A Hypergraph Partitioning Based Approach for Scheduling of Tasks with Batch-shared I/O*

Gaurav Khanna†, Nagavijayalakshmi Vydyanathan†, Tahsin Kurc‡,
Umit Catalyurek‡, Pete Wyckoff+, Joel Saltz‡, P. Sadayappan†

† Department of Computer Science and Engineering
‡ Department of Biomedical Informatics
The Ohio State University
+ Ohio Supercomputer Center

Abstract

Data analysis applications are increasingly becoming a key part of data-driven science and application systems. In this paper, we examine strategies for scheduling of multiple data analysis tasks with batch-shared I/O behavior (i.e., tasks access overlapping sets of files stored in a storage system). We propose a novel, hypergraph partitioning based strategy. This strategy formulates the sharing of files among tasks as a hypergraph to minimize the I/O overheads due to transferring of the same set of files multiple times and employs a dynamic scheme for file transfers to reduce end-point contention on the storage system. We experimentally evaluate the proposed approach using application emulators from two application domains; analysis of remotely-sensed data and biomedical imaging.

1 Introduction

The development of new technologies in several areas is making it more feasible to take a data-driven approach to address complex problems in science and engineering. First of all, our ability to collect data has increased tremendously with the help of advanced sensors that can rapidly capture data at high-resolutions and Grid technologies that enable simulation of complex numerical models. Moreover, inexpensive platforms for large scale, disk-based storage are becoming increasingly available. It is currently possible to implement disk-based storage systems with 100+ TB storage space. An example is the Ohio Supercomputer Center, which currently has a 500TB disk-based storage system. We anticipate that in the not-too-distant future, supercomputing centers will not only provide computational resources, but also serve as high-end data centers for scientific and engineering applications.

*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.

Several research groups have targeted applications that generate, collect, and reference large datasets in biomedicine [4, 52, 38]. An increasing number of research projects also are developing techniques and tools for data-driven science in application areas ranging from dynamic numerical modeling to data assimilation in measured data to design optimization [17, 22, 21, 48, 45]. The ultimate goal in collecting large volumes of data is to gain a better understanding of the problem under study and to more efficiently refine the search space for solutions. Hence, *data analysis applications* are a key part of data-driven science and application systems. In the context of data-driven science, a data analysis application can be defined as one that accesses and processes a subset of a dataset. Most scientific datasets are stored in files. A request for the region of interest specifies a subset of data files and/or segments in data files – either as part of the input parameters or after an index lookup, which finds the files and file segments that can address the request. The data of interest is then processed and transformed into a data product, which is more suitable for examination by the scientist.

Several approaches have been proposed to achieve high data rates in data intensive applications through parallel I/O, caching, etc. [3, 16, 7, 30, 40, 55, 37, 42, 47, 49, 55, 53]. Shen and Choudhary [49], for example, present a Multi-Storage I/O System (MS-I/O) that manages various distributed storage resources in the system and provides high performance storage access schemes. MS-I/O employs many state-of-the-art I/O optimizations such as collective I/O, asynchronous I/O and a number of new techniques such as data location, data replication, subfile, superfile and data access history. Scheduling is another optimization approach that can improve performance, when multiple tasks are submitted to the system.

In this paper we address the problem of scheduling a batch of data analysis tasks submitted to a server cluster so that the total execution time of the batch is minimized. The need to handle batches of such jobs can arise in several situations. In a server accessed by many clients concurrently, there may be multiple jobs waiting in the job queue for execution, because of limited resources on the server. A client may submit multiple jobs in a batch to potentially analyze subsets of a dataset with different input parameters. For example, applications that search a parameter space to optimize one or more objective functions may submit multiple jobs, each with different parameter values, against a group of datasets. Traditional job scheduling techniques are designed to address the requirements of batches consisting of compute intensive jobs. As more data driven applications become candidates for consuming storage and compute resources, scheduling of data analysis jobs will have an increasing importance.

Thain et.al. [54] characterize types of I/O sharing in data intensive application batches; *pipeline-shared*, in which files are shared in write-then-read fashion in a single pipeline of data processing jobs, and *batch-shared*, in which files are shared across tasks in different pipelines. Batch-shared I/O can be attributed to several factors. If jobs are submitted by clients working in the same application domain, there may be a number of overlapping regions of interest, or ”hot spots”, as scientists in the same domain are likely to have similar interests. Sharing of I/O also depends on how data is distributed across data files in the system. Requests by two jobs may not overlap in the underlying attribute space of the dataset, but data elements required to serve those requests might have been stored in the same set of files.

In earlier work [51, 59] we examined algorithms for scheduling pipelines of data processing with *pipeline-shared* I/O behavior. This paper focuses on scheduling of tasks with *batch-shared* I/O behavior. We propose a novel, hypergraph based approach. Hypergraphs have attracted much at-

tention for modeling the computational structure of many parallel applications [9, 10, 13, 23]. The main advantages of the hypergraph model are that a hypergraph can model asymmetric dependencies and the cut metric is well suited for minimizing the total volume of communication [10, 23]. The approach proposed in this paper formulates the sharing of files (batch-shared I/O) among tasks as a hypergraph and employs a two-stage strategy for scheduling of tasks and file transfers. In the first stage, tasks are partitioned into groups via hypergraph partitioning. Each group is mapped to a compute processor in the system. In the second stage, a dynamic strategy is applied to order tasks in each group for execution and to transfer files from storage system to compute nodes for task execution. We experimentally evaluate the proposed approach using application emulators from two application domains; analysis of remotely-sensed data and biomedical imaging.

2 Related Work

Most of classic work on scheduling for parallel machines is derived from Sarkar [46], and later work such as Yang and Gerasoulis [61]. The goal on distributed memory parallel machines is to trade-off parallelism with communication. Most techniques for scheduling are either based on clustering [18], list-scheduling [32], or combination of both. Several techniques have also been developed for scheduling in heterogeneous computing systems [18, 56, 60, 36, 6, 25]. Some of these techniques deal with a single application structured as a DAG, while others apply to globally scheduling many independent tasks. In our work we target data intensive jobs.

Relatively little research so far has addressed the scheduling of data intensive jobs. In [43], a decoupled approach to scheduling of computations and data for data-intensive applications was proposed, and evaluated using a simulation testbed. However, a simple first-come first-served scheduling strategy was used in that study. Another study that explicitly modeled the cost of data movement in job scheduling was for parameter sweep applications [8]. A parameter sweep application consists of a set of independent jobs that should be scheduled to the resources in the environment. The authors modify several heuristics that were designed for compute intensive applications on parallel machines [24, 35] for parameter sweep applications in a Grid environment. The modified heuristics take into account data transfer times and affinity of tasks to machines. Our approach and target platform are different. We look at scheduling on coupled storage and compute clusters and propose a hypergraph based approach to model batch-shared I/O. Spencer et.al. [51] developed two algorithms for scheduling multiple pipelined operations in heterogeneous environments. However, that work targets applications that exchanged data buffers via TCP/IP and memory copy operations.

Multi-query workloads also arise in the context of database applications [15, 44, 12, 62]. The work of Mehta et al. [39] is one of the first to address the problem of scheduling queries in a parallel database by considering batches of queries. The authors target sharing data structures for joins and selects, and the goal of their scheduling algorithms is to find the global schedule for all queries that minimizes the total execution time of a given batch. In [2], Andrade et.al. propose a dynamic scheduling model for multi-query workloads in data analysis applications. The goal is to maximize data and computation reuse and concurrent execution on SMP nodes through semantic caching and ordering of queries. The model is based on a priority queue implementation using a directed graph and a strategy for ranking queries. The directed graph is used to express commonalities and dependencies among queries in a query batch. We too aim to minimize I/O, networking, and computation overheads by taking into account overlaps among tasks, like the previous work in

multi-query scheduling. However, previous strategies mainly target efficient reuse of results from previously executed queries.

Chang et.al. [13] examine optimization methods for executing data aggregation operations on disk-resident datasets on distributed-memory machines with local disk farm. Drawing from the *computational hypergraph* model proposed in [9, 10] for decomposition of sparse matrices in parallel matrix-vector multiplication operations, the authors propose a hypergraph based algorithm for partitioning of workload among processors and for scheduling of processing. The algorithms model retrieval, communication, and processing requirements of parallel aggregation as a weighted hypergraph. An execution plan is generated by computing a multi-way partitioning of the hypergraph. Our approach is targeted towards scheduling of multiple tasks, rather than efficient parallel execution of a single task.

Jain et.al. [26] model scheduling of I/O operations (with certain assumptions) as a bipartite graph with two separate sets of nodes namely, disks and processors. The algorithm proceeds by coloring such a graph where k instances of a single color in the graph represent k simultaneous I/O operations. The goal is to color the graph with minimum number of colors so as to minimize the length of the schedule. Our main difference is we consider grouping and mapping of tasks to compute nodes in tandem with ordering of tasks and scheduling of remote I/O operations for file transfers.

3 Applications and Problem Definition

In this section, we present a brief overview of two applications from the application domains we used to evaluate the scheduling strategies and the definition of the problem we are targeting to address in this paper.

3.1 Applications

Satellite data processing. Remotely sensed data is an invaluable source of information for earth scientists studying Earth's atmosphere, land, and sea properties and conditions. This kind of data is either continuously acquired or captured on-demand via sensors attached to satellites orbiting the earth. In most remote sensing applications, a data element acquired by a sensor reading is associated with a location on earth, the time of recording, and the type of sensor. Datasets of remotely sensed data can be organized into multiple files in various ways. A common characteristic of different file organizations is that 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. A typical query [1, 14, 34, 50] involves processing of the data within a time period and region, and generating one or more composite images of the area under study. That is, a data analysis task can request a subset of the data, which is defined by a spatio-temporal window. Such a request results in retrieval of files (or segments of files), the spatio-temporal bounding box of which intersects the query window. The spatio-temporal bounding box of a file is defined by the maximum and minimum spatial coordinates and recording times of all of the sensor readings stored in that file. The composite image is generated by processing the sensor readings and combining the sensor values that map to

the same output image pixels [57]. 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 (e.g., cancer) diagnosis and for monitoring therapy. Imaging studies such as Dynamic Contrast Enhanced MRI [28, 29, 41] capture time-dependent sequences of 2D and 3D images. In DCE-MRI, after an extracellular contrast agent is injected in the subject, a sequence of images are obtained over some period of time. The time-dependent signal changes acquired in these images are quantified by a pharmacokinetic model to map out the differences between tissues. State-of-the-art research studies in DCE-MRI make use of large datasets, which consist of time dependent, multidimensional collections of data from multiple imaging sessions. Although single images are relatively small (2D images consists of 128^2 to 512^2 pixels, 3D volumes of 64 to 512 2D images), a single study carried out on a patient can result in an ensemble of hundreds of 3D image datasets. A study that involves time dependent studies from different patients can involve thousands of images. 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 data from multiple different studies to assess ability to predict outcome or effectiveness of a treatment across patient groups. Such a study would be likely to involve image datasets containing thousands of 2D and 3D images and multiple image analysis jobs with different input parameters.

3.2 Problem Definition

In scientific and engineering applications, a dataset is stored as a set of *data files*, each consisting of a disjoint subset of data items. Data files are declustered across storage units (or file systems) using a declustering algorithm, in order to better utilize the aggregate storage space and I/O bandwidth.

In this paper, we target configurations consisting of coupled compute and storage platforms. Datasets are stored on a pool of storage nodes (storage cluster). Storage nodes are connected to a pool of compute nodes (compute cluster) over a local area network or switch. Each compute node has one or more local disks and can request files from any of the storage nodes. Such configurations are likely to be common in institutions as well as supercomputer centers, since compute and storage clusters are designed with different goals in mind. A compute cluster will have high-end processors with high-speed networking among them. On the other hand, a storage cluster may forgo computing power in favor of large storage space.

A batch consists of independent sequential tasks (data analysis 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. If a file is required for processing by one or more tasks, it may be retrieved multiple times as a whole and transferred to the respective compute nodes.

Our objective is, given a batch of tasks and a set of files required by these tasks, to schedule the tasks in an efficient manner 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 compute weight, list of input files, and their file sizes. The tasks in the batch may share a number of files.

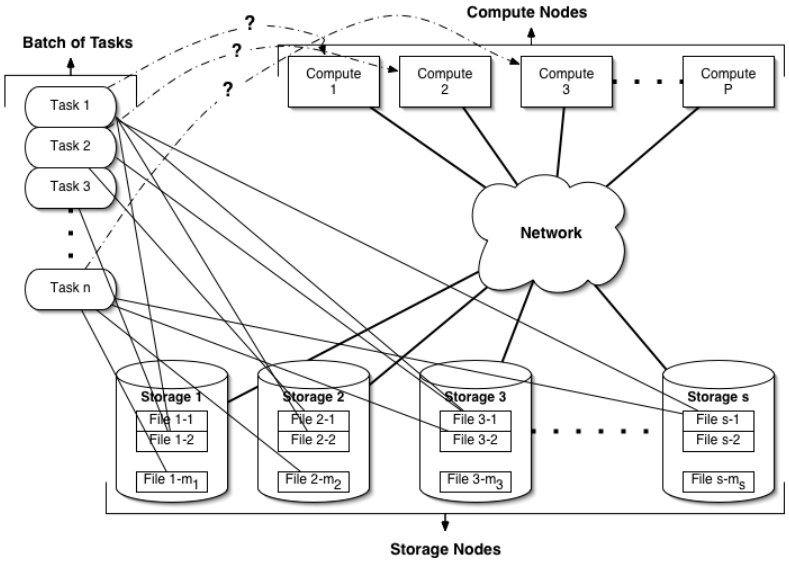


Figure 1: Scheduling problem.

4 Task Scheduling Strategies

In this paper, we examine the MinMin, MaxMin, Sufferage, which are originally proposed for scheduling independent computational tasks to compute resources [24, 35], along with Shortest Job First heuristics and our proposed algorithm to determine task-compute node assignments. As in [8], we modify the MinMin, MaxMin, and Sufferage to take into account 1) the time it takes to transfer input and output files to and from compute nodes in the environment and 2) files that have already been staged to a compute node in estimating the minimum completion time of a task.

4.1 Shortest Job First, MinMin, MaxMin, and Sufferage

Shortest Job First (SJF). Tasks are ordered for execution based on 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 and the execution time for processing the files. In the SJF strategy, the shorter the execution time of a task is, the earlier the task is executed.

MinMin. Given a set of tasks that have not yet been scheduled, this strategy computes the minimum expected completion time of each task on each node in the system. When computing the completion time for a task on a node, it takes into account, the files, required by the task, that is already transferred to that node by tasks previously executed on that node. Among the unscheduled tasks in the batch, MinMin chooses the task that can complete the earliest and assigns it to the node that can execute that task fastest.

MaxMin. As in MinMin, the MaxMin strategy computes the estimated minimum completion time (MCT) of a task on each node in the system. Among the unscheduled tasks, it chooses the task with the maximum MCT.

Sufferage. The Sufferage strategy looks at how much a task will suffer if it is not assigned to the

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

In these schemes, once a task is mapped to a compute node, files required for the task are staged from the storage system to the compute node independent of staging of files for tasks that are concurrently mapped to other compute nodes.

4.2 A Hypergraph-based Approach

We propose a two-stage scheduling algorithm. In the first stage, we partition and map the tasks to the compute nodes. In the second stage, ordering of the tasks in each compute node is determined. Here, we will describe these stages in more detail. We first present a brief overview of hypergraph partitioning.

4.2.1 Hypergraph Partitioning

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 $\mathcal{V}$. A partition is said to be balanced if $W_p \leq W_{avg}(1+\varepsilon)$ for $1 \leq p \leq P$, where $W_p = \sum_{v_i \in V_p} w_i$ is the sum of the vertex weights of part V_p, $W_{avg} = (\sum_{v_i \in \mathcal{V}} w_i)/P$ denotes the weight of each part under the perfect load balance condition, and ε represents the predetermined maximum imbalance ratio allowed.

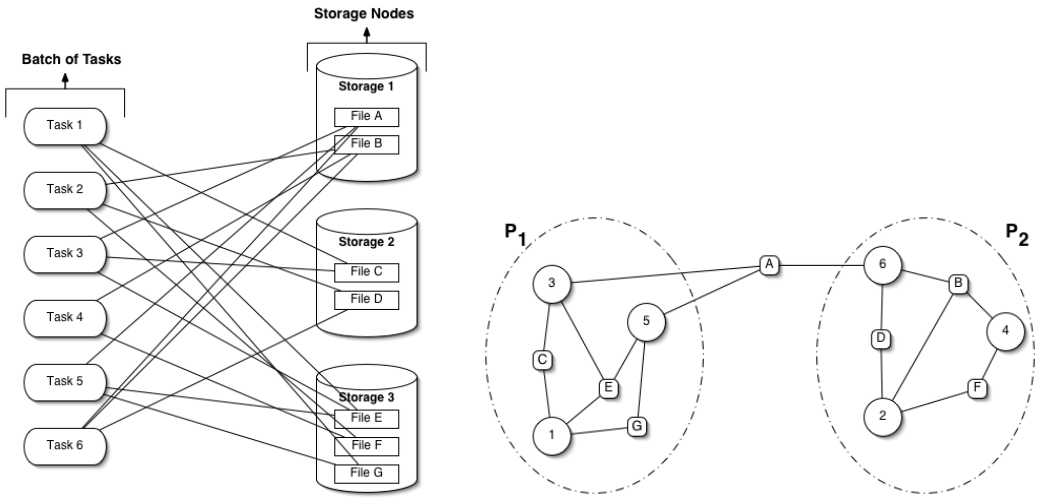
In a partition $\mathcal{P}$ of $\mathcal{H}$, a net that has at least one vertex in a part is said to *connect* that part. *Connectivity* λ_j of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* if it connects more than one part (i.e. $\lambda_j > 1$). The cut nets are also referred to here as *external* nets and is denoted as $\mathcal{N}_E$. There are various *cutsize* definitions for representing the cost $\chi(\mathcal{P})$ of a partition $\mathcal{P}$ [33]. The relevant, *connectivity-1*, definition is:

$$\chi(\Pi) \quad = \quad \sum_{n_j \in \mathcal{N}_E} c_j(\lambda_j - 1). \tag{1}$$

In (1), each cut net n_j contributes $c_j(\lambda_j - 1)$ to the cutsize. Hence, the hypergraph partitioning problem [33] 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 [27] and PaToH [10], have been shown to compute good partitions quickly for this NP-hard problem [20, 33].

4.2.2 Hypergraph Formulation for Partitioning and Mapping of Tasks

If two tasks that share a lot of files are executed concurrently on two different processors, it will result in transfer of the same files multiple times from the storage cluster to the compute cluster.



(a) (b)

Figure 2: a) A sample batch of tasks with their file dependencies, b) Hypergraph representation of the batch of the tasks displayed in (a).

On the other hand, if the tasks are scheduled to the same compute node in order to minimize file transfer times, load imbalance may be incurred with some computational nodes having more task load than others. Our goal is to partition tasks into groups such that the amount of data transfer between the storage cluster and the compute cluster is minimized while load balance across compute nodes is maintained. When scheduling a task, the MinMin, MaxMin, and Sufferage strategies consider the affinity of a task to compute nodes based on files that have already been transferred to the compute nodes for other tasks. This implements a simple way to minimize the number of times the same file is transferred from storage cluster. However, these approaches only consider sharing of files between the task to be scheduled and the tasks that have already been scheduled and executed.

We propose a hypergraph formulation to model sharing of files among tasks and a hypergraph partitioning based approach to compute a partitioning and mapping of tasks to compute nodes. Our hypergraph model represents each task t_i by a vertex v_i in the hypergraph. Each hyper-edge n_j represents a file f_j and connects the vertices (tasks) that require this file as input. This hypergraph is partitioned into P groups, where P is the number of compute nodes, and each group is mapped to a compute node. The partitioning is done so that the compute and I/O weight of the clusters are balanced and the cost of transferring shared files across clusters is minimized. Figure 2 illustrates a sample batch of tasks and a partitioning of the hypergraph representation of them.

The weight w_i of a vertex v_i is equal to the estimated execution time of the corresponding task. As in the SJF, MinMin, and Sufferage strategies, the estimated execution time of a task 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. However, in the previous strategies, a task's estimated execution time is dynamically updated, as files are staged to compute nodes for other tasks. That is when a task is considered for execution, its execution time is estimated by taking into account the set of files that have already been transferred to the compute cluster and their distribution across compute nodes. Similarly, the weight of a vertex in the hypergraph will depend on which group the corresponding task is assigned and the order in which the tasks in

that group are executed. Our current hypergraph partitioning strategy assumes static edge costs and vertex weights. Moreover, the hypergraph based strategy globally partitions all the tasks in a given batch into groups before any order for task execution is determined for a group. In order to alleviate these issues and provide a better estimate of the execution time of a task, we compute the weight of a vertex as follows.

Let the set of files a task t_i needs be F_i and the number of compute nodes in the system be P. The cost of transferring file f_j, $Transfer_j$, for task t_i is equal to

$$\begin{aligned} Transfer_j(one\_byte) &= \frac{Prob_{first\_task}}{RemoteBW} + \\ & (1 - Prob_{first\_task}) * \frac{(1 - Prob_{mapped\_to\_the\_same\_node})}{RemoteBW} \end{aligned} \quad (2)$$

Here, $RemoteBW$ is the I/O bandwidth between a storage node and a compute node, $Prob_{first\_task}$ is the probability that task t_i will be the first task to execute in its group that requires f_j, and $Prob_{mapped\_to\_the\_same\_node}$ 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. Hence, we have used $Prob_{first\_task} = \frac{1}{s_j}$ and $Prob_{mapped\_to\_the\_same\_node} = \frac{1}{P}$. Recall that s_j is the size of the hyper-edge n_j that represents file f_j. Hence it also 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

$$TimeExecution_i = \sum_{f_j \in F_i} filesize(f_j) \times (Transfer_j + \frac{1}{LocalBW} + Compute_{byte}) \quad (3)$$

where $LocalBW$ is the I/O bandwidth from local disk on a compute node. By assigning the files sizes as hyper-edge costs, the proposed method reduces the task mapping problem to the P-way hypergraph partitioning problem according to the cutsize definition given in (1). Each and every file needed by a task in the batch will be transfered to the compute system at least once. By using connectivity-1 metric as our cost function, extra transfers of the files due to task assignment is minimized. 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.

4.2.3 Ordering of Tasks in a Group and Transfer of Files

Once the tasks are partitioned into groups, the second phase of the scheduling algorithm is to order tasks in each group and schedule transfer of files from storage cluster to compute cluster. By minimizing the cutsize, the hypergraph based grouping of tasks will reduce the number of times the same file is retrieved. However, hypergraph partitioning alone will not be sufficient to minimize the I/O overhead. Our current hypergraph model accounts for sharing of files among tasks, but it does not take into account *end-point* contention on storage nodes explicitly. Even two tasks that do not share files may have their input files stored on the same set of 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 a *Gantt chart* for storage nodes. When a task in a group is scheduled for execution, time slots are reserved on storage nodes for file transfers required for this task. These time slots for a task are marked on the Gantt chart. In calculating the duration of time slots and marking them on the Gantt chart, we assume that a compute node and a storage node can take part in at most one file transfer at a time. That is, multiple requests to the same storage node is serialized and the compute node can receive a file after it has finished storing the previously received file on local disk.

The earliest completion time of a task t_i is estimated as the sum of time to stage its input files F_i and its execution time. The staging time is the time spent to make the input files ready in the compute node. 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 last input file from the storage node. The transfer completion time for each file $f_j \in F_i$ (TCT_j) is estimated as the sum of the earliest time a transfer can start (first available slot in the Gantt chart after the time that the compute node becomes available) and the actual transfer time (size of f_j divided by the storage bandwidth; computed as the minimum of remote disk bandwidth and network bandwidth). The file f_j with the minimum TCT_j is picked and tentatively scheduled for transfer. TCTs of the rest of the input files are recomputed and the next file with the minimum TCT is picked and tentatively scheduled. This process is repeated until all of the input files are scheduled. TCT of the last file scheduled actually gives the staging time. Then the earliest estimated completion time for t_i is computed as the sum of 1) the completion time of file transfers from storage nodes, 2) I/O time to read the files on local disk, and 3) CPU time to process the files. The scheduling algorithm determines the task with the least completion time in each group, and the task t_i with the lowest *earliest completion time* out of these is scheduled first. Once t_i is scheduled, out of the other task groups (excluding the one containing t_i), the task with the minimum earliest completion time (taking into account the current reservations) is now picked and scheduled. When a running task completes, the task with earliest completion time from that group is scheduled.

Figure 3 illustrates the execution of the ordering algorithm on the batch of tasks shown in Figure 2. In this figure transfer of each file takes 1 unit of time, and processing of a file takes 0.5 units. Since task 4 depends on two files, its earliest completion time is 3. Hence it has been scheduled first and 1 unit of time on storage node 1 and 1 unit of time on storage node 3 have been reserved. Since a task has been scheduled from group 2, next the task with the earliest completion time from group 1 is scheduled. Since all of the tasks in the group depends on 3 files, and they can be scheduled to transfer all of the files in 3 units, we pick one of them, say task 1. The algorithm continues by reserving the transfer of files for task 1, and another task from group 2 is picked.

5 Experimental Results

We experimentally evaluated the scheduling algorithms using two application classes, satellite data processing and biomedical image analysis, described in Section 3. We employed application emulators [58] to generate various application scenarios. An application emulator provides a

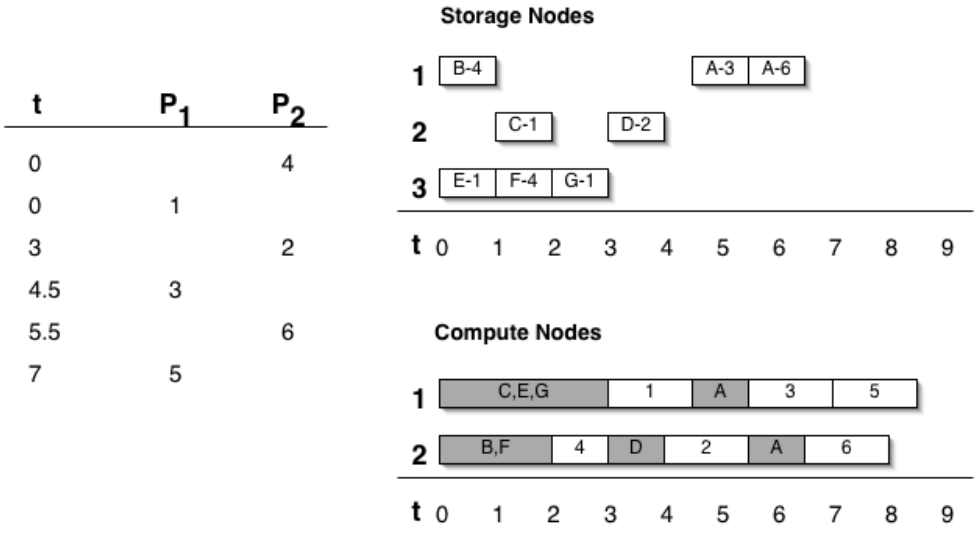


Figure 3: An illustration of the execution of the ordering algorithm on the batch of tasks shown in Figure 2.

parameterized model of an application class; adjusting the parameter values makes it possible to generate different application scenarios within the application class in a controlled way. We used the PaToH toolkit [10, 11] to obtain good quality partitionings of the hypergraphs generated for the workloads in the experiments. In our experiments, we observed that the hypergraph partitioning overhead is minimal compared to the execution time of a batch.

5.1 Hardware Configuration, Application Emulators, and Workloads

The experiments were carried out on two systems. The first system is a cluster of Pentium III 933 MHz nodes (**OSUMED**). Each node of this cluster has 300GB disk space and 512MB of memory. The nodes are connected through a Switched FastEthernet. In the experiments, a subset of the nodes were designated as storage nodes, to emulate a storage cluster coupled to a compute cluster over a network. The second system (**OSC**) is a coupled compute and storage cluster system at the Ohio Supercomputer Center. The compute cluster consists of dual-processor nodes equipped with 2.4 GHz Intel P4 Xeon processors and 4 GB of memory, 62 GB of local scratch space, interconnected by an 8 Gbps Infiniband Switch. The compute cluster is connected to the storage system over another Infiniband Switch. The storage system consists of networked nodes, each of which is connected to an array of IBM FASTt600s over a Fiber Channel Switch [5]. Each node has a local file system that resides on FASTt600 storage units. For each of the workloads and hardware systems, we measured throughput (in terms of MBytes processed per second) for a batch and the amount of data transferred from storage nodes to compute nodes.

For the satellite data processing application, we used the emulator developed in [58, 31]. This emulator generates synthetic datasets that have similar characteristics to those obtained from AVHRR sensors on NOAA-7 series satellites [14]. The application (**TITAN**) operates on data blocks (chunks) that are formed by grouping subsets of sensor readings that are close to each other in spatial and temporal dimensions. On a parallel machine, the chunks are distributed across storage nodes to achieve I/O parallelism. The emulator allows the user to generate datasets of varying

sizes (corresponding to different numbers of days of sensor readings), the amount of data acquired per reading, and grouping of data blocks into files (one or multiple blocks can be stored in a data file). In our emulation, we assigned one data chunk per file. A data analysis task specifies the data of interest via a spatio-temporal window. For such a request, a list of data chunks, whose spatio-temporal bounding boxes intersect the request window, is generated. These blocks (files in our case) are retrieved from the storage system and processed by the data analysis task.

For the image analysis application, we implemented a program to emulate studies that involve analyses on images obtained from MRI and CT scans (captured on multiple days as follow-up studies). A dataset generated by the emulator is 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. The emulator allows creation of collections of such datasets, hence forming a database of images from multiple patients. Each image is stored in a separate file. A data analysis program can select a subset of images based on a set of patient ids and study ids, image modality, and a date range.

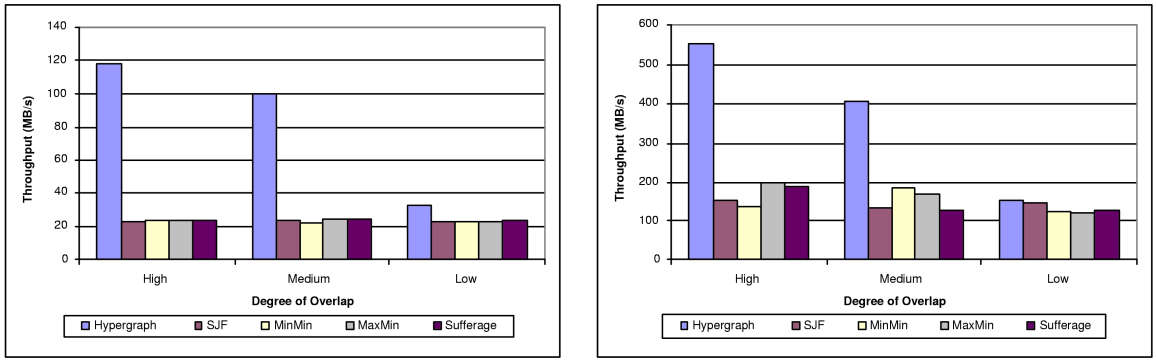
We evaluated the system for three different types of workloads; *high overlap*, *medium overlap*, and *low overlap*, each of which represents different amounts of file sharing among tasks in a batch. To generate a high overlap workload for the satellite data processing application, the emulator shifted the request window by 15% in all dimensions. This resulted in a 85% overlap, on the 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 the image analysis application, 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.

5.2 Performance Results

For the experiments, we generated 35 days worth of data, about 162 GB, for the satellite data processing application. The data was distributed across 4 storage nodes on each hardware configuration using a Hilbert-curve based declustering method [19]. Each file in the dataset was 4.5 MB. The number of tasks in a batch was equal to 200. In the high overlap case, each task accessed on an average 25 files. In the medium and low overlap cases, each task accessed on an average 10 files. For the image analysis application, the dataset generated by the emulator corresponded to a dataset of 200 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 about 68GB. Each batch comprised of 200 tasks, and each task accessed 5 MRI scans and 5 CT scans on average in the high, medium, and low overlap cases. Images for each patient were distributed among 4 storage nodes in a round robin fashion.

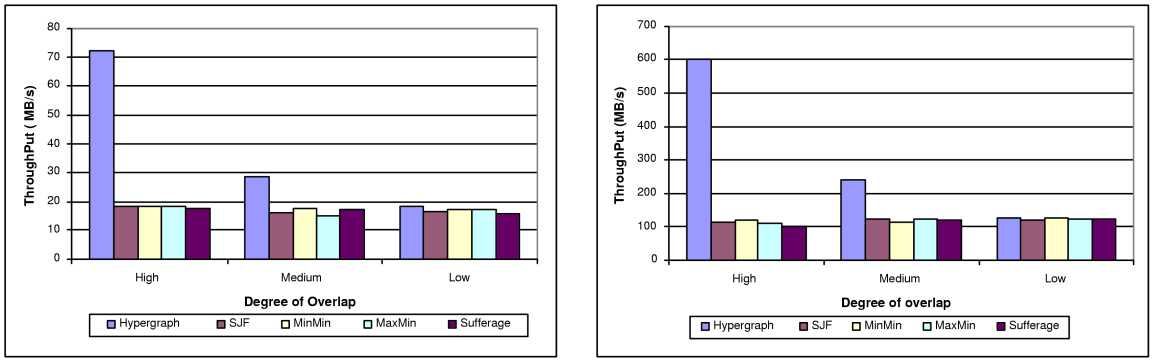
The two application classes have a wide range of operations that can be applied on data and that can have different computational requirements. In this paper we target data intensive applications, where I/O and communication overheads are comparable to that of computation. In order to create data intensive workloads and to emulate configurations where communication and remote I/O costs are relatively expensive, we chose the processing time for each task to be 0.001 seconds per Megabyte of data.

Figures 4 and 5 show the relative performance of the various scheduling schemes on work-



(a) (b)

Figure 4: Throughput achieved by different algorithms on the (a) OSUMED cluster and (b) OSC cluster, for the satellite data processing application.

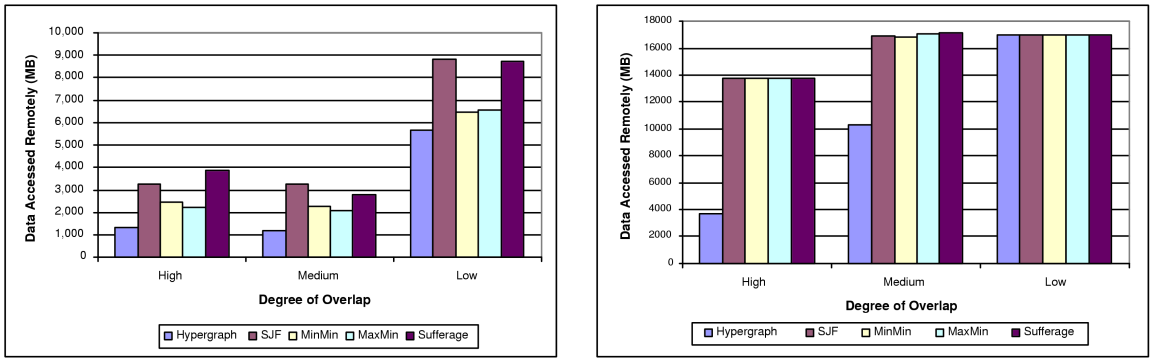


(a) (b)

Figure 5: Throughput achieved by different algorithms on the (a) OSUMED cluster and (b) OSC cluster, for the biomedical image analysis application.

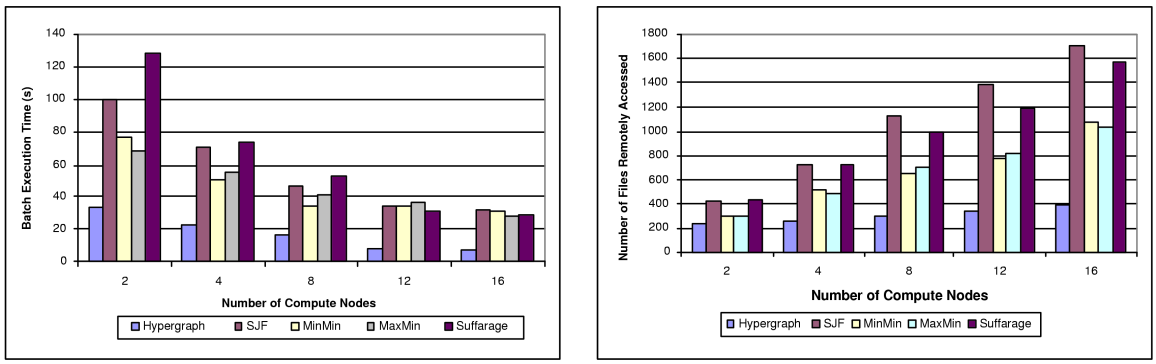
loads with different degrees of shared I/O among tasks. These experiments were conducted using 4 compute nodes and 4 storage nodes on both OSUMED and OSC systems. As is seen from the figures, the hypergraph based strategy performs better than the other algorithms for all cases. This is because the hypergraph algorithm is able to cluster tasks that share files together, thereby reducing the number of times the same file is transferred from the remote storage system. In addition, while minimizing the networking and I/O overheads, the hypergraph algorithm maintains computational load balance across the nodes. The gain due to hypergraph partitioning is maximum for the high overlap workload and reduces as the degree of overlap decreases, as expected. Among MinMin, MaxMin, SJF, and Sufferage, the Sufferage strategy performs slightly worse than other strategies. However, on average, MinMin, MaxMin, SJF, and Sufferage achieve more or less the same throughput irrespective of the type of workload.

The amount of data that is transferred from remote storage nodes to compute nodes is displayed in Figure 6. These experiments were carried out on the OSUMED cluster with 4 nodes as compute nodes and 4 nodes designated as storage nodes. The numbers in these graphs were computed as the sum of sizes of files transferred from remote storage nodes for all the tasks in the batch. Note that a file may be accessed multiple times from the storage cluster, if tasks that require the file are



(a) (b)

Figure 6: The amount of data remotely accessed for different algorithms for (a) the satellite data processing application and (b) the biomedical image analysis application.



(a) (b)

Figure 7: The performance of the scheduling strategies in the medium overlap case in the satellite data processing application as the number of compute nodes is varied on the OSC system. The number of storage nodes is equal to 4. (a) Batch execution time. (b) The number of files accessed remotely from the storage cluster.

mapped to nodes that do not have that file locally. Each such transfer is counted in calculating the amount of data transferred. We see from the figure that for both applications, the hypergraph-based strategy reduces the volume of data remotely accessed. Thus, we can expect that the performance gain obtained by the hypergraph based strategy over the other strategies will be bigger on systems with high remote I/O costs.

Figure 7 shows how the performance of the various schemes changes as the number of compute nodes is varied on the OSC system. In this experiment, the workload for the medium-overlap case in the satellite data processing application was used. The number of storage nodes was set to 4. As is seen from the figure, the hypergraph strategy achieves better performance than the other strategies in all configurations. An increase in the number of compute nodes allows for more computational parallelism. However, it also is likely to increase end-point contention on the storage nodes. Compared to the other strategies, the hypergraph strategy achieves better speedup in batch execution time when the number of compute nodes is increased, since it reduces end-point contention by reducing both the volume of data remotely accessed and contention on storage nodes

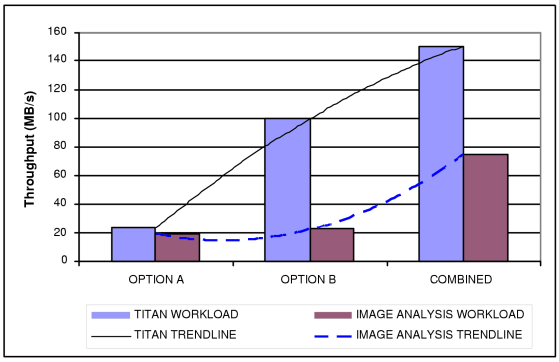


Figure 8: Contribution of different stages of the proposed scheduling strategy to throughput (in MBytes processed per second). The experiments were done on the OSC system with 4 compute and 4 storage nodes for the high overlap case in both applications. TITAN in the graph refers to the satellite data processing application.

via ordering tasks in a group and scheduling of file transfers dynamically. We observe that the volume of data transferred from the storage cluster increases with increasing number of compute nodes. This is expected since tasks will be distributed across more nodes when the number of compute nodes is increased. This will increase the probability that two tasks that share files will be mapped to different processors for execution and, as a result, the number of times a file is staged from the storage cluster to the compute cluster will increase. As is seen from the figure, the increase in the number of files transferred from storage nodes is less with the hypergraph strategy than that in the other strategies, when the number of compute nodes is increased. This is a result of the fact that sharing of files is explicitly modeled and taken into account in the hypergraph strategy for grouping and mapping of tasks to compute nodes.

Figure 8 quantifies the contribution of each stage of the hypergraph partitioning algorithm. *Option A* applies only the first stage (i.e., hypergraph partitioning of tasks; Section 4.2.2), but no dynamic scheduling of file transfers is done. That is, when a task is mapped to a processor, files for that task is transferred without taking into account storage node loads. *Option B* applies only the second stage of the algorithm (Section 4.2.3) without hypergraph partitioning of tasks. The ordering of tasks is applied to the entire batch and tasks are mapped to idle processors. *Combined* is the hypergraph based scheduling strategy applying both stages. We observe that Option A does not perform as well as Option B and the combined approach. This is because, minimizing the edge-cut weight may not ensure that there is no file system contention (as different files can map to the same file system or storage node). Option B improves the performance compared to Option A in the satellite data processing application, but the performance improvement in the image analysis application is small. The best performance is obtained by the combined approach. The improvement in using the combined approach over Option B is more in the case of image analysis workload than the satellite data processing workload, since the image analysis files are larger (16 MB) in comparison to titan files (4.5 MB). In that case, the grouping and mapping of tasks to compute nodes taking into account sharing of files is more beneficial. A result of our experiments is that both grouping and mapping of tasks to compute nodes and ordering of tasks and scheduling of file transfers should be considered in tandem to obtain the best performance.

Figure 9 displays the relative performance of the various scheduling schemes on a workload

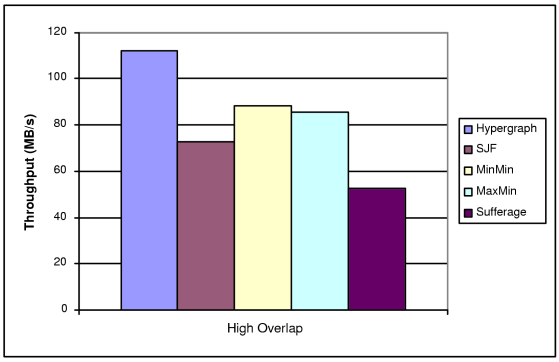


Figure 9: The satellite data processing application with large (100MB) files.

with high overlap of files among tasks in the satellite data processing application. In this experiment, the sizes of the files were scaled from 4.5MB per file to 100MB per file. The experiments were carried out on the OSC system with 4 compute nodes and 4 storage nodes. A comparison of this figure with Figure 4(b) reveals a lower throughput. This is because the increase in the file size reduces the effects of file system cache on the compute nodes, and the cost of local I/O becomes increasingly more pronounced in the execution time. For the same reason, even though the hypergraph strategy still performs better than the others, its performance improvement is less in this case.

6 Conclusion and Future Work

We presented and experimentally evaluated a new, hypergraph based strategy for scheduling a batch of tasks with batch shared I/O behaviour on systems with coupled storage and compute clusters. The proposed scheme aims to minimize the volume of remote data transfers and contention on storage nodes, while maintaining a balanced distribution of computational load across compute nodes. The salient features of this algorithm are that 1) it formulates the sharing of files among tasks as a hypergraph and uses hypergraph partitioning to map tasks to processors and 2) employs a dynamic task ordering and file transfer scheme to efficiently stage files from storage nodes to compute nodes. Our experimental results shows that our strategy achieves better performance compared to Shortest Job First, MinMin, MaxMin, and Sufferage strategies.

We plan to extend our work in several ways. One of the issues not addressed in our current approach is that we do not take into account how many files will be transfered to each group explicitly, which may lead to imbalanced transfer time and also imbalanced consumption of storage space on compute nodes. A possible approach to address this is to add another balance constraint to hypergraph partitioning, balancing of the hyperedges. We will investigate how we can incorporate this new balancing requirement to the current version of the hypergraph toolkit (PaToH) we are using and evaluate its performance. Another extension will be to incorporate dynamic updates to the hypergraph model (as new tasks arrive in the system), while taking into account the mapping of tasks that are executing and the mapping of files already staged from storage nodes to compute nodes. We also plan to investigate the efficiency of our strategy when applied in combination with different pre-fetching and data caching and cache replacement policies.

References

[1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, and A. Sussman. Tuning the performance of I/O-intensive parallel applications. In *Proceedings of the Fourth ACM Workshop on I/O in Parallel and Distributed Systems*, May 1996.

[2] H. Andrade, T. Kurc, A. Sussman, and J. Saltz. Scheduling multiple data visualization query workloads on a shared memory machine. In *Proceedings of the 2002 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002)*, Fort Lauderdale, FL, April 2002.

[3] R. Bennett, K. Bryant, A. Sussman, R. Das, and J. Saltz. Jovian: A framework for optimizing parallel I/O. In *Proceedings of the 1994 Scalable Parallel Libraries Conference*, pages 10–20. IEEE Computer Society Press, Oct. 1994.

[4] Biomedical Informatics Research Network (BIRN). http://www.nbirn.net.

[5] S. Bokhari, B. Rutt, P. Wyckoff, and P. Buerger. An evaluation of the osc fastt600 turbo storage pool. Technical Report OSUBMI_TR_2004_n02, The Ohio State University, Department of Biomedical Informatics, Sep 2004.

[6] 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.

[7] P. H. Carns, W. B. Ligon, R. B. Ross, and R. Thakur. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Oct. 2000.

[8] 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.

[9] U. V. Çatalyürek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplications. In *Proceedings of 3rd International Symposium on Solving Irregularly Structured Problems in Parallel, Irregular'96*, volume 1117 of *Lecture Notes in Computer Science*, pages 75–86. Springer-Verlag, 1996.

[10] 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.

[11] U. V. Çatalyürek and C. Aykanat. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at http://bmi.osu.edu/∼umit/software.htm, 1999.

[12] U. S. Chakravarthy and J. Minker. Multiple query processing in deductive databases using query graphs. In *Proceedings of the 12th International Conference on Very Large Data Bases Conference (VLDB 1986)*, pages 384–391, 1986.

[13] C. Chang, T. Kurc, A. Sussman, U. Catalyurek, and J. Saltz. A hypergraph-based workload partitioning strategy for parallel data aggregation. In *Proceedings of the Eleventh SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Mar. 2001.

[14] 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, Apr. 1997.

[15] F.-C. F. Chen and M. H. Dunham. Common subexpression processing in multiple-query processing. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):493–499, 1998.

[16] P. F. Corbett and D. G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, Aug. 1996.

[17] Earth Systems Grid (ESG). http://www.earthsystemgrid.org.

[18] M. M. Eshaghian and Y. C. Wu. Mapping heterogeneous task graphs ontp heterogeneous system graphs. pages 147–160. IEEE Computer Society Press, 1997.

[19] 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, Mar. 1989.

[20] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[21] GEON: Cyberinfrastructure for the Geosciences. http://www.geongrid.org.

[22] Grid Physics Network (GriPhyN). http://www.griphyn.org.

[23] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519–1534, 2000.

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

[25] M. Iverson and F. Ozguner. Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment. IEEE Computer Society Press, 1998.

[26] R. Jain, K. Somalwar, J. Werth, and J. Browne. Heuristics for scheduling i/o operations. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):310–320, Mar 1997.

[27] 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.

[28] M. V. Knopp, F. Giesel, H. Marcos, H. von Tengg-Kobligk, and P. Choyke. Dynamic contrast-enhanced magnetic resonance imaging in oncology. *Topics in Magnetic Resonance Imaging*, 12(2):301–308, 2001.

[29] M. V. Knopp, E. Weiss, H. Sinn, J. Mattern, H. Junkermann, J. Radeleff, A. Magener, G. Brix, S. Delorme, I. Zuna, and G. van Kaick. Pathophysiologic basis of contrast enhancement in breast tumors. *Journal of Magnetic Resonance Imaging*, 10:260–266, 1999.

[30] D. Kotz. Disk-directed I/O for MIMD multiprocessors. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74. ACM Press, Nov. 1994.

[31] T. M. Kurc, A. Sussman, and J. Saltz. Coupling multiple simulations via a high performance customizable database system. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Mar. 1999.

[32] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs. *ACM Computing Surveys*, 31(4):406–471, Dec. 1999.

[33] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Willey–Teubner, Chichester, U.K., 199.

[34] S. Liang, L. Davis, J. Townshend, R. Chellappa, R. Dubayah, S. Goward, J. JaJa, S. Krishnamachari, N. Roussopoulos, J. Saltz, H. Samet, T. Shock, and M. Srinivasan. Land cover dynamics investigation using parallel computers. In *Proceedings of the 1995 International Geoscience and Remote Sensing Symposium, Quantitative Remote Sensing for Science and Applications.*, pages 332–4, July 1995.

[35] 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–, Apr. 1999.

[36] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. IEEE Computer Society Press, 1998.

[37] J. M. May. *Parallel I/O for High Performance Computing*. Morgan Kaufmann Publishers, 2000.

[38] MEDIGRID. http://creatis-www.insa-lyon.fr/MEDIGRID/home.html.

[39] M. Mehta, V. Soloviev, and D. J. DeWitt. Batch scheduling in parallel database systems. In *Proceedings of the 9th International Conference on Data Engineering (ICDE 1993)*, Vienna, Austria, 1993.

[40] N. Nieuwejaar and D. Kotz. The Galley parallel file system. In *Proceedings of the 1996 International Conference on Supercomputing*, pages 374–381. ACM Press, May 1996.

[41] A. R. Padhani. Dynamic contrast-enhanced MRI in clinical oncology: Current status and future directions. *Journal of Magnetic Resonance Imaging*, 16:407–422, 2002.

[42] J.-P. Prost, R. Treumann, R. Hedges, B. Jia, and A. Koniges. MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS. In *Proceedings of the 2001 ACM/IEEE SC01 Conference*. ACM Press, Nov. 2001.

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

[44] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM International Conference on Management of Data (SIGMOD 2000)*, pages 249–260, 2000.

[45] J. Saltz and et.al. Driving scientific applications by data in distributed environments. In *Dynamic Data Driven Application Systems Workshop, held jointly with ICCS 2003*, Melbourne, Australia, June 2003.

[46] V. Sarkar. Determining average program execution times and their variance. In *Proceedings of the ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, pages 298–312. ACM Press, June 1989.

[47] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective I/O in Panda. In *Proceedings Supercomputing '95*. IEEE Computer Society Press, Dec. 1995.

[48] SEEK: Science Environment for Ecological Knowledge. http://seek.ecoinformatics.org.

[49] X. Shen and A. Choudhary. A distributed multi-storage i/o system for high performance data intensive computing. In *International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, May 2002.

[50] C. T. Shock, C. Chang, B. Moon, A. Acharya, L. Davis, J. Saltz, and A. Sussman. The design and evaluation of a high-performance earth science database. *Parallel Computing*, 24(1):65–90, Jan. 1998.

[51] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the Grid. In *Proceedings of the 2002 ACM/IEEE SC02 Conference*. ACM Press, Nov. 2002.

[52] Shared Pathology Informatics Network (SPIN). http://www.sharedpath.org.

[53] H. Tang and T. Yang. An efficient data location protocol for self.organizing storage clusters. In *ACM/IEEE SC2003*, Phoenix, AZ, November 2003.

[54] 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.

[55] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi. Passion: Optimized I/O for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.

[56] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 3–14, San Juan, Puerto Rico, Apr. 1999. IEEE Computer Society Press.

[57] The USGS General Cartographic Transformation Package, version 2.0.2. *ftp://mapping.usgs.gov/pub/software/current_software/gctp/*, 1997.

[58] 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 Run-time Systems for Scalable Computers*, number 1511 in Lecture Notes in Computer Science, pages 243–258. Springer-Verlag, May 1998.

[59] N. Vydyanathan, G. Khanna, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. Use of pvfs for efficient execution of jobs with pipeline-shared i/o. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, 2004. to appear.

[60] J. B. Weissman and X. Zhao. Run-time support for scheduling parallel applications in heterogeneous nows. August 1997.

[61] S. Yang, D. Gannon, S. Srinivas, and F. Bodin. High Performance Fortran interface to the Parallel C++. In *Proceedings of the Scalable High Performance Computing Conference (SHPCC-94)*, pages 301–308. IEEE Computer Society Press, May 1994.

[62] Y. Zhao, P. M. Deshpande, J. F. Naughton, and A. Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. In *Proceedings of the 1998 ACM International Conference on Management of Data (SIGMOD 1998)*, pages 271–282, Seattle, WA, 1998.