers to encapsulate dependences between computation and data movement. Chakrabarti et al. [5] propose an integrated replication and scheduling strategy that aims at an iterative improvement of performance based on the coupling between scheduling and replication approaches. Mohamed et al. [12] presented a Close-To-Files (CF) job placement algorithm which tries to place jobs on clusters with enough idle processors that are close to the storage sites where the files reside. They integrate this job placement algorithm with file replication.

7 Conclusions and Future Work

This paper presents a novel strategy for simultaneously scheduling a collection of batch-shared data intensive tasks along with replication of files. The proposed approach formulates the problem of coordinating scheduling and replication using a 0-1 Integer programming based approach. Our approach also models the case where the storage space on the compute cluster is limited and may not be sufficient to store all the files required by the batch at once. The performance results obtained on real machines and through simulations show that our strategy achieves significant performance improvement over *MinMin* with *Implicit replication* and *JobDataPresent* with *Data Least Loaded*. The base schemes do not explicitly consider inter-task dependencies arising out of file-sharing and thus make local decisions based on greedy heuristics. Our approach integrates the scheduling and replication problem in the IP formulation based on a global view of the tasks and their file sharing behavior. Our future work will consist of solving the relaxed linear programming variant of the proposed formulation followed by an approximation algorithm to convert the linear solution into an integer solution. We think that such an approach would make the algorithm very efficient in terms of the scheduling overhead.

References

[1] J. Bent, D. Rotem, A. Romosan, and A. Shoshani. Coordination of Data Movement with Computation Scheduling on a Cluster. In *Challenges of Large Applications in Distributed Environments (CLADE2005)*, Research Triangle Park, NC, July 2005.

[2] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve: Open source (Mixed-Integer) Linear Programming system. *Version 5.5.0.0*, May 2005.

[3] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Proc. of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, pages 430–441, 2001.

[4] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363. IEEE Computer Society, 2000.

[5] A. Chakrabarti, R. A. Dheepak, and S. Sengupta. Integration of scheduling and replication in data grids. In L. Bougé and V. K. Prasanna, editors, *HiPC*, volume 3296 of *Lecture Notes in Computer Science*, pages 375–385. Springer, 2004.

[6] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: A high performance remote-sensing database. In *Proc. of the 1997 International Conference on Data Engineering*, pages 375–384. IEEE Computer Society Press, April 1997.

[7] F. Desprez and A. Vernois. Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid. In *CLADE 2005*, Research Triangle Park, NC, July 2005. IEEE Computer Society Press.

[8] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadelphia, PA, March 1989.

[9] G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O. In *Proc. of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, 2005.

[10] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *ICDCS '04: Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 342–349, Washington, DC, USA, 2004. IEEE Computer Society.

[11] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the simgrid simulation framework. In *Proc. of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 138–145, 2003.

[12] H. Mohamed and D. Epema. An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *2004 IEEE International Conference on Cluster Computing*, pages 287–298. IEEE Society Press, 2004.

[13] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proc. of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC)*, Edinburgh, Scotland, July 2002.

[14] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Pipeline and batch sharing in grid workloads. In *Proc. of High-Performance Distributed Computing (HPDC-12)*, pages 152–161, Seattle, Washington, June 2003.

[15] M. Uysal, T. M. Kurc, A. Sussman, and J. Saltz. A performance prediction framework for data intensive applications on large scale parallel machines. In *Proc. of the Fourth Workshop on Languages, Compilers and Run-time Systems for Scalable Computers*, number 1511 in Lecture Notes in Computer Science, pages 243–258. Springer-Verlag, May 1998.