Scheduling of Tasks with Batch-shared I/O on Heterogeneous Systems*

Nagavijayalakshmi Vydyanathan[1], Gaurav Khanna[1], Umit Catalyurek[2,3],
Tahsin Kurc[2], P. Sadayappan[1], Joel Saltz[1,2]

[1] Dept. of Computer Science and Engineering, [2] Dept. of Biomedical Informatics
[3] Dept. of Electrical and Computer Engineering
The Ohio State University, Columbus, OH, 43210
{vydyanat, khannag, saday}@cse.ohio-state.edu, {umit, kurc, jsaltz}@bmi.osu.edu

Abstract

This paper proposes a novel strategy that uses hypergraph partitioning and K-way iterative mapping-refinement heuristics for scheduling a batch of data-intensive tasks with batch-shared I/O behavior on heterogeneous collections of storage and compute clusters. The strategy formulates file sharing among tasks as a hypergraph to minimize the I/O overheads due to duplicate file transfers and employs a K-way iterative mapping-refinement scheme to adapt to the heterogeneity of compute clusters and storage networks in the system. We evaluate the proposed approach through real experiments and simulations on application scenarios from two application domains; satellite data processing and biomedical imaging. Our experimental results show that our approach can achieve significant performance improvement over algorithms such as HPS, Shortest Job First, MinMin, MaxMin and Sufferage for workloads with high degree of shared I/O among tasks.

1 Introduction

Data-driven approaches that make use of large datasets to solve complex problems in science and engineering have become increasingly important. Data analysis is a key component in data-driven science and engineering to gain a better understanding of the problem under study and to more efficiently refine the search space for solutions. Most scientific datasets are stored as a collection of files. In a data analysis application, a data retrieval request specifies a subset of data files from the dataset, which is being analyzed. The set of files are specified as input parameters or are obtained after an index lookup (e.g., the data analysis application may submit a high level query which defines the data of interest on the attributes of the dataset and/or metadata associated with the data files; an index can be used to quickly find the files that can satisfy the query). The data of interest is retrieved from the storage system and processed for analysis.

This paper looks at the problem of scheduling a batch of data analysis tasks with *batch-shared I/O* behavior [27] in a heterogeneous environment. Our goal is to minimize the execution time of the batch. The target environment consists of a heterogeneous collection of compute clusters connected over switched/shared network(s) to one or more storage clusters with different I/O bandwidths. We expect that such configurations will increasingly be common in supercomputing centers as the capacity of commodity disks continues to increase and their cost per gigabyte to decrease.

We propose a two-stage scheduling heuristic based on hypergraph partitioning and a K-way iterative mapping refinement scheme. The proposed heuristic formulates the sharing of files among tasks in the batch as a hypergraph. In the first stage, an initial mapping of tasks to compute nodes is computed without taking the system heterogeneity into account and then this initial mapping is refined using hill-climbing based K-way iterative mapping heuristics that take system heterogeneity into account. In the second phase, tasks that are mapped to compute nodes are ordered in order to minimize end-point contention on the storage cluster. We experimentally evaluate the proposed approach on real platforms and using simulations with application emulators from two application domains; analysis of remotely-sensed data and biomedical imaging. The experimental results show that our approach should be preferred for workloads with high amount of I/O sharing among tasks. For such workloads, significant performance gains are achieved over algorithms such as Shortest Job First, MinMin, MaxMin

*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.

1-4244-0054-6/06/$20.00 ©2006 IEEE

and Sufferage.

2 Related Work

Many techniques have been developed for scheduling in heterogeneous computing systems [1, 8, 14]. Some deal with a single application structured as a DAG, while others apply to globally scheduling many independent tasks. These techniques target compute-intensive tasks with no file sharing. Maheswaran et al. [23] considered three heuristics designed for completely independent tasks (no input file sharing). Casanova et al. [5] modified the MinMin, MaxMin, and Sufferage heuristics to take into account the additional constraint of inter-task file affinities. Their work targets the scheduling of parameter sweep applications in a Grid environment.

The work of Giersch et al. [11] addressed the problem of scheduling a collection of tasks sharing files onto heterogeneous clusters. They proposed extensions to the well-known MinMin heuristic [13] to lower the cost of scheduling while achieving scheduling quality (i.e., batch execution time) similar to that of MinMin. Our work differs from their work in that we investigate whether the quality of scheduling can be improved with the proposed algorithms.

In our recent work [19], we looked at the problem of scheduling tasks exhibiting batch-shared I/O behavior on homogeneous clusters. We modeled the file-sharing in tasks using a hypergraph approach and employed hypergraph partitioning to get a load-balanced cut-minimized partitioning of tasks onto compute nodes. That approach inherently looked at homogeneous platforms. Our current work targets truly heterogeneous environments and uses efficient mapping refinement heuristics to map tasks onto heterogeneous compute clusters. Kaya and Aykanat have concurrently developed an iterative improvement based heuristic for scheduling tasks sharing files on heterogeneous systems [17]. Their work assumes a central master file server, while we target clustered storage systems where multiple files are accessed in parallel.

3 Problem Definition and Use-case Applications

Given a batch of tasks and a set of files required by these tasks, our goal is to schedule the tasks in an efficient manner so as to minimize the batch execution time (makespan). Tasks in a batch may share files, i.e., the set of files required by a task may overlap with the sets of files required by other tasks. Our target hardware platforms are coupled heterogeneous compute and storage clusters. In these settings, data files are distributed across storage clusters. The storage clusters are connected to a heterogeneous collection of compute clusters over switched/shared networks with differing bandwidths. Each compute cluster consists of a homogeneous collection of nodes. Each node in a compute cluster is assumed to have local disks and can request files from any of the storage nodes in the system. The files required by a task are copied from storage nodes to the compute node to which the task has been assigned, before the task is executed.

We have evaluated our approach using application scenarios from two application classes; analysis of remote sensing data and biomedical image analysis. These application scenarios are briefly described below.

Satellite data processing. Remotely sensed data is either continuously acquired or captured on-demand via sensors attached to satellites orbiting the earth [7]. Datasets of remotely sensed data can be organized into multiple files. Each file contains a subset of data elements acquired within a time period and a region of the earth surface. For instance, a dataset in the form of a snapshot of the surface captured by a Landsat thematic mapper satellite consists of N files (usually 4 or 5 files), with each file corresponding to a specific sensor on the satellite and storing data captured by the sensor within the time period and surface region specified by the ground control. 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.

Biomedical Image Analysis. Biomedical imaging is a powerful method for disease diagnosis and for monitoring therapy. State-of-the-art studies make use of large datasets, which consist of time dependent sequences of 2D and 3D images from multiple imaging sessions. Systematic development and assessment of image analysis techniques requires an ability to efficiently invoke candidate image quantification methods on large collections of image data. A researcher may apply several different image analysis methods on image datasets containing thousands of 2D and 3D images to assess ability to predict outcome or effectiveness of a treatment across patient groups.

4 HPS, Shortest Job First, MinMin, MaxMin, and Sufferage

In this work, we compare our proposed approach against our previous work, Hypergraph-based Scheduling Approach (HPS) [19], which was targeting homogeneous systems, and modified MinMin, MaxMin, Sufferage, and Shortest Job First (SJF) heuristics, which take heterogeneity into account. MinMin, MaxMin, SJF, and Sufferage were originally proposed for scheduling independent computational tasks onto compute resources [13]. We employ the algorithms as modified in [5, 4] to take into account the time it takes to transfer input files to compute nodes and files that have already been staged to a compute node in estimating the minimum completion time (MCT) of a task. Though the work of Giersch et al. [11] addresses the problem of scheduling a collection of tasks sharing files onto heterogeneous clusters, their focus is on providing lower cost heuristics that achieve scheduling quality that is reasonably close to the MinMin and Sufferage heuristics [13]. Since our focus in this paper is on improving the quality of the schedule over Min-Min, Sufferage and other existing scheduling heuristics and not on optimizing the scheduling time, we do not compare our schemes with those proposed in [11]. In the rest of the section, we briefly describe the schemes against which we compare the performance of our approach.

Hypergraph Partitioning Based Scheduling (HPS). HPS formulates the sharing of files (batch-shared I/O) among tasks as a hypergraph and clusters the tasks into groups via hypergraph partitioning. Each group is mapped to a compute processor in the system. The scheduling problem is translated into a load-balanced cut minimizing hypergraph partitioning problem. However, HPS formulation does not take heterogeneity into account.

Shortest Job First (SJF). SJF orders tasks in increasing order of their expected execution times. The execution time of a task t_i is calculated as the sum of the time it takes to transfer files needed for t_i (assuming all files have to be transferred from the remote storage) and the execution time for processing the files. The task with the least expected execution time is scheduled on the next processor that becomes idle.

MinMin and MaxMin. This algorithm computes the minimum completion time (MCT) of each task on each node in the system. Among the unscheduled tasks in the batch, MinMin chooses the task with the minimum MCT and assigns it to the node that can execute that task fastest. MaxMin chooses the task with the maximum MCT. When computing the MCT of a task on a node, both strategies take into account the files already available on the node and the files that will be staged onto that compute node by currently running tasks.

Sufferage. The underlying idea is that the system should execute the task that will *suffer* the most if the task is not assigned to the host that will execute the task fastest. The sufferage of a task is computed as the difference between the task's best MCT and its second best MCT. Among unscheduled tasks, Sufferage chooses the task with highest sufferage and assigns it to the node that will achieve the best MCT for the task.

5 A Hypergraph-based Scheduling Heuristic for Heterogeneous Systems (Het-HPS).

We propose a two-stage heuristic for scheduling tasks with batch-shared I/O on heterogeneous systems. The first stage consists of hypergraph partitioning-based mapping of tasks to the compute nodes. The second stage is the ordering of tasks on each compute node. Here we will first present a very brief introduction to hypergraph partitioning and the stages of our scheduling algorithm will follow.

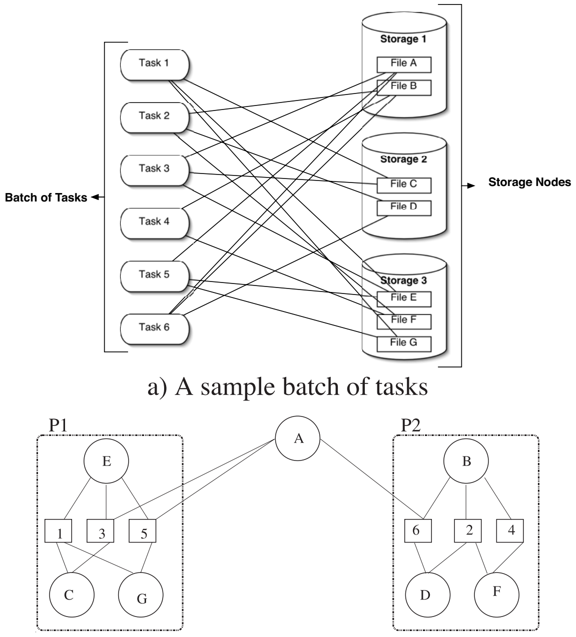
5.1 Hypergraph Partitioning

Hypergraphs are mostly used for VLSI layout placement [22] and modeling the computational structure of parallel applications [6]. Their success in parallel and distributed computing area stems from the fact that they can model asymmetric dependencies and the total volume of communication as a cut metric [6]. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyper-edges) $\mathcal{N}$ among those vertices. Every net $n_j \in \mathcal{N}$ is a subset of vertices, i.e., $n_j \subseteq \mathcal{V}$. The size of a net n_j is equal to the number of vertices it has, i.e., $s_j = |n_j|$. Weights (w_i) and costs (c_j) can be assigned to the vertices ($v_i \in \mathcal{V}$) and edges ($n_j \in \mathcal{N}$) of the hypergraph, respectively. $\mathcal{P} = \{V_1, V_2, \ldots, V_P\}$ is a *P-way partition* of $\mathcal{H}$ if 1) each part is a nonempty subset of $\mathcal{V}$, 2) parts are pairwise disjoint and 3) union of P parts is equal to $\mathcal{V}$. In a partition $\mathcal{P}$ of $\mathcal{H}$, a net n_j is said to be *cut* if it connects more than one parts. The hypergraph partitioning problem can be defined as the task of dividing a hypergraph into two or more parts such that the cutsize is minimized, while a given balance criterion among the part weights is maintained. Algorithms based on the *multi-level* paradigm, such as hMETIS [16] and PaToH [6], have been shown to compute good partitions quickly.

5.2 Task Mapping

Our goal is to find a mapping of tasks to compute nodes such that computational and I/O load of the compute nodes and I/O load of the storage nodes are balanced, and the total communication volume between the storage nodes and compute nodes is minimized. Our solution for this problem is again a two-phase approach. In the first phase, a partitioning of tasks is done by modeling file-sharing interaction as a hypergraph and partitioning is achieved by assuming all the nodes are homogeneous. In the second phase, this initial partition is refined using a K-way mapping heuristic that takes heterogeneity into account. For the first phase, we leverage our previous work [19] on scheduling tasks with batch-shared I/O on homogeneous systems and use a publicly available hypergraph partitioner, namely PaToH [6], to compute the partitioning. For the second phase, we propose a K-way iterative mapping heuristics based on Sanchis [25] multi-way circuit partitioning algorithm.

First Phase: Hypergraph Partitioning. In the hypergraph formulation of a bag-of-tasks, each task t_i is represented by



a) A sample batch of tasks

b) Hypergraph representation

Figure 1. Hypergraph representation of a sample batch of tasks. The numbers indicate tasks. The letters are files required by the tasks.

a vertex v_i in the hypergraph. Each hyper-edge n_j represents a file f_j and connects the vertices that require this file as input. Computation requirement of the task t_i and size of the file f_j are used as weight of the vertex v_i and cost of the net n_j. An example batch of tasks and its hypergraph representation are illustrated in Figure 1.

The estimated execution time of a task on a compute node is calculated as the sum of I/O overhead (the transfer time of files from storage nodes plus the I/O time to read files from local disk) and the computation cost of the task. To employ an existing hypergraph partitioner without any modification, we use a probabilistic approach when computing the execution time $ExecT_i$ of task t_i as vertex weights in the partitioner. Let the set of files a task t_i needs be F_i and the number of compute nodes in the system be K. The cost of transferring one byte of file f_j, Tr_j, for task t_i is equal to

$$Tr_j = \frac{Prob_{FNE}}{BW} + (1 - Prob_{FNE}) * \frac{(1 - Prob_{FE})}{BW} \quad (1)$$

Here, BW is the minimum {I/O,network} bandwidth between any storage and compute node pair, $Prob_{FNE}$ is the probability that task t_i will be the first task to execute in its group that requires f_j, and $Prob_{FE}$ is the probability that t_i executes on a node, to which file f_j has already been transferred. In our current implementation, we assume uniform probability distribution, $Prob_{FNE} = \frac{1}{s_j}$ and $Prob_{FE} = \frac{1}{K}$. s_j denotes the number of tasks that shares the file f_j. With the assumption that computation time is linear with the size of the input files, the estimated execution time of task t_i is computed as

$$ExecT_i = \sum_{f_j \in F_i} filesize(f_j) \times (Tr_j + \frac{1}{LBW} + C) \quad (2)$$

where LBW is the I/O bandwidth from local disk on a compute node and C is the compute cost of one byte [19]. By assigning file sizes as hyper-edge costs and the estimated execution times as vertex weights, the proposed method reduces the task mapping problem to the K-way hypergraph partitioning problem according to the *connectivity-1* cutsize definition [6]. Each and every file needed by a task in the batch will be transfered to the compute system at least once. More specifically, if the tasks that share the file f_j is assigned to λ_j compute nodes, file f_j needs to transfered $\lambda_j - 1$ more times after its first transfer. By using expected execution times as vertex weights, the algorithm aims to balance computational load across the compute nodes.

Second Phase: Refining Initial Partition. The initial partitioning is done assuming a homogeneous system. Hence, it may lead to computational load imbalance and should be refined to account for heterogeneity in the system. We propose a direct K-way mapping refinement heuristic based on Sanchis [25] multi-way circuit partitioning algorithm. Given an initial mapping, the algorithm iteratively refines the mapping by reconsidering the assignment of each of the tasks by tentatively moving them to different parts one by one. There are many different iterative refinement heuristics in the literature, such as Kernighan-Lin-based heuristics [18, 10], Simulated Annealing-based heuristics [20, 2, 12, 15] and Genetic Algorithms-based heuristics [24, 26]. We have chosen Sanchis's Algorithm as our base algorithm because of two reasons. First, Sanchis's algorithm is a generalization of Kernighan-Lin-based algorithms that have been proven to produce very good solutions. Second, it does not require parameter tuning to achieve faster executions.

Algorithm 1 outlines our mapping heuristic. The goal of the algorithm is to minimize the overall execution time. In this work, we modeled the total execution time as the sum of execution time of the maximally loaded compute-node and the I/O time of the maximally loaded storage node. That is

$$ExecutionTime = \max_i Exec(P_i) + \max_p IO(S_p) \quad (3)$$

where $Exec(P_i)$ and $IO(S_p)$ are the execution time of compute node i and I/O time of the storage node p. The algorithm selects a task, from the most heavily loaded part, that will yield the maximum reduction in the above mentioned cost. The amount of the reduction is called the *move-gain* of that task.

Algorithm 1 Direct K-way Mapping Refinement Heuristic

```
Require: $M$: Initial mapping
Ensure: $M$: Final mapping
$BEST \leftarrow ExecutionTime(M)$;
repeat
    unlock all vertices
    let $s$ be the heavily loaded part
    compute $K-1$ move gains of each vertex $v$ in part $s$
    while there exists an unlocked vertex do
        select an unlocked vertex $v$ with max gain $g_{max}$ from $s$ to processor $t$
        tentatively realize the move of vertex $v$; $M[v] \leftarrow t$
        lock vertex $v$;
        update the move gains of unlocked vertices in $s$
        if $ExecutionTime(M) < BEST$ then
            $BEST \leftarrow ExecutionTime(M)$
            permanently realize the moves up to current move
        let $s'$ be the heavily loaded part
        if $s \neq s'$ then
            $s \leftarrow s'$
            recompute $K-1$ move gains of each unlocked vertex $v$ in part $s$
until no more improvements in execution time
```

The execution time of a part and I/O time of a storage node is estimated as follows. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_K\}$ be a K-way partitioning of tasks, where each P_j be the set of tasks allocated to part j. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ be the set of storage nodes. The execution time of part i, $Exec(P_i)$, is the sum of two components; computation and network. The computation component $Comp(P_i)$ represents the aggregate computation weight of the part in terms of the estimated time that would be spent in computation by all the tasks belonging to that part. The network component $Network(P_i)$ represents the total communication weight of that part in terms of the estimated time spent in transferring files to that part. This component is calculated keeping in mind the fact that tasks belonging to a part share files and a particular file needed by multiple tasks needs to be transferred only once for that part. The I/O cost of storage node p, $IO(S_p)$, is the aggregate I/O weight of the storage node in terms of the estimated time that would be spent in I/O for all the files resident on that storage node.

$$Exec(P_i) = Comp(P_i) + Network(P_i) \tag{4}$$

$$Comp(P_i) = \sum_{t_k \in P_i} \sum_{f_t \in F_k} filesize(f_t) \times (\frac{1}{LBW_i} + C_i) \tag{5}$$

$$Network(P_i) = \sum_{f_j \in File_i} filesize(f_j) \times \frac{1}{NBW_{i,M(f_j)}} \tag{6}$$

$$IO(S_p) = \sum_{f_j \in S_p} filesize(f_j) \times \frac{1}{IBW_p} \tag{7}$$

Here, $Files(P_i)$ represents the set of all the files required by the tasks allocated to the part i, $M(f_j)$ represents the storage node that file j is stored, IBW_p represents the I/O bandwidth available at storage node p, and NBW_i, p represents the network bandwidth between the compute node i and the storage node p. These estimates take into account both file affinities and the fact that different compute nodes may have different computing capacities and different network bandwidths with the remote storage nodes. Once the execution time of each part is computed, the part with the highest time is chosen and all the free vertices are considered to move. After each such move, the cost function is recomputed. If the current value is less than the best one so far, all the moves (including the ones with negative gain) are committed. Allowing tentative negative moves allows the algorithm to get out off a local optima. This procedure works in an iterative manner until no improvement in the batch execution time is obtained.

5.3 Ordering of Tasks

Once the tasks are partitioned into groups, the second phase of the scheduling algorithm is to order tasks in each group and to schedule transfer of files from the storage cluster to the compute cluster. Two tasks that are in different groups may have their input files stored on the same set of storage nodes. Thus, ordering of tasks in each group and transfer of files should be done in a way to minimize end-point contention on the storage cluster. We employ a strategy in which tasks within a group are scheduled based on their earliest completion time. The earliest completion time of a task is computed iteratively and dynamically based on the availability of resources.

The algorithm maintains an estimate of the wait times for each of the storage nodes. The wait time of a storage node is the earliest time at which the storage node would become free to service a queued request. When a task in a group is scheduled for execution, the estimated transfer cost of the task from each of the storage nodes is added to the wait times associated with the corresponding storage nodes. In our model, we assume that multiple requests to the same storage node are multiplexed and that a compute node can receive a file after it has finished storing the previously received file on disk.

The earliest estimated completion time for task t_i is computed as the sum of 1) time to stage its input files, 2) time to read the files on local disk, and 3) CPU time to process the files. If all of the input files are already in the compute node, the staging time will be zero. Otherwise, it will be the amount of time spent to transfer the required files from the remote storage system. The staging time is computed as the sum of the actual transfer times (size of the file divided by the storage bandwidth) from each of the the storage nodes and the corresponding wait times at each of those storage nodes.

When a compute node becomes idle, the task with the *earliest expected completion time* in that group is executed.

6 Experimental Results

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

6.1 Application Workloads

The datasets for the satellite data processing application (referred to here as **SAT**) were generated using the application emulator developed in [28]. The SAT application [7] operates on data chunks that are formed by grouping subsets of sensor readings that are close to each other in spatial and temporal dimensions. In our experimental setup, one data chunk is stored in each data file. A data analysis task specifies the data of interest via a spatio-temporal window. For the image analysis application (referred to here as **IMAGE**), we developed 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 metadata describing patient and study related information (in our case, we used patient id and study id as the metadata). 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.

The scheduling strategies were evaluated under 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. The number of queries in each set varies from 50 for smaller workloads to 500 for bigger workloads. Across the sets, there is no overlap between the queries, and in each set, queries are adjusted such that for high overlap, 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.

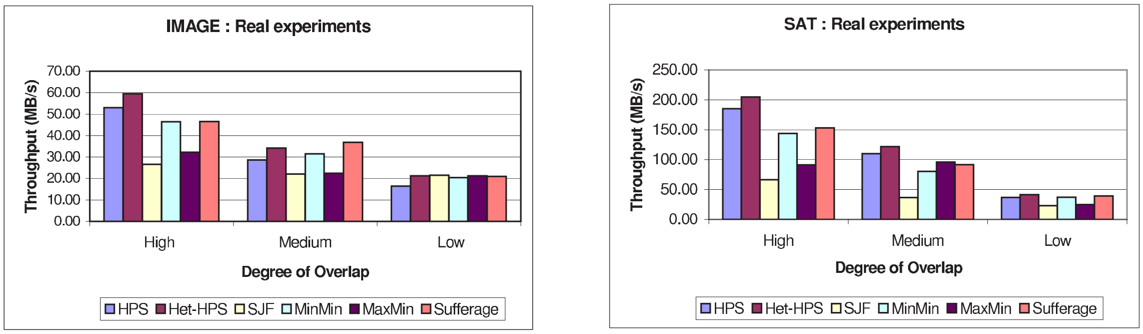
We generated 35 days worth of data, about 162 GB for SAT. The data was distributed across the storage nodes using a Hilbert-curve based declustering method [9]. Each file in the dataset was around 4.5 MB. In the high overlap case, each task accessed on an average 30 files. In the medium and low overlap cases, each task accessed on an average 8 files. For IMAGE, the dataset generated by the emulator corresponded to a dataset of 5000 patients and images acquired over several days from MRI and CT scans. The sizes of images were 1 MB and 16 MB for MRI and CT scans, respectively. The overall size of the dataset was around 330 GB. Images for each patient were distributed among all the storage nodes in a round robin fashion. For both application domains, the number of tasks in a batch varied from 200 tasks for small experiments to 2000 tasks for larger experiments. 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.

6.2 Performance Evaluation on Real Machines

Our experiments were carried out using two compute clusters and a single storage cluster as described below. The first system (**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 second is a 5 node cluster of dual Intel P4 Xeon 2.4 GHz nodes (**DC**). Each node on this cluster has 2 GB of memory and uses switched Gigabit Ethernet for intra-cluster communication. Through micro-benchmarks, we measured each DC node to be about 1.2 times faster than an OSC node[1]. 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 links between the storage cluster OSUMED and the compute clusters OSC and DC to be around 100 Mbps.

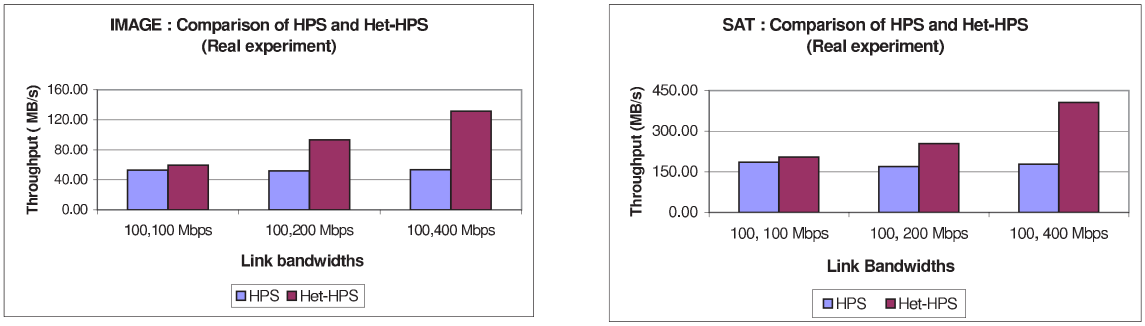
We evaluated the algorithms on configurations with dif-

[1] Even though both systems have same type of CPUs we believe that the difference of the speed comes form hyper threading and possibly from memory bandwidth differences of the motherboards.



(a) (b)

Figure 2. Throughput achieved by different algorithms on 8 OSC nodes and 4 DC nodes (a) IMAGE and (b) SAT.



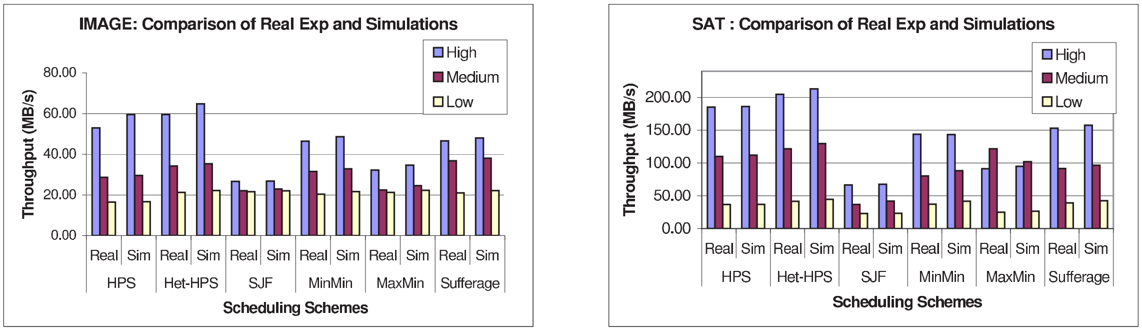
(a) (b)

Figure 3. Performance of HPS and Het-HPS with varying degrees of network heterogeneity (a) IMAGE and (b) SAT.

ferent number of compute nodes in each cluster to capture varying degrees of heterogeneity. Figure 2 shows the relative performance of the various scheduling schemes on workloads with different degrees of shared I/O among tasks, for both application classes. These experiments were conducted using 12 compute nodes (8 OSC and 4 DC nodes) and 6 storage nodes (OSUMED) on the high, medium and low overlap workloads of 200 tasks each. The results show that the Het-HPS strategy performs better than the other algorithms for most cases. This is because the mapping heuristic groups tasks that share files together, thus leveraging data reuse, while adapting to the system and network heterogeneity. The performance improvement due to the mapping heuristic is maximum for the high overlap workload and reduces as the degree of overlap decreases, as expected. Among the base algorithms, Sufferage seems to perform well in most cases. For image analysis workload, SJF seems to perform well for the case of low overlap. This is because, in the image analysis workload, low overlap corresponds to no sharing of files among tasks and hence all schemes transfer the same amount of data from the storage server. In this scenario, SJF achieves maximum load balance among all schemes, since it implicitly balances the load after each task completion.

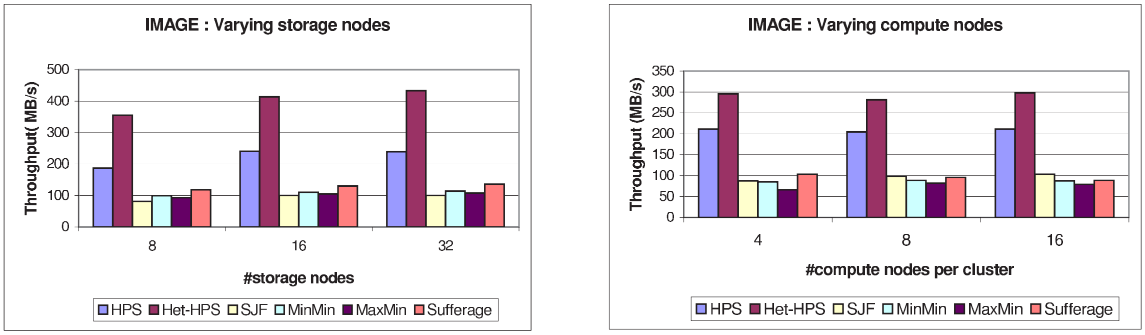
In terms of scheduling time, the Het-HPS algorithm does comparable to the MinMin, MaxMin and Sufferage schemes. Although we have not optimized our implementation of the scheduling algorithms, we observed in our experiments that the scheduling times for all of the algorithms were significantly less than the corresponding batch execution times.

The next set of experiments (Figure 3) is to demonstrate how the Het-HPS approach adapts to varying levels of network heterogeneity. In this experiment we have used 6 storage nodes and 8 compute nodes from OSC and 4 compute nodes from DC cluster. The workload used for these experiments was a 200 task high overlap workload. While we keep the network bandwidth between OSUMED and OSC at 100 Mbps, we have varied network bandwidth between the OSUMED storage nodes and the DC compute nodes from 100 Mbps to 400 Mbps, by transferring proportionally smaller amounts of data to the DC nodes. The results show that the Het-HPS scheme does better than the HPS scheme. The performance benefit of the Het-HPS scheme



(a) (b)

Figure 4. Comparison of real experiment and simulation trends on 8 OSC nodes and 4 DC nodes for high, medium and low degrees of overlap (a) IMAGE and (b) SAT.



(a) (b)

Figure 5. Performance of different algorithms for IMAGE with varying number of (a) storage nodes and (b) compute nodes .

over the HPS scheme increases as the level of network heterogeneity increases. This is expected, since the Het-HPS scheme is able to adapt well to increasing levels of network heterogeneity. For the medium and low overlap workloads, we observed that Het-HPS was able to adapt to the heterogeneity of the system well.

6.3 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, 21]. This toolkit implements event-driven simulation of applications on heterogeneous distributed systems. It models a resource by two performance characteristics: latency (time to access the resource) and service rate (number of work units performed per time unit). It also provides the flexibility of modeling time-shared resources like shared links and different topologies. In our simulations, we used version 2.18.5 of this toolkit. Since Simgrid does not provide an abstraction for disk, we modeled the disk as a time-shared resource with bandwidth equal to disk bandwidth. Each task was modeled as a set of data transfer tasks to stage necessary files from the remote storage, followed by a computation task which simulates processing of the input files.

For the purpose of validating the simulation results, we simulated a hardware configuration similar to the experimental setup for the real experiments. We simulated two clusters, ClusterA and ClusterB. ClusterA simulated the configuration of the OSC cluster and ClusterB simulated the configuration of the DC cluster. Nodes within each cluster are homogeneous in terms of processing capability and local disk bandwidth. The networks between compute clusters (ClusterA and ClusterB) and the storage nodes is simulated as two separate 100 Mbps links. The heterogeneity in the network comes from different number of nodes in each of the clusters which means that the bandwidth seen by a node of ClusterA and a node of ClusterB differ. This is because all the nodes of a compute cluster share the link to the storage cluster and thus, in the worst case, the bandwidth is shared by all of them. Nodes in ClusterB are 1.2 times faster in processing capability than those in ClusterA. Figure 4 shows the comparison between the real experiments and the simulated results for both 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.

To analyze the performance of our scheduling strategy with respect to the varying number of storage and compute nodes in the system, we ran simulations of high overlap workloads of 2000 IMAGE tasks using a 4 compute cluster configuration, and the results are presented in Figure 5. The network bandwidth between the compute clusters and the storage cluster was simulated to be in the ratio 1:4 for the compute cluster with the slowest network to the compute cluster with the fastest network. The simulated network bandwidth values varied from 12.5 MB/sec to 50 MB/sec. The disk bandwidth in these simulations was taken to be as 40 MB/sec. The number of compute nodes in each cluster were taken to be as 4. Figure 5(a) shows the performance of the various scheduling algorithms as the number of storage nodes in the system are scaled. The results show that as the number of storage nodes increase, the performance of all the algorithms improves only slightly. The reason is that in these simulations, the network is the bottleneck since, even the fastest network bandwidth of 50 MB/sec between one of the compute clusters and the storage clusters is shared among 4 compute nodes. Thus, increasing the number of storage nodes does not quite yield the benefit of distributing the data across more storage nodes. The results however, do show that the Het-HPS scheme performs significantly better than all the other schemes as the number of storage nodes in the system increase. Figure 5(b) shows the simulation results while varying the number of compute nodes to 4, 8 and 16 in each cluster. The number of storage nodes in these simulations was 6. The Het-HPS algorithm gives roughly 280% improvement over the base algorithms (SJF, MinMin, MaxMin and Sufferage) and 40% over the HPS algorithm. The results show that the throughput values do not scale well as the number of compute nodes per cluster increases, because there is increasing degree of contention on the shared link between the compute cluster and the storage cluster. We also ran both real experiments and simulations for the 2-cluster configuration (OSC and DC) for validation, by varying the number of OSC compute nodes from 4 to 16 and keeping the number of compute nodes of DC fixed at 4. Figure 6 shows the comparison between the real experiments and the simulated results for IMAGE. We see that the relative trends of the simulated results closely follow those of the real experiments.

7 Conclusions and Future Work

This paper presents a novel strategy for scheduling a collection of data intensive tasks with batch-shared I/O on heterogeneous systems. The performance results obtained on real machines and through simulations show that our strategy achieves significant performance improvement over HPS, SJF, MinMin, MaxMin and Sufferage. The base schemes like MinMin and Sufferage look at each task-host pair in isolation for making scheduling decisions and do not explicitly consider inter-task dependencies arising out of file-sharing. Our proposed approach, on the other hand, maps tasks to processors based on a global view of the tasks and their file sharing behavior. In comparison to our earlier work HPS, HPS only looks at task-file affinities without taking into account any system heterogeneity whereas our new approach Het-HPS models the system heterogeneity, resulting in significantly better schedules on systems with diverse resources.

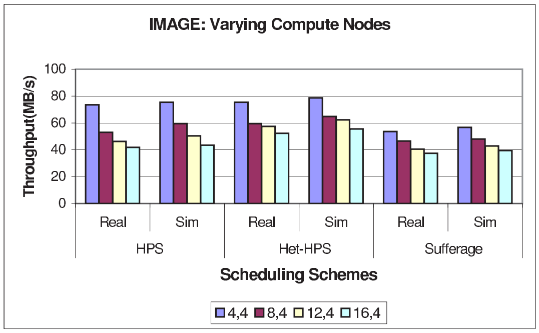


Figure 6. Comparison of real experiments and simulations for different algorithms with varying number of compute nodes for IMAGE.

References

[1] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison study of static mapping heuristics for a class of meta- tasks on heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[2] T. N. Bui, C. Heigham, C. Jones, and F. T. Leighton. Improving the performance of the Kernighan-Lin and simulated annealing graph bisection algorithms. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 775–778, 1989.

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

[4] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *Proceedings of the 2000 ACM/IEEE SC00 Conference*, pages 75–76, 2000.

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

[6] U. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

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

[8] M. M. Eshaghian and Y. C. Wu. Mapping heterogeneous task graphs ontp heterogeneous system graphs. In *Proceedings of the 6th Heterogeneous Computing Workshop*, pages 147–160, Geneva, Switzerland, April 1997. IEEE Computer Society Press.

[9] 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.

[10] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, Las Vegas, Nevada, June 1982.

[11] A. Giersch, Y. Robert, and F. Vivien. Scheduling tasks sharing files from distributed repositories. In *Euro-Par 2004: Parallel Processing: 10th International Euro-Par Conference, volume 3149 of Lecture Notes in Computer Science*, pages 246–253, Sept. 2004.

[12] S. M. Hart and C. L. S. Chen. Simulated annealing and the mapping problem: A computational study. *Computers Operations Research*, 21(4):455–461, 1994.

[13] O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on nonindentical processors. *Journal of the ACM*, 24(2):280–289, Apr 1977.

[14] M. Iverson and F. Ozguner. Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment. In *Proceedings of the 7th Heterogeneous Computing Workshop*, Orlando, FL, March 1998. IEEE Computer Society Press.

[15] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, November 1989.

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain. In *34th Design Automation Conference*, Anaheim, CA, June 1997.

[17] K. Kaya and C. Aykanat. Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments. *IEEE Transactions on Parallel and Distributed Systems*, 2006. accepted subject to minor revision.

[18] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970.

[19] 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 *Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CC-Grid 2005)*, 2005.

[20] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

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

[22] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Willey–Teubner, Chichester, U.K., 1990.

[23] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop (HCW'99)*, pages 30–44, Apr. 1999.

[24] H. Maini, K. Mehrotra, C. Mohan, and S. Ranka. Genetic algorithms for graph partitioning and incremental graph partitioning. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 449–457, New York, NY, USA, 1994. ACM Press.

[25] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.

[26] E.-G. Talbi and P. Bessiere. A parallel genetic algorithm for the graph partitioning problem. In *ICS '91: Proceedings of the 5th international conference on Supercomputing*, pages 312–320, New York, NY, USA, 1991. ACM Press.

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

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

Biographies

Nagavijayalakshmi Vydyanathan is a PhD student in the Department of Computer Science and Engineering at the Ohio State University. She received her BE in Electronics and Instrumentation and MSc(Tech) in Information Systems from the Birla Institute of Technology and Sciences, India in 2001. Her research interests are in scheduling and resource management in parallel and distributed system with focus on data-intensive applications.

Gaurav Khanna is a PhD student in the Department of Computer Science and Engineering at the Ohio State University. He received his BE in Computer Science and Engineering from the University of Delhi, India in 2002. His research interests include Job scheduling in the context of I/O intensive applications, Performance Modeling and Parallel I/O systems.

Umit Catalyurek is an Assistant Professor in the Department of Biomedical Informatics at The Ohio State University. His research interests include graph and hypergraph partitioning algorithms, grid computing, and runtime systems and algorithms for high-performance and data-intensive computing. He received his PhD, M.S. and B.S. in Computer Engineering and Information Science from Bilkent University, Turkey, in 2000, 1994 and 1992, respectively.

Tahsin Kurc is an Assistant Professor in the Department of Biomedical Informatics at the Ohio State University. His research interests include runtime systems for data-intensive computing in parallel and distributed environments, and scientific visualization on parallel computers. He received his PhD in computer science from Bilkent University, Turkey, in 1997 and his B.S. in electrical and electronics engineering from Middle East Technical University, Turkey, in 1989.

P. Sadayappan is a Professor in the Department of Computer Science and Engineering at the Ohio State University, Columbus. His research interests include compile/runtime optimization and scheduling and resource management for parallel and distributed systems. He received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, in 1977 and the MS and PhD degrees in electrical engineering from the State University of New York, Stony Brook, in 1978 and 1983 respectively.

Joel Saltz is Professor and Chair of the Department of Biomedical Informatics, Professor in the Department of Computer and Information Systems and a Senior Fellow of the Ohio Supercomputer Center. He received his M.D. and PhD in computer science from Duke University in 1985 and 1986, respectively. He earned his B.S. in mathematics and physics from University of Michigan in 1978. His research interests are in the development of systems software, databases and compilers for the management, processing and exploration of very large datasets.