Task Scheduling and File Replication for Data-Intensive Jobs with Batch-shared I/O *

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

Abstract

This paper addresses the problem of efficient execution of a batch of data-intensive tasks with batch-shared I/O behavior, on coupled storage and compute clusters. Two scheduling schemes are proposed: 1) a 0-1 Integer Programming (IP) based approach, which couples task scheduling and data replication, and 2) a bi-level hypergraph partitioning based heuristic approach (BiPartition), which decouples task scheduling and data replication. The experimental results show that: 1) the IP scheme achieves the best batch execution time, but has significant scheduling overhead, thereby restricting its application to small scale workloads, and 2) the BiPartition scheme is a better fit for larger workloads and systems – it has very low scheduling overhead and no more than 5-10% degradation in solution quality, when compared with the IP based approach.

1 Introduction

Several scientific applications store datasets in collections of files. A request for data analysis specifies a subset of data files, either as a parameter of the request or through an index lookup that locates the files (or file segments) that satisfy the request. The data of interest is retrieved from the storage system and transformed into a data product, which is more suitable for examination by the scientist. However, unlike traditional compute intensive jobs, data analysis tasks may require access to large numbers of files and high data volume. When mapping tasks to compute nodes, scheduling mechanisms need to take into account not only their computation times, but the staging of files should also be carefully coordinated to minimize the I/O overheads.

This paper addresses the efficient execution of a batch of data-intensive tasks exhibiting batch-shared I/O behavior [14] on coupled storage and compute clusters. Batch-shared I/O simply means that the same file may be required by multiple tasks in a batch. In a coupled storage-compute cluster system, a group of machines with a large local disk pool form the storage cluster. This cluster is connected to a compute cluster over a local area network. The files required by the tasks are initially resident on the storage cluster. If tasks can be executed on the storage nodes, the cost of data staging can be avoided. However, often it is not feasible to execute tasks on storage nodes – the access policies may not allow user tasks to execute on storage nodes, or the storage nodes may be designed to maximize storage space and I/O bandwidth, forgoing computation power. In this work, we assume that tasks cannot be scheduled on storage nodes – when a task is scheduled on a processing node, the files accessed by the task must be staged on the processing node before the task is executed. A file can be staged on a node either through a remote transfer from the storage cluster or by copying it from another compute node that already has it, thereby reducing contention on the storage cluster. We also model disk space constraints on the compute cluster.

We approach the problem as a three stage process. The first stage, called sub-batch selection, partitions a batch of tasks into sub-batches such that the total size of the files required for a sub-batch does not exceed the available aggregate disk space on the compute cluster. The second stage accepts a sub-batch as input and yields an allocation of the tasks in the sub-batch onto the nodes of the compute cluster to minimize the sub-batch execution time. The third stage orders the tasks allocated to each node at runtime and dynamically determines what file transfers need to be performed and how they should be scheduled to minimize the end-point contention on the storage cluster.

We propose two approaches to solve this three stage problem. The first approach formulates the sub-batch selection problem using a 0-1 Integer Programming (IP) formulation. The second stage is also modeled as a 0-1 IP formulation to determine the mapping of tasks to nodes, source and destination nodes for all replications, and the destination nodes for all remote transfers. The second approach, called BiPartition, employs a bi-level

*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.

hypergraph partitioning based scheduling heuristic that formulates the sharing of files among tasks as a hypergraph. The BiPartition strategy uses a two level partitioning approach to address the first and the second stage of the problem. The first level partitioner is used to divide the batch of tasks into sub-batches, whose data requirement fits into the available disk space on the compute cluster. The second phase yields a load balanced, cut minimizing partition of tasks on the compute cluster. We carry out an experimental evaluation of these algorithms, comparing them against a MinMin based heuristic and a decoupled computation and data scheduling approach [13]. Application emulators from two application domains are used – analysis of remotely-sensed data and biomedical imaging.

2 Problem Definition

We target batches which consist of independent sequential programs. Each task requests a subset of data files from a dataset and can be executed on any of the nodes in the compute cluster. The data files required by a task should be staged to the compute node where the task is allocated for the task to execute correctly; a data file is the unit of I/O transfer from the storage cluster to the compute cluster. The tasks in the batch may share a number of files. If a file is required for processing by one or more tasks on a particular node, it may be retrieved either from the remote storage system or from another compute node which already has the file. The decision to *replicate*[1] a file in this way depends on the mapping of the tasks that require the file and vice-versa. We assume a single port model wherein 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.

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

We have evaluated our approach using application scenarios from two application classes; analysis of remote sensing data and biomedical image analysis: (1)

[1] In the context of this work, we use the term replication to denote only the file transfers between pairs of compute nodes, one of which acts as the source and the other the destination. Staging of files from the storage system can also create replicas of files on the compute cluster. For such transfers, we use the term remote transfers and do not associate them with replication.

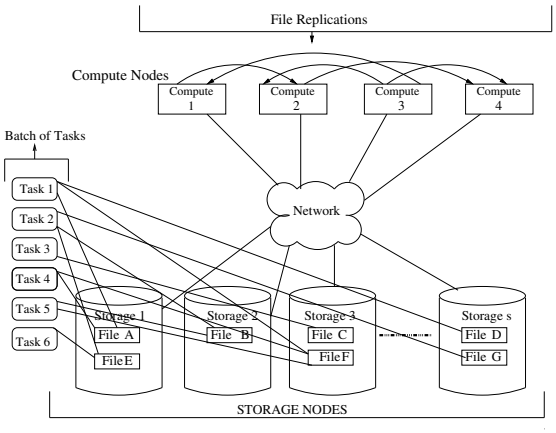


Figure 1. Scheduling problem.

Satellite data processing. Remotely sensed data is either continuously acquired or captured on-demand via sensors attached to satellites orbiting the earth [7]. Datasets of remotely sensed data can be organized into multiple files. Each file contains a subset of data elements acquired within a time period and a region of the earth. 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. (2) **Biomedical Image Analysis.** Biomedical imaging is a powerful method for disease diagnosis and for monitoring therapy. State-of-the-art studies make use of large datasets, which consist of time dependent sequences of images from multiple imaging sessions. Systematic development of image analysis techniques requires an ability to efficiently invoke candidate image quantification methods on large collections of images. A researcher may apply several different image analysis methods on image datasets to assess ability to predict outcome or effectiveness of a treatment across patient groups.

3 Related Work

Relatively little research has so far addressed the co-scheduling of task execution and data replication. MinMin [12] is a well-known algorithm for job scheduling, and can be applied to our problem scenario by incorporating the cost of data access. When computing the expected minimum completion time (MCT) of a task on a node, MinMin takes into account the files already available on the node and files already available on other compute nodes which can therefore act as alternate sources for creating file replicas other than the remote storage system. When a task is scheduled on a node, all of its files are staged on the corresponding node. This leads to an implicit replication policy as multiple copies of files may be created on different nodes of the compute cluster. Each file required for a task is staged from one

of the replicas or from the storage cluster such that the time to transfer the file is minimized.

Ranganathan et. al. [13] proposed a decoupled approach to scheduling of computations and data for data-intensive applications in a grid environment, and evaluated its effectiveness via simulation studies. The algorithm combines a scheduling scheme, called *Job Data Present* with a replication heuristic, referred to as *Data Least Loaded*, in a decoupled fashion. In this algorithm, a single file per task is employed which means that either a compute node stores the file required by a task or it does not. The algorithm incorporates a notion of eligible nodes for each task, which are the set of nodes that store the file required by the task. It works by selecting a task from a FIFO queue and assigning it to the node that already has the required data. If more than one compute node is an eligible candidate, then it chooses the least loaded node. The replication mechanism *Data Least Loaded* is decoupled from the scheduling policy. The replication mechanism keeps track of the popularity of files, and when the popularity of a file exceeds a threshold, it is replicated on the least loaded node in the compute cluster. For the applications considered here, multiple files may be accessed by a task which means that there may exist compute nodes which store subsets of the files required by a job. This essentially amounts to allocating a job to a node such that the expected data transfer time to stage in the set of files required by a task is minimized. Moreover, since we are focusing on scheduling of a batch of tasks, a local FIFO scheduling policy is not meaningful since all tasks arrive at the same time. We apply a simple local scheduling policy based on the least expected earliest completion time of the tasks to adapt the aforesaid scheme to our scenario.

While the approach of Ranganathan et. al. [13] was shown to be well suited to an online environment where tasks arrive over time and there is a lack of knowledge about file access patterns of future jobs, we show that our proposed approaches which seek to exploit global inter-task file affinity information are more effective in the batch context we consider. We experimentally compare our approach against MinMin scheduling with implicit replication of files and a batch-mode variant of the *Job Data Present with Data Least Loaded* approach proposed by Ranganathan et al. [13].

Casanova et al. [4] modified the MinMin, MaxMin, and Sufferage job scheduling heuristics to take into account the cost of inter-site file access, in the context of scheduling parameter sweep applications in a Grid environment. Our work targets an environment with a compute cluster and storage cluster, and explicitly models the effect of file replication. Desprez et al. [8] proposed an algorithm that combines data management and scheduling using a steady state approach. Their model does not incorporate any limited disk space constraints. However, our formulation takes into account the fact that the aggregate disk space may be not be sufficient to concurrently store at least one copy of each file. The work of Bent et al. [1] also focuses on the problem of coordination of data movement and computation scheduling. However, they assume that a task accessing multiple files can be split into a set of subtasks accessing a single file each, that can be allocated and scheduled independently. We use a more general model with single/multiple files per task.

In earlier work [10], we modeled the sharing of files among tasks as a hypergraph and employed hypergraph partitioning to obtain a computationally load-balanced mapping of tasks onto compute nodes that reduced remote I/O operations for file transfers. Our earlier work assumed that a compute node had enough disk space to hold all of the files staged on that node (i.e., assumed infinite disk cache space on each compute node) and did not consider replication of files. In contrast to our earlier work, here we allow explicit replication of files on compute nodes, and also incorporate disk space constraints on the compute nodes.

4 0-1 Integer Programming-based Approach

We approach the overall problem as a 3 stage process: The first stage is sub-batch selection; the second stage handles allocation of tasks; and the third stage implements scheduling of file transfers. We assume that each node on the compute cluster has a local disk, which can be used as a disk cache for files staged from storage nodes. We first present the IP formulation for the *unlimited disk cache space* case. In this case, each compute node has enough space to store at least one copy of each file requested by the tasks in the batch. We then describe an extension to handle limited disk cache space.

4.1 Unlimited Disk Cache Space

For this case, the sub-batch selection problem need not be solved, since the disk space on the compute cluster is not a constraint. Therefore, we directly solve for the second stage.

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

$R_{\ell i}$=1 if file f_ℓ is remotely transferred to node c_i, 0 otherwise. Let T_{ki} be a binary variable where T_{ki}=1 if task t_k is allocated to node c_i, 0 otherwise. The objective function is the minimization of the overall batch execution time under a set of constraints. The constraints are as follows:

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

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

A file is copied to (i.e., replicated on) a compute node, only if a task requiring the file is allocated to that node.

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

For a particular node c_i and a file f_ℓ stored on one or more other compute nodes, the file will be copied to (replicated onto) c_i from only one of the other compute nodes. Note that in the limited cache space case, a file f_ℓ may need to be copied to a node c_i multiple times, if the file is evicted due to cache space constraints and is required by tasks allocated to c_i in the future. In the unlimited cache space case, however, files are never evicted. Thus, a file is replicated on a node only once. Constraints represented by equations 3– 5 obey this condition.

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

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

$$(\forall i)(\forall \ell) X_{\ell i} = R_{\ell i} + \sum_{\forall j, j \neq i} Y_{ji\ell} \quad (4)$$

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

$$(\forall i)(\forall \ell) R_{\ell i} + \sum_{\forall j, j \neq i} Y_{ji\ell} \leq 1 \quad (5)$$

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

$$(\forall k) \sum_{\forall i} T_{ki} = 1 \quad (6)$$

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

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

Every file requested by the tasks in the batch will be retrieved from remote storage nodes at least once. We assume that initially all the files are resident on only the remote storage cluster. Thus, each file needs to have at least one remote transfer.

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

Given these constraints, the objective is to minimize the batch execution time $Batch\_Exec\_Time$, which is the maximum of the execution time of each node. The execution time of a node c_i is defined as $Exec_i$. It is the sum of three components: the replication cost associated with that node ($Replication_i$), the computation cost of tasks allocated to that node ($Computation_i$), and the remote transfer cost of files transfered to that node ($Remote_i$).

$$Replication_i = \sum_{(\forall \ell)(\forall j, j \neq i)} (Y_{ji\ell} + Y_{ij\ell}) \times t_{rep} \times fsize(f_\ell) \quad (9)$$

$$Computation_i = \sum_{\forall k} Comp_k \times T_{ki} \quad (10)$$

$$Remote_i = \sum_{\forall \ell} R_{\ell i} \times t_{rem} \times fsize(f_\ell) \quad (11)$$

$$Exec_i = Replication_i + Remote_i + Computation_i \quad (12)$$

$$Batch\_Exec\_Time = \max_{\forall i} \{Exec_i\} \quad (13)$$

In these equations, t_{rep} is the replica creation cost per byte; t_{rem} is the per byte remote transfer time; $Comp_k$ represents the computation cost of task t_k, and $fsize(f_\ell)$ is the size of the file f_ℓ.

There can be overlap between communication and computation across different nodes in the system, i.e., a task may be executing on a compute node, while files for another task are being staged on another compute node. We assume a single port model wherein 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. In addition, no files are staged on a compute node while a task is executing on the node. Equation 12 reflects these constraints.

The IP formulation effectively exploits the global task-file sharing information and yields a one-step solution which comprises of both mapping of tasks and placement of files. However, it does not provide a solution when disk cache space is limited. To address the problem of limited cache space, we propose a 2-stage IP formulation as described in the next section.

4.2 Limited Disk Cache Space: A Two-stage 0-1 IP Solution

In the limited disk cache space case, we assume that there is enough space on each compute node to store all

the files required for any single task. Since the aggregate available disk space on the compute cluster is not sufficient to stage in all the files required by the batch under consideration, a disk file eviction mechanism is needed in conjunction with the IP based scheduling approach. The file eviction mechanism is discussed in more detail in Section 4.3.

The first stage of the two-stage IP solution takes as input a set of tasks and yields a subset of the tasks, referred to here as a sub-batch, such that the size of the files required for a sub-batch is less than or equal to the aggregate available disk space on the compute cluster. The second stage applies on each sub-batch the 0-1 IP formulation for the unlimited disk space with one additional constraint. The additional constraint is required to account for the fact that the space on a compute node may not be sufficient to store all the files required by the sub-batch. After a sub-batch is executed, the two stages are applied on the remaining set of pending tasks. This is repeated until all tasks in the batch are executed. We note that subsequent iterations of this two stage solution also model the fact that copies of some files have already been created on the compute cluster due to previous sub-batch executions.

First Stage: Sub-batch Selection. The primary goal of this stage is to divide a batch of tasks into subsets of tasks such that the tasks in a subset can execute on the compute cluster without the need for any file eviction. It also aims to minimize the number of sub-batches so that the scheduling overhead of multiple sub-batch executions is reduced. This is achieved by choosing a *maximally sized* subset of tasks at each sub-batch selection step, i.e., the sub-batch is formed with the maximum number of tasks which do not violate the disk space constraints. This essentially amounts to choosing a subset of tasks which have high degree of file sharing among themselves. In addition, allocation of the tasks in the sub-batch across compute nodes should be computationally balanced. The objective function of the IP formulation is then to choose a load balanced, maximally sized subset of tasks which do not violate disk space constraints.

$$Objective\_Function = \max_{(\forall i)(\forall j)} \sum T_{ij} \qquad (14)$$

The constraints of the IP formulation are as follows. The allocation of a task to a node entails the staging of all the files required by the task onto the node.

$$(\forall i)(\forall k)(\forall \ell \in \ Access_k) T_{ki} \leq X_{\ell i} \qquad (15)$$

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

$$(\forall i) \sum_{\forall \ell} X_{\ell i} \times fsize(f_\ell) \leq DiskSpace_i \qquad (16)$$

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

$$(\forall k) \sum_{\forall i} T_{ki} \leq 1 \qquad (17)$$

In order to achieve a load balanced mapping of tasks in a sub-batch to compute nodes, we enforce a constraint that the computation time on any node should be within a certain tolerance $Thresh$ of the average computation time over all the nodes. Essentially what it means is that the sub-batch should be chosen in such a way that the eventual sub-batch allocation in the second stage leads to a load balanced solution. In the following equations, Avg_Comp_time denotes the average of the computation times over all the nodes and C is the number of compute nodes.

$$(\forall i) Computation_i <= Avg\_Comp\_Time \times (1 + Thresh) \qquad (18)$$

$$Computation_i = \sum_{\forall k} Comp_k \times T_{ki} \qquad (19)$$

$$Avg\_Comp\_Time = \frac{1}{C} \times \sum_{\forall i} Computation_i \qquad (20)$$

Second Stage: Sub-batch Allocation Optimized for Overall Execution Time. The sub-batch selection phase yields a subset of tasks and their allocations. However, it does not take into account the fact that some of the files may have already been copied to compute nodes (for previous sub-batches) and that fetching a file from a nearby compute node is less expensive than fetching it from the remote storage system. Thus, to achieve the best possible allocation for a given sub-batch, the set of tasks in the sub-batch is input to the 0-1 IP algorithm given in Section 4.1 with one additional constraint on disk space: The total storage space taken by files allocated to a particular node should not exceed the disk space available on that node.

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

This second stage of the algorithm yields a schedule and data placement information for the sub-batch.

4.3 File Eviction Policy

Once a sub-batch finishes execution, a disk file eviction mechanism is invoked, which marks files for deletion in increasing order of their popularity. At the end

of this phase, each node has as much storage space as required to execute at least a single task. This phase is followed by the 2-stage process explained before - with the input being the set of remaining pending tasks. The popularity of a file, $Popularity_\ell$, is calculated as follows.

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

$Access\_Freq_\ell$ represents the number of pending requests to the file. This information can be easily obtained from the original batch and the set of tasks which have already finished execution. $fsize(f_\ell)$ represents the size of the file f_ℓ. $Numcopies_\ell$ represents the number of copies of file f_ℓ in the compute cluster. If two files have the same probability of access and the same size, the file with fewer copies gets a higher popularity, since deleting that file is more likely to result in remote file transfer when the file is needed. The intuition behind including the file size in the popularity computation is that the greater the size of the file, greater the cost of getting the file back to a node. The algorithm deletes smaller files, since the cost of staging such files again in the future is lower.

We have integrated this mechanism into our proposed approach as well as *MinMin* with *Implicit Replication*. For the algorithm *Job Data Present* with *Data Least Loaded*, we employ an LRU based mechanism as described in [13].

5 Bi-level Hypergraph-based Approach

In recent work [11], a bi-level hypergraph partitioning based approach was developed in the context of efficient execution of parallel out-of-core applications operating on block-sparse data. In this work, we extend this idea with an intelligent method to assign weights to vertices of a hypergraph, while taking into account node-to-node data replication in the compute cluster. We also couple this bi-level task mapping approach with task ordering and file staging (see Section 6) for a complete end-to-end solution. In the bi-level hypergraph partitioning approach, the task-file sharing interactions are modeled using a hypergraph. The first level of partitioning divides the batch of tasks into multiple disjoint sub-batches such that the storage space requirement of each sub-batch does not exceed the available aggregate disk space on the compute cluster. The second level of partitioning takes as input a sub-batch and computes a load-balanced cut minimizing mapping of the tasks in the sub-batch onto the compute nodes of the cluster.

5.1 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 can be assigned to the vertices (w_i for $v_i \in \mathcal{V}$) and nets (c_j for $n_j \in \mathcal{N}$) of the hypergraph. $\Pi = \{V_1, V_2, \ldots, V_P\}$ 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 the traditional hypergraph partitioning problem, a partition is said to be balanced if $W_p \leq W_{avg}(1+\epsilon)$ for $1 \leq p \leq P$, where $Wp = \sum_{v_i \in V_p} w_i$ is the sum of the vertex weights of part V_p , $W_{avg} = (\sum_{v_i \in V} w_i)/P$ denotes the weight of each part under the perfect load balance condition, and ϵ represents the predetermined maximum imbalance ratio allowed. In a partition Π 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 computed as:

$$\chi(\Pi) = \sum_{n_j \in N_E} c_j(\lambda_j - 1) \quad (23)$$

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. 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 [5], have been shown to compute good partitions quickly for this NP-hard problem.

To address the limited storage space constraint, we employ a different flavor of the hypergraph partitioning problem called the *Bounded Incident Net Weight (BINW) Partitioning* [11]. In BINW partitioning, the cost of a partition is again computed using the connectivity-1 cut-size definition (Eq. 23), but the constraint on the partitioning is different. Let $\Pi = \{V_1, V_2, ..., V_P\}$ be the P-way partition of hypergraph H and $I(V_i)$ denote the nets that are incident on vertices in V_i, i.e., $I(V_i) = \{n_j | v_k \in n_j, \forall v_k \in V_i\}$. The BINW partitioning is defined as finding a minimum cost partition where each part's incident net weight sum is bounded by a predetermined weight constraint D:

$$\sum_{n_j \in I(V_i)} c(n_j) \leq D \quad (24)$$

Note that P is not predetermined in this problem; however, minimizing the connectivity-1 cost while obeying the incident net weight constraint would also minimize the number of parts.

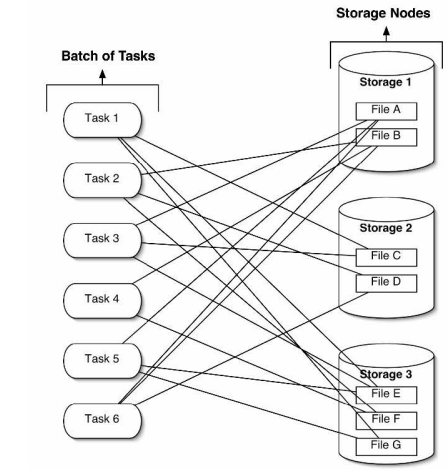
In this work, we have used the BINW partitioner proposed in [11] as a modification of the successful multi-

level hypergraph partitioner PaToH [6]. PaToH achieves P-way partitioning through recursive bisection. BINW partitioning necessitates revisiting all three phases of multilevel partitioning; *coarsening, initial partitioning* and *refinement*, as well as the recursive bisection core of PaToH. During the recursive bisection, after each bisection the nets that are in the cut are *split* into two nets in order to achieve correct accounting of the connectivity-1 cost metric. In PaToH, the default action for *size-1* nets is to discard them, since they cannot be in the cut for a future bisection. However, since our weight constraint is based on incident net weights, we have modified the code so that the sum of the weights of such size-1 nets are accumulated in a separate weight variable for each vertex. These additional weights need to be propagated during the coarsening phase – while computing the weight constraint in the initial partitioning and refinement phases, those weights are aggregated with the sum of the internal net weights to compute a part's incident net weight.

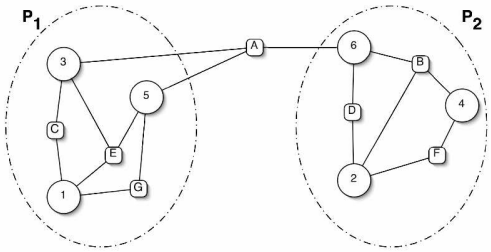
5.2 First-level Partitioning: Sub-batch Selection

We employ the hypergraph model together with the BINW partitioning to solve the sub-batch selection problem. In our hypergraph formulation, each task t_i is represented by a vertex v_i in the hypergraph. Each hyper-edge n_j represents a file f_j and connects the vertices that require this file as input. The expected execution time of the task t_i and the size of the file f_j are used as the weights of the vertex v_i and net n_j, respectively. An example batch of tasks and its hypergraph representation are illustrated in Figure 2.

Let $\Pi = \{V_1, V_2, ..., V_P\}$ be the P-way BINW partition of hypergraph H, where the weight constraint D is set to the aggregate available disk space of the compute cluster. Each partition obtained by the BINW partitioning corresponds to a sub-batch. Since the files that are required by each task is represented by nets connected to that task, the files required by the tasks of a partition V_i constitute the incident net set $I(V_i)$. By the definition of the BINW partitioning and the associated constraint (Eq. 24), we know that the total sum of the net weights (the sizes of the files) corresponding to files required by the partition V_i is less than D. Hence, all the files required by a sub-batch of tasks corresponding to V_i will fit into the aggregate disk space of the compute cluster. This essentially means that each sub-batch can execute on the compute cluster without disk space constraint violation. Minimizing the connectivity-1 metric corresponds to minimizing the number of times that a file is shared among the sub-batches thereby minimizing the I/O cost because of files shared among sub-batches.



a) A sample batch of tasks



b) Hypergraph representation

Figure 2. Hypergraph representation of a sample batch of tasks. The numbers indicate tasks. The letters are files required by the tasks.

5.3 Second-level Partitioning: Task Mapping

The second level of partitioning divides a sub-batch across the nodes of the compute cluster such that the remote data transfer cost is minimized while load balance across compute nodes is maintained. We model the problem of locality-aware load-balancing as a hypergraph partitioning problem. A task is represented by a vertex and a file by a net in the hypergraph. The expected execution time of tasks and the size of files are used as weights of respective vertices and nets.

The expected execution time of a task is calculated as the sum of I/O overhead (the transfer time of files either through remote transfer or replication plus the I/O time to read files from local disk) and the computation cost of the task. To employ an existing hypergraph partitioner without any modification, we use a probabilistic approach when computing the execution time $ExecT_i$ of task t_i as vertex weights in the partitioner. Let the set of files a task t_i needs be F_i and the number of compute nodes in the system be K. The cost of transferring one

byte of file f_j, Tr_j, for task t_i is equal to

$$Tr_j = \frac{Prob_{FNE}}{BW_s} + (1 - Prob_{FNE}) * \frac{(1 - Prob_{FE})}{\min(BW_s, BW_c)} \quad (25)$$

Here, BW_s is the minimum of I/O and network bandwidth between any storage and compute node pair, BW_c is the bandwidth between any two compute nodes of a cluster, $Prob_{FNE}$ 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 a uniform probability distribution, $Prob_{FNE} = \frac{1}{s_j}$ and $Prob_{FE} = \frac{s_j}{T} * \frac{1}{K}$. s_j denotes the number of tasks that share the file f_j and T denotes the number of tasks in the sub-batch. With the assumption that computation time is linearly correlated with the size of the input files, the estimated execution time of task t_i is computed as

$$ExecT_i = \sum_{f_j \in F_i} fsize(f_j) \times (Tr_j + \frac{1}{BW_\ell} + C) \quad (26)$$

where BW_ℓ is the I/O bandwidth from local disk on a compute node and C is the compute cost of one byte [10]. By assigning file sizes as hyper-edge weights and the estimated execution times as vertex weights, the proposed method reduces the task mapping problem to the K-way hypergraph partitioning problem according to the *connectivity-1* cutsize definition [5].

Note that the sub-batch which is given by the first level partitioner satisfies the aggregate disk space constraint on the compute cluster. A second level partitioning of such a sub-batch may lead to violation of disk space constraint on individual nodes of the cluster. To address this issue, we employ a simple heuristic. For each node of the cluster, we sort the list of files f_j to be staged onto it in the order of increasing s_j values where s_j is the number of tasks that share the file f_j. We remove files from this list in the specified order as long as the disk space requirements of the files in the list do not violate the disk space constraint. Finally, we remove any tasks assigned to this node if one or more of its files have been removed from the list. The tasks thus removed are then executed in subsequent batches.

6 Ordering of Tasks and Staging of Files

The proposed scheduling schemes exploit global task-file sharing information to minimize transfers of the same file multiple times. However, two tasks that are mapped to different compute nodes 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 such a way to minimize end-point contention on the storage cluster as well as the compute cluster. We use a strategy in which tasks allocated to each compute node 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 and compute nodes. For each file required by a task which is not present locally, the algorithm chooses to stage the file from one of its multiple possible sources so that the file transfer completion time is minimized. This is accompanied by reserving time slots on the selected source of the file as well as the destination node.

The earliest estimated completion time of a task t_i is computed as the sum of 1) the completion time of its file transfers, 2) the I/O time to read the files on local disk, and 3) the CPU time to process the files. If all of the input files are already in the compute node, then the completion time of file transfers is the current time. Otherwise, it is the completion time for the transfer of the last input file to the compute 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 and the actual transfer time (size of f_j divided by the bandwidth). For a remote transfer, the bandwidth is the minimum of remote disk bandwidth and network bandwidth, whereas for replications, the bandwidth is the node to node bandwidth in the compute cluster. The file f_j with the minimum TCT_j is selected and tentatively scheduled for transfer. The TCTs of the remaining input files are recomputed and the next file with the minimum TCT is selected and tentatively scheduled. This process is repeated until all of the input files are scheduled. The transfer completion time for the task is the TCT of the last file scheduled 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, from the other task groups (excluding the one containing t_i), the task with the minimum earliest completion time is selected and scheduled. When a running task completes, the task with the earliest completion time from that group is scheduled.

The proposed task ordering solution is applied as a subsequent step to the mapping approaches. Since the IP based solution also gives information about data staging (replications or remote transfers), we apply the dynamic scheduling strategy with a minor modification. For each file, the algorithm uses the current state of the Gantt chart and computes the transfer completion time for staging the file based on the IP based solution. It does not consider multiple options for staging the file; it just uses the solution provided by the IP based approach.

7 Experimental Results

We now present an experimental evaluation of the proposed strategies along with the MinMin algorithm with implicit replication and the *Job Data Present with Data Least Loaded* approach [13]. The proposed IP approach used a publicly available solver called **lp_solve** [2].

For evaluation, we used two application classes: satellite data processing and biomedical image analysis. To generate datasets for the satellite data processing application (referred to here as **SAT**), we employed an emulator developed in [15]. The application [7] operates on data chunks that are formed by grouping subsets of sensor readings that are close to each other in spatial and temporal dimensions. These chunks can be organized into multiple files. In our emulation, we assigned one data chunk per file. A satellite data analysis task specifies the data of interest via a spatio-temporal window. For the image analysis application (referred to here as **IMAGE**), we implemented a program to emulate studies that involve analysis on images obtained from MRI and CT scans (captured on multiple days as follow-up studies). An image dataset consists of a series of 2D images obtained for a patient and is associated with meta-data describing patient and study related information (in our case, we used patient id and study id as the meta-data). Each image in a dataset is associated with an imaging modality and the date of image acquisition, and is stored in a separate file. An image analysis program can select a subset of images based on a set of patient ids and study ids, image modality, and a date range.

We employed three different types of workloads; *high overlap*, *medium overlap*, and *low overlap*, representing different amounts of file sharing among tasks in a batch. For SAT, we simulated queries directed to geographically distant parts of the world. Four sets were generated, representing queries directed to 4 hot spot regions. Across the sets, there was no overlap between the queries, and in each set, queries were adjusted such that for high overlap workload, they resulted in 85% overlap on average, in terms of files requested by different tasks in the batch. Similarly, we generated medium and low overlap workloads with 40% and 10% overlap, respectively. For IMAGE, different degrees of overlap were 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, respectively.

We generated 20 days worth of data, about 50 GB for SAT. The data was distributed across the storage nodes using a Hilbert-curve based declustering method [9]. Each file in the dataset was 50 MB. In the high overlap case, each task accessed on an average 8 files. In the medium and low overlap cases, each task accessed on an average 14 files. For IMAGE, the 2 Terabyte dataset corresponded to a dataset of 2000 patients and images acquired over several days from MRI and CT scans. Each task on an average accessed 8 files. The sizes of images were 4 MB and 64 MB for MRI and CT scans, respectively. Images for each patient were distributed among all the storage nodes in a round robin fashion. The IMAGE and the SAT application typically involve computations equivalent to two floating point operations per word and this translates to a processing time of approximately 0.001s/MB of data in our test-bed.

Our experiments were carried out using two systems. The first 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, 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 (**XIO**), each of which is connected to an array of IBM FASTt600s over a Fiber Channel Switch [3]. Each node has a local file system that resides on FASTt600 storage units. The disk bandwidth available on these storage nodes is around 210 MB/sec. The second system employs the same compute cluster as the first but uses another cluster as a storage cluster – consisting of 933 MHz Pentium III nodes (**OSUMED**) equipped with 512MB of memory. These nodes are connected through a Switched 100 Mbps Ethernet. The disk bandwidth available on these storage nodes varies from 18 MB/sec to 25 MB/sec. The bandwidth of the shared link between the OSUMED and OSC clusters is around 100 Mbps.

Figures 3 and 4 show the relative performance of the various scheduling/replication schemes on workloads with different degrees of shared I/O among tasks. These experiments were conducted using 4 compute nodes and 4 storage nodes for both IMAGE and SAT. Both the IMAGE workload and the SAT workload consisted of 100 tasks each. As is seen from the figures, the IP based strategy and the BiPartition approach performs better than the other algorithms for all the cases. This is because the IP formulation is able to leverage the global task-file affinity information by incorporating it into its goal function of minimizing the batch execution time. In addition, while minimizing the I/O overheads, the IP approach also maintains computational load balance across the nodes. The BiPartition approach tries to cluster tasks that share files together, thereby attempting to avoid the transfer of the same file multiple times. In addition, the partitioning heuristic ensures load balance across nodes. The IP based approach performs slightly

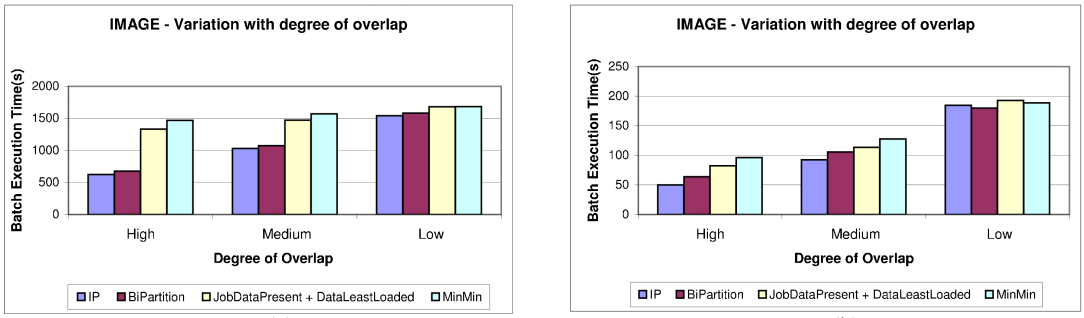


Figure 3. Batch execution time achieved by different algorithms on (a) OSUMED storage cluster and (b) XIO storage cluster, for the IMAGE application

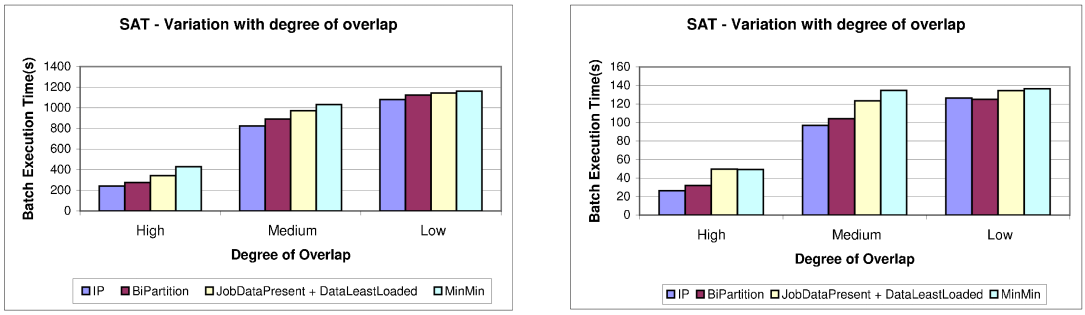
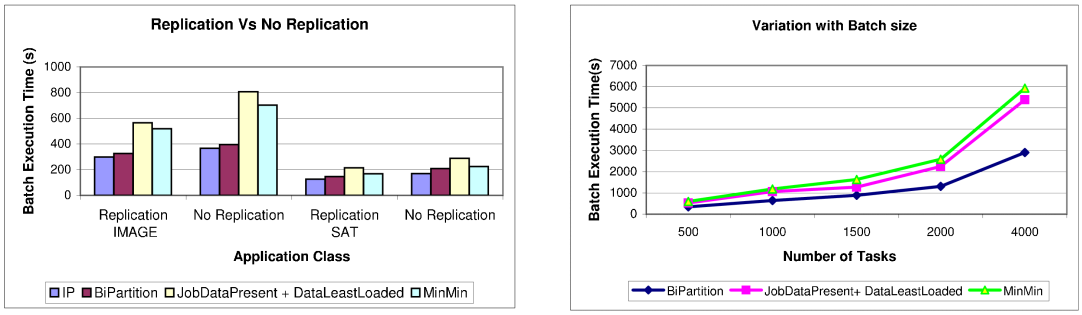


Figure 4. Batch execution time achieved by different algorithms on (a) OSUMED storage cluster and (b) XIO storage cluster, for the SAT application

better than the BiPartition approach because it solves the problem of scheduling and replication in an integrated fashion and thus it is able to explore a larger search space thereby achieving a better global solution. The benefit of the proposed approaches is greatest for the high overlap workload and reduces as the degree of overlap decreases, as expected. The base schemes do not perform as well as the two proposed approaches because they are greedy heuristics and hence make local decisions without exploiting the inter-task affinities arising out of file sharing. Among the base schemes, *Job Data Present* coupled with *Data Least Loaded* does better than *MinMin* with *Implicit Replication*. This is because the *Job Data Present* strategy favors data locality and hence is able to make good use of the data replication of popular datasets performed by the *Data Least Loaded*. For the low overlap case, the IP based scheme performs slightly worse than the BiPartition scheme for some of the experiments. In the low overlap case, end-point contention increases due to a significantly higher number of remote file transfers. In the IP approach, we obtain both the task mappings and file transfer information (where from and where to replicate) statically and then realize this solution at run-time. In BiPartition, only the task mappings are obtained statically and the decision of where to fetch the file from for each file transfer is made dynamically at run-time. The IP based scheme however does a better global modeling of the overall problem as compared to BiPartition since it captures the differential between the remote and replica access as well as the parallelism in the system. We conjecture that the effects of contention outweigh the advantage of better modeling of the IP approach when tasks do not have affinities among themselves.

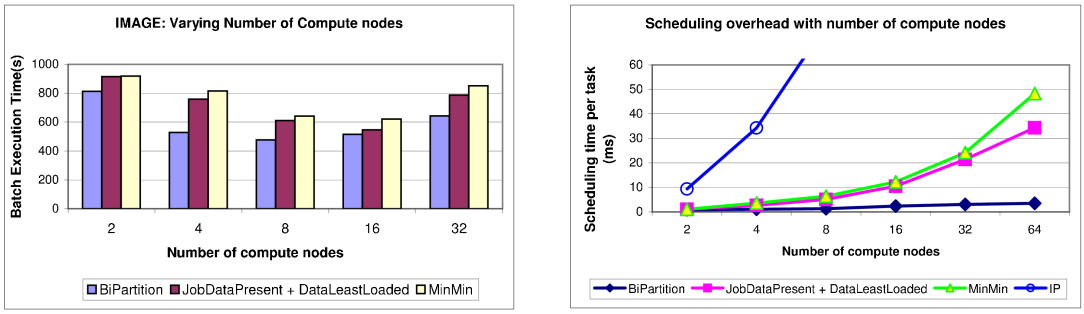
Figure 5(a) quantifies the benefit which can be obtained through replication. This experiment was conducted on 8 OSC compute nodes and 4 OSUMED storage nodes. The workloads employed for both application classes was 100 task high overlap batches. The *No Replication* in Figure 5(a) refers to the case when there is no replication. The results show that replication gives significant performance improvement because it exploits the choice of using one of many sources of a file thereby reducing contention on the storage cluster.

Figure 5(b) demonstrates how the proposed scheme and the base schemes perform with respect to varia-



(a) (b)

Figure 5. (a) Benefit of compute node to compute node data replication over no replication. (b) Variation of batch execution time with increasing batch size.



(a) (b)

Figure 6. (a) Performance of different algorithms for IMAGE with varying number of compute nodes. (b) Scheduling overhead with varying number of compute nodes.

tion in batch size. This experiment was conducted on 4 OSC compute nodes and 4 XIO storage nodes using a high overlap IMAGE workload. The number of tasks in the IMAGE workload was varied from 500 to 4000. The disk space on each machine of the compute cluster is 40GB. The aggregate data requirements of the batch vary from around 40GB for the 500 task batch to around 330GB for the 4000 task batch. Here, the aggregate data requirements refer to the total disk space required to store one copy of each file. Here, we only show results for the BiPartition approach and the base schemes because for such large batch sizes, the IP based scheme has significant scheduling overhead. The results show that as the batch size increases, the base schemes show a greater increase in the batch execution time. The BiPartition scheme does the best. This is expected, since the base schemes suffer a lot of file evictions on the compute cluster as the batch size increases and the disk space becomes a constraint. The BiPartition approach makes efficient allocations of tasks and files and therefore, suffers considerably lesser evictions.

To analyze the scalability of the proposed scheme with respect to the number of compute nodes, we ran experiments with a high overlap workload consisting of 1000 high overlap IMAGE tasks. These experiments were run using 8 storage nodes of the XIO cluster. The number of compute nodes was varied from 2 to 32. Figure 6(a) shows the results with varying number of compute nodes. As is seen from the figure, BiPartition achieves the best performance. An increase in the number of compute nodes is likely to increase contention on the storage nodes; hence the batch execution time increases at 32 compute nodes for all approaches. We observe that the volume of data transferred increases with increasing number of compute nodes since tasks are distributed across more nodes, thereby increasing the probability that two tasks sharing files will be mapped to different nodes.

Figure 6(b) shows the per task scheduling times (in milliseconds) for various schemes. The graph shows that the BiPartition scheme has very little scheduling overhead. The IP based scheme has high scheduling overhead for larger configurations, due to the exponential complexity of the search. The scheduling time of *MinMin* is higher than *Job Data Present*. This is because *MinMin* involves iterating over all task-host pairs at ev-

ery scheduling step. The *Job Data Present* scheme is a dynamic scheme and at each step picks up the next task from a queue and schedules it.

8 Conclusions

The paper developed two strategies for scheduling a collection of batch-shared data intensive tasks - one approach formulating the problem of coordinating scheduling and replication using a 0-1 Integer Programming and another using a bi-level hypergraph partitioning strategy. Both approaches also model disk storage space constraints at the compute cluster. The performance results show that our strategies achieve significant performance improvement over *MinMin* with *Implicit replication* and *JobDataPresent* with *Data Least Loaded*. The base schemes do not explicitly consider inter-task dependencies arising out of file-sharing and thus make local decisions based on greedy heuristics. Among the proposed algorithms, the IP formulation results in the best batch execution time. However, it suffers from high scheduling time. The BiPartition approach results in slightly longer batch execution times, but is much faster than the IP based approach. Our conclusion is that the IP based approach is attractive for small workloads, while the BiPartition approach is preferable for large scale workloads and system configurations.

References

[1] J. Bent, D. Rotem, A. Romosan, and A. Shoshani. Coordination of Data Movement with Computation Scheduling on a Cluster. In *Workshop on Challenges of Large Applications in Distributed Environments (CLADE2005)*, pages 25–34, Research Triangle Park, NC, July 2005. IEEE Computer Society Press.

[2] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve open source mixed integer linear programming system. In *Version 5.5.0.7*, May 2005. http://lpsolve.sourceforge.net/5.5/.

[3] S. Bokhari, B. Rutt, P. Wyckoff, and P. Buerger. An evaluation of the OSC FAStT600 Turbo storage pool. Technical Report OSUBMI_TR_2004_n02, Dept. of Biomedical Informatics, The Ohio State University, 2004. http://web.bmi.ohio-state.edu/resources/techreports/OSUBMI_TR_2004_n02.pdf.

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

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

[6] 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.

[7] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. H. Saltz. Titan: A high-performance remote sensing database. In *Proc. of the 13th International Conference on Data Engineering (ICDE 1997)*, pages 375–384, Washington, DC, USA, 1997. IEEE Computer Society.

[8] F. Desprez and A. Vernois. Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid. In *Workshop on Challenges of Large Applications in Distributed Environments (CLADE2005)*, pages 66–74, Research Triangle Park, NC, July 2005. IEEE Computer Society Press.

[9] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *Proc. of the 8th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS 1989)*, pages 247–252, New York, USA, 1989.

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

[11] S. Krishnamoorthy, U. Catalyurek, J. Nieplocha, and P. Sadayappan. An approach to locality-conscious load balancing and transparent memory hierarchy management with a global-address-space parallel programming model. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Performance Optimization for High-Level Languages and Libraries (POHLL)*, Rhodes Island, Greece, 2006. to appear.

[12] 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 *8th Heterogeneous Computing Workshop (HCW'99)*, pages 30–44, San Juan, Puerto Rico, 1999. IEEE Computer Society.

[13] K. Ranganathan and I. T. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC 2002), Edinburgh, UK*, pages 352–358, 2002.

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

[15] M. Uysal, T. M. Kurc, A. Sussman, and J. Saltz. A performance prediction framework for data intensive applications on large scale parallel machines. In *Proc. of the 4th Workshop on Languages, Compilers and Runtime Systems for Scalable Computers, Lecture Notes in Computer Science, Vol. 1511*, pages 243–258. Springer-Verlag, May 1998.