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[†]

[†] Dept. of Computer Science and Engineering, [‡] Dept. of Biomedical Informatics
The Ohio State University
[+] Ohio Supercomputer Center

Abstract

This paper proposes a novel, hypergraph partitioning based strategy to schedule multiple data analysis tasks with batch-shared I/O behavior. 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 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, 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, platforms for large scale, disk-based storage are becoming increasingly available to store and manage large scientific datasets.

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 component in data-driven science. A data analysis application 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.

In earlier work [15] we examined algorithms for scheduling pipelines of data processing with *pipeline-shared* I/O behavior, where files and data are shared between tasks forming a single pipeline of data processing operations. This paper focuses on scheduling of tasks with *batch-shared* I/O behavior [13], in which files are shared across tasks in different pipelines. We propose a novel, hypergraph based approach. Hypergraphs have attracted much attention for modeling the computational structure of many parallel applications [4, 5]. 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 [4].

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 [12], and later work such as Yang and Gerasoulis [16]. The goal on distributed memory par-

[*]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.

allel machines is to trade-off parallelism with communication. Some of these techniques deal with a single application structured as a DAG, while others apply to scheduling many independent tasks. Relatively little research so far has addressed the scheduling of data intensive jobs. In [11], 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. Casanova et.al. [3] proposed modification to several heuristics that were designed for compute intensive applications on parallel machines [8] for parameter sweep applications in a Grid environment.

Multi-query workloads also arise in the context of database applications. The work of Mehta et al. [10] is one of the first to address the problem of scheduling queries in a parallel database by considering batches of queries. In [1], 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 based on priority metric. These strategies mainly target efficient reuse of results from previously executed queries.

Chang et.al. [5] examine optimization methods for executing data aggregation operations on disk-resident datasets on distributed-memory machines with local disks. Drawing from the *computational hypergraph* model proposed in [4], the authors propose a hypergraph based algorithm for partitioning of workload among processors and for scheduling of processing. Jain et.al. [9] model scheduling of I/O operations (with certain assumptions) as a bipartite graph coloring problem with two separate sets of nodes namely, disks and processors. Our difference is that 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

Satellite data processing. Remotely sensed data is an invaluable source of information for earth scientists. This kind of data is either continuously acquired or captured on-demand via sensors attached to satellites orbiting the earth [6]. 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 (e.g., cancer) 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.

Problem Definition. In this paper, we target configurations consisting of coupled compute and storage platforms. Datasets are stored, as a set of *data files*, 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. 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. The tasks in the batch may share a number of files. For example, if tasks 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. 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

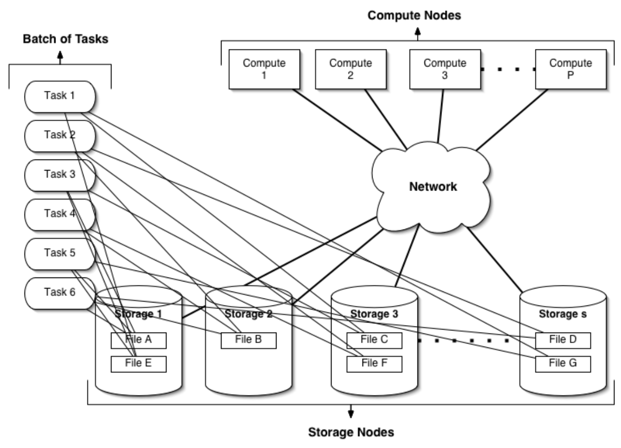


Figure 1. Scheduling problem.

files, and their file sizes.

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 [8], along with Shortest Job First heuristics and our proposed algorithm to determine task-compute node assignments. As in [3], 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 (MCT) 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 MCT of each task on each idle 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 MCT of a task on each idle 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.

4.2 A Hypergraph-based Approach

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. The algorithm operates in two stages. 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.

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 $\mathcal{H}$, *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 cost of a partition Π is $\chi(\Pi) = \sum_{n_j \in \mathcal{N}_E} c_j(\lambda_j - 1)$, where $\mathcal{N}_E$ is the set of cut nets and each cut net n_j contributes $c_j(\lambda_j - 1)$ to the cutsize. Hence, this cost metric is also known as *connectivity-1* metric. 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 PaToH [4], have been shown to compute good partitions quickly for this NP-hard problem.

4.2.1 Hypergraph Formulation for Partitioning and Mapping of Tasks

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. 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

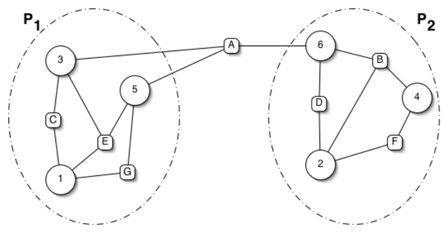


Figure 2. Hypergraph representation of the batch of the tasks displayed in Figure 1.

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 partitioning of the hypergraph representation of the sample batch shown in Figure 1.

The weight of a vertex is equal to the estimated execution time of the corresponding task. 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. 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. Hence it has to use a static vertex weights. In order to alleviate this issue 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 one byte of file f_j, Tr_j, for task t_i is equal to

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

Here, RBW is the I/O bandwidth between a storage node and a compute node, $Prob_{FNE}$ 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. Hence, we have used $Prob_{FNE} = \frac{1}{s_j}$ and $Prob_{FE} = \frac{1}{P}$. 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

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

where LBW is the I/O bandwidth from local disk on a compute node and C is the compute cost of one byte. 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 *connectivity-1* cutsize definition [4]. 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.

4.2.2 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. Two tasks that are in different groups 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 multiple requests to the same storage node are serialized and that a 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

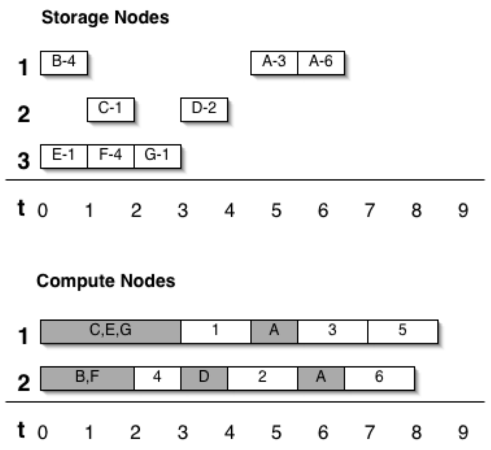


Figure 3. An illustration of the execution of the ordering algorithm on the batch of tasks shown in Figure 2.

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 I/O and processing of a file takes 0.3 and 0.2 units of time, respectively. 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.

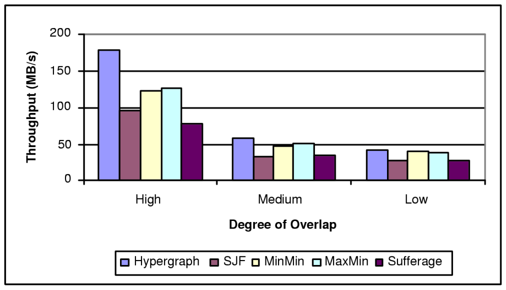
5 Experimental Results

We experimentally evaluated the scheduling algorithms using two application classes, satellite data processing and biomedical image analysis, described in Section 3. We used the PaToH toolkit [4] 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.

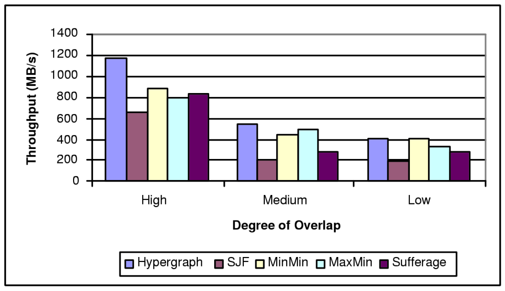
The experiments were carried out on two systems. The first system is a cluster of Pentium III 933 MHz nodes (**OSUMED**). Each node of this cluster has 300GB disk space and 512MB of memory. The nodes are connected through a Switched FastEthernet. In the experiments, a subset of the nodes were designated 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 [2]. 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 [14]. The application (**TITAN**) [6] operates on data chunks that are formed by grouping subsets of sensor readings that are close to each other in spatial and temporal dimensions. 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 chunks into files. In our emulation, we assigned one data chunk per file. A data analysis task specifies the data of interest via a spatio-temporal window. The corresponding files 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. 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*,

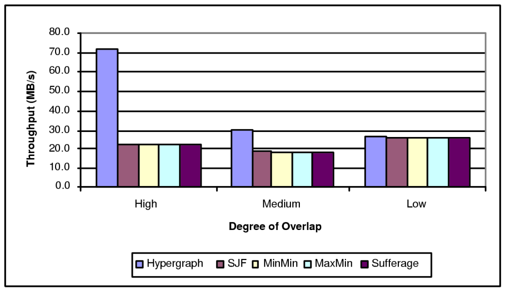


(a)

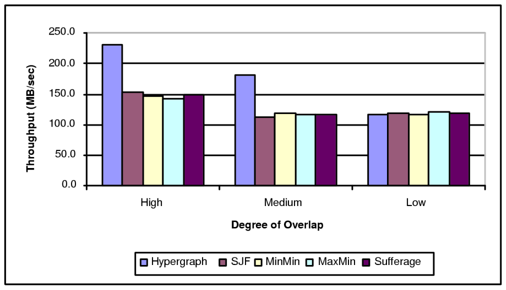


(b)

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



(a)



(b)

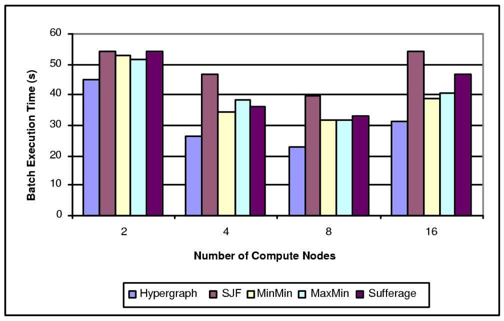
Figure 5. Throughput achieved by different algorithms on the (a) OSUMED cluster and (b) OSC cluster, for the biomedical image analysis application.

each of which represents different amounts of file sharing among tasks in a batch. To generate workload for the satellite data processing application, we have simulated queries directed to geographically distant parts of the world. 4 sets of queries with 50 queries in each has been generated representing the queries directed to 4 hot spot regions. Across the sets there is no overlap between the queries, and in each set queries are adjusted such that for high overlap workload, they resulted in a 85% overlap, on 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.

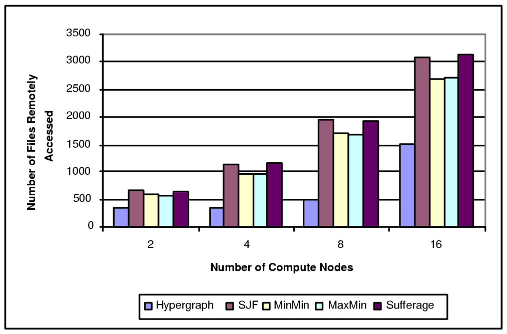
For the experiments, we generated 35 days worth of data, about 162 GB, for the satellite data processing application. The data was distributed across 4 storage nodes on each hardware configuration using a Hilbert-curve based declustering method [7]. Each file in the dataset was 4.5 MB. The number of tasks in a batch was equal to 200. In the high overlap case, each task accessed on an average 30 files. In the medium and low overlap cases, each task accessed on an average 8 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.

In order to create data intensive workloads which are targeted in this paper 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 workloads 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



(a)



(b)

Figure 6. 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.

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.

Figure 6 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 high-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 hence the performance decrease at 16 compute nodes in all approaches. 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.

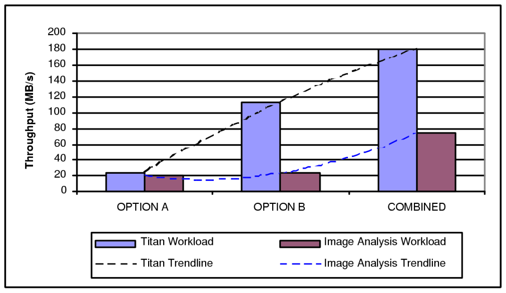


Figure 7. Contribution of different stages of the proposed scheduling strategy to throughput (in MBytes processed per second).

Figure 7 quantifies the contribution of each stage of the hypergraph partitioning algorithm. The experiments were done on the OSC system with 4 compute and 4 storage nodes for the high overlap case in both applications. *Option A* applies only the first stage (i.e., hypergraph partitioning of tasks; Section 4.2.1), 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 4.2.2) 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.

6 Conclusions

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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.