# Scheduling File Transfers for Data-Intensive Jobs on Heterogeneous Clusters

GAURAV KHANNA, UMIT CATALYUREK, TAHSIN KURC, P. SADAYAPPAN AND JOEL SALTZ

Technical Report
OSU-CISRC-1/07-TR05

# Scheduling File Transfers for Data-Intensive Jobs on Heterogeneous Clusters *

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

**Abstract**

This paper addresses the problem of efficient collective scheduling of file transfers requested by a batch of tasks. Our work targets a heterogeneous collection of storage and compute clusters. The goal is to minimize the overall time to transfer files to their respective destination nodes. Two scheduling schemes are proposed and experimentally evaluated against an existing approach, the Insertion Scheduling. The first is a 0-1 Integer Programming (IP) based approach which is based on the idea of time-expanded networks. This scheme achieves the minimum total file transfer time, but has significant scheduling overhead. To address this issue, we propose a maximum weight graph matching based heuristic approach. This scheme is able to perform as well as the insertion scheduling approach and has much lower scheduling overhead. We conclude that the heuristic scheme is a better fit for larger workloads and systems.

## 1 Introduction

Several scientific applications (e.g., biomedical imaging, satellite data processing) involve data-analysis tasks that analyze the huge volumes of data generated via emerging experimentations and simulations. These simulation datasets typically reside on storage archives which may be centralized or distributed across multiple individual storage sites. Scientists perform data analysis by downloading subsets of these large datasets. Simulations are then executed on "local" resources to produce additional scientific data. However, unlike traditional compute intensive tasks, data analysis tasks may require access to large numbers of files and high data volume.

Efficient execution of such data-intensive tasks is essentially a two phase process which involves addressing two key problems. The first one involves the mapping of tasks to nodes such that the volume of overall data transfer is minimized. In other words, task mapping needs to be locality conscious. The second one involves the scheduling of file transfers by transferring them from one of their multiple possible sources to the destination nodes where the tasks requesting those files have been mapped. This staging of files should be carefully coordinated to minimize

*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.

the contention while accounting for the topology with which the nodes are connected and the heterogeneity of network and I/O bandwidths in the system.

In our past work, we have looked at the problem of scheduling a batch of tasks exhibiting shared I/O behavior on homogeneous clusters [19]. We modeled the task-file sharing patterns using a hypergraph and employ hypergraph partitioning to get a load-balanced cut-minimized partitioning of tasks onto compute nodes. We have extended this work along multiple directions [18], [17], [24]. Naga et al. [24] proposes a K-way iterative mapping refinement heuristic to schedule a batch of data sharing tasks onto a collection of heterogeneous clusters. In a very recent work, we have also addressed the data-intensive task scheduling problem in a more generic online context where a set of data-intensive tasks arrive over time and the scheduling needs to be done so as to minimize the average response time [17]. We extend the aforesaid hypergraph partitioning based approach by also taking into account the fact that prior task executions would have created replicas of files on certain nodes and the future arriving tasks need to exploit this node to file locality information for better task placement decisions. In all these prior works, our prime focus has been to address the first phase of the overall problem that is to accomplish task mapping by taking into account the global task-file affinity information.

This paper addresses the efficient collective scheduling of a set of file transfers requests made by a batch of data-intensive tasks exhibiting batch-shared I/O behavior [22] on a heterogeneous set of storage and compute clusters. In other words, it focuses on the second phase of the overall problem. Batch-shared I/O simply means that the same file may be required by multiple tasks in a batch. Each task may request multiple files which need to be staged onto the node on which the task has been allocated. A file can be staged on a node by accessing one of the existing replicas of the file which could be present either on the storage cluster or one of the compute clusters. This should be done in a way to minimize the contention in the network or the end-points. The target environment consists of a heterogeneous collection of compute clusters connected over switched/shared network(s) to one or more storage clusters with different I/O bandwidths. We expect that such configurations will increasingly be common in supercomputing centers as the capacity of commodity disks continues to increase and their cost per gigabyte to decrease. The topology of the network and the heterogeneity of the system govern the usefulness of each file replica for the purpose of data staging. The problem therefore amounts to scheduling of a set of file transfers to their respective destinations so as to minimize the overall completion time of file transfers.

We propose two approaches to solve the file transfer scheduling problem. The first approach formulates the problem using 0-1 Integer Programming by employing the idea of flows over time and the concept of time-expanded networks [10]. The second approach employs max-weighted graph matching to yield a schedule which tries to minimize contention and maximize the parallelism in the system. We carry out an experimental evaluation of these algorithms, comparing them against our previously proposed Insertion scheduling based heuristic [19]. Application emulators from two application domains are used - analysis of remotely sensed data and biomedical imaging.

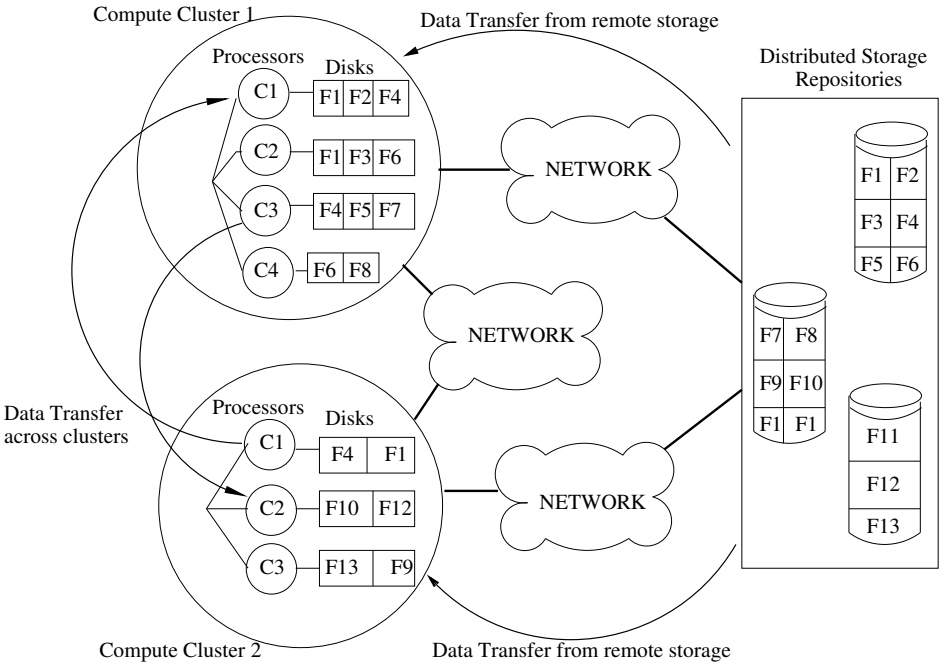



Figure 1: Scheduling problem.

# 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 node where the task is allocated for the task to execute correctly. The tasks in the batch may share a number of files. Therefore, a file may be required to be transferred to multiple different nodes. A file could be staged to a node by transferring it from one of its multiple possible replicas. In other words, it may be retrieved either from remote storage systems or from another compute node which already has the file. We assume a single port model wherein multiple requests to the same node are serialized and that a node can receive a file after it has finished storing the previously received file on local disk. Formally, the scheduling problem can be stated as follows:

- Given a network topology which is basically a graph $G = (V, E)$ where $V$ represents the nodes in the system and $E$ represents the edges connecting them. The overall topology may encompass multiple heterogeneous storage/ compute clusters but we hide the distinction between nodes belonging to different clusters by abstracting them as vertices in the overall platform graph. We assume that the Graph $G$ is connected which means that there is a path between every pair of nodes possibly spanning multiple other nodes. Furthermore we employ a store and forward model of file transfer which implies that if a file $f_\ell$ needs to be transferred from a node $v_i$ to a non-adjacent node $v_j$, then the file would be routed along one of the multiple possible paths between $v_i$ and $v_j$. In our model, a copy of the file is left in each intermediate node thereby increasing the number of replicas of each file leading to potentially higher parallelism for other requests.

- The input to the system is a file transfer request set represented by the set $R = \{< f_\ell, v_i >\}$, which means that the file $f_\ell$ needs to be transferred to the node $v_i$
- The scheduling system also has access to an initial mapping of files onto the nodes. The mapping of files to nodes (storage and/or compute nodes, if the file has been replicated on a compute node for a previous batch) is represented by the set $D = \{< f_\ell, v_j >\}$, which means that the file $f_\ell$ is initially present on the node $v_j$.

*Our objective is, given a set of file transfers where each file request is a two tuple $< f_\ell, v_i >$ consisting of a file id and a destination node, and a network topology which is basically a graph $G = (V, E)$ where $V$ represents the nodes in the system and E represents the edges connecting them, to find the complete schedule which comprises of a set of four tuples $< v_i, v_j, f_\ell, t >$, each tuple consisting of a source node, a destination node, a file id corresponding to the file being transferred and the file transfer start time. This has to be accomplished with the overall goal of minimizing the overall file transfer completion time.*

Fig 1 shows an illustration of the problem with two compute clusters and a set of distributed storage repositories. The figure shows the existence of multiple replicas of each file at different locations either of which can be used to stage the file to its associated destinations.

We have evaluated our approach using application scenarios from two application classes; analysis of remote sensing data and biomedical image analysis: (1) **Satellite data processing.** Remotely sensed data is either continuously acquired or captured on-demand via sensors attached to satellites orbiting the earth [5]. 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

Jain et.al. [16] 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. However, they consider a very simplistic model where all files are of the same size and all transfers take a unit time. Our work is in the context of a generic platform consisting of a set of heterogeneous sites with different interconnection bandwidths and does not have such assumptions on file sizes or transfer times.

Kosar et al. [20] propose STORK, a specialized scheduler for data placement activities on the Grid. The scheduler allows check-pointing and monitoring of data transfers as well as use of DAG schedulers to encapsulate dependences between computation and data movement. In this paper,

we focus on modeling the system topology and heterogeneity as well as the global information of a set of file requests to make efficient collective file transfer scheduling decisions. Therefore, our work is complementary to STORK and can be applied in conjunction with it. The work of Giersch et al. [12], [13] addressed the problem of scheduling a collection of tasks sharing files onto heterogeneous clusters. Their work focused mainly on task mapping and they proposed extensions to the well-known MinMin heuristic [15] to lower the cost of scheduling while achieving scheduling quality (i.e., batch execution time) similar to that of MinMin. In this paper, we focus on the file transfer scheduling as opposed to task mapping and propose efficient ways to do it.

GridFTP [2] is a widely used protocol which enables secure, reliable and high performance data movement. It facilitates efficient data transfer between end-systems by employing techniques like multiple TCP streams per transfer , striped transfers from a set of hosts to another set of hosts and partial file transfers. SRB [3] is a system which provides a uniform interface to access distributed, heterogeneous storage resources. The storage resources could be anything ranging from filesystems, databases to archival storage devices like tapes. The scheduling algorithms proposed in this work can act as middlewares which lie between SRB and GridFTP in the hierarchy of the overall data transfer mechanism. In other words, a scheduling system accepts a bunch of requests from a user-interacting system like SRB and sends commands to a lower level protocol like GridFTP to perform the data transfers.

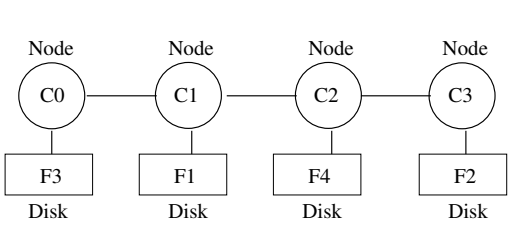
# 4 Problem Complexity

**Definition 1** *The chromatic index of a graph $G = (V, E)$ is defined as the minimum number of colors required to color the edges of the graph such that no two adjacent edges have the same color.*

**Theorem 1** *Given an arbitrary graph $G = (V, E)$, determining its chromatic number is $NP - complete$ [14].*

**Theorem 2** *The optimization problem defined in Section 2 is $NP - complete$.*

**Proof** We prove the theorem by considering a simplified version of the problem where all files are of the same size and all the links have same bandwidth. Furthermore, let us assume that each file transfer takes one unit of time to finish. We also assume that for every input request tuple $< f_\ell, v_i >$, we have determined the path of file transfer from one of the multiple possible sources of the file to the destination node. Let $Path_{\ell i}$ denote the file transfer path of the tuple $< f_\ell, v_i >$ in the set of pending input requests. $Path_{\ell i}$ consists of a set of nodes $v_{i1}$ ,$v_{i2}$ .... $v_{ik}$ where the first node in the path $v_{i1}$ is one of the multiple possible sources of file and the last node is the respective destination node for the corresponding tuple in the input request set. The optimization problem is to schedule the set of file transfers for each input request to minimize the total file transfer completion time. An edge coloring of a file transfer graph, where each edge of the graph represents a file transfer, is equivalent to generating the entire schedule since edges with the same color can start and finish at the same time. The makespan of the schedule, therefore, equals the chromatic number of the file transfer graph. In other words, the aforesaid simplified version of the




Topology of the system

| Fileid | Destination |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 2 | 0 |
| 4 | 1 |
| 3 | 1 |
| 3 | 2 |

File Transfer Request Set

Figure 2: Topology of the system and the file transfer request set.

problem is equivalent to finding the chromatic number of a graph. Finding the chromatic number of an arbitrary graph $G = (V, E)$ has been proven to be $NP - complete$ [14]. Therefore, the given problem is also NP-complete. Since the simplified instance of the problem is NP-complete, the general problem is also $NP - complete$.

# 5 Scheduling Schemes

In Sections 5.2 and 5.3, we talk about our proposed schemes for scheduling. Section 5.2 proposes a 0-1 integer programming formulation of the scheduling problem. Section 5.3 proposes a graph matching based heuristic for the file transfer scheduling problem. In the section 5.1, we discuss a previously proposed scheduling scheme to solve the aforesaid scheduling problem.

## 5.1 Insertion Scheduling Based Approach

Giersch et al. [12] employ an insertion scheduling scheme to schedule file transfers. In our past work [19], we developed a Gantt chart based heuristic based on a similar idea which is applied in the conjunction with the task mapping schemes. The basic idea was to memorize the duration and the start time of file transfers for each link and use this information to generate schedules for pending requests.

The transfer completion time to transfer a file $f_\ell$ from a node $v_i$ to a node $v_j$, $TCT_{\ell ij}$ is estimated as the sum of the earliest time a transfer can start and the actual transfer time (size of $f_\ell$ divided by the bandwidth of the link between $v_i$ and $v_j$). At each step, the algorithm chooses a fileid, destination pair $< f_\ell, v_k >$ and schedules the transfer of file $f_\ell$ to the node $v_k$ . To accomplish this, it finds the minimum transfer completion time $TCT$ of each file in the input request set on its respective destination node and among them chooses the $< f_\ell, v_k >$ pair with the minimum completion time. This is accompanied by reserving time slots on the selected source of the file as well as the destination node. This process is then repeated until all the files have been scheduled to their respective destinations. Note that this is the same principle as applied in MinMin.

Fig 2 shows an example instance of the scheduling problem. It shows four sites connected to each other in the form of a linear topology. Each of the sites has one file initially available on it. The file transfer request set shows a set of file requests and their respective destinations. We

assume that the transfer time of each file on any of the links is 1 unit. Fig 3 (a) shows the schedule obtained by running the Insertion scheduling algorithm on the aforesaid problem instance. The file transfer completion time obtained is 7 units of time.

### 5.1.1 Complexity Analysis of Insertion Scheduling:

Consider the topology graph $G = (V, E)$ representing the system. The input is a file transfer request set which comprises of a set of two tuples. Each two tuple $< f_\ell, v_i >$ denotes a fileid and the corresponding destination node. The computation of the minimum transfer completion time of a file on a destination node in a general topology network requires running a variant of Dijkstra's shortest path algorithm to find which one of the multiple possible sources to stage the file from. We modify the Dijkstra's algorithm suitably to take into account the wait times of the source and the destination nodes as well as the bandwidth of the links. For a file $f_\ell$ under consideration, the Dijkstra's algorithm needs to be run from each of the nodes already containing the file by choosing them as source nodes for the algorithm. The complexity of Dijkstra's shortest path algorithm is $O(|E| + |V|log(|V|))$. The complexity of finding the minimum transfer completion time of a file $f_\ell$ is therefore, equal to $O(|V| \times (|E| + |V|log(|V|)))$. To find the $< fileid, destination >$ pair with the best transfer completion time, the computation of minimum transfer completion time has to be done for all the pending file requests. Therefore, the complexity of each step in the algorithm is $O(|R| \times |V| \times (|E + |V|log(|V|)))$. Finally, the complexity of the insertion scheduling algorithm is $O((|R|^2) \times |V| \times (|E| + |V|log(|V|)))$.

## 5.2 0-1 Integer Programming-based Approach

In the following discussion we use subscripts $i$ and $j$ for nodes, $e$ for edges, $\ell$ for files and $t$ for time. We represent time in discrete units and the smallest unit of time represents the least time taken to transfer a file from a source node to a destination node among all files and node pairs. Before we present the IP formulation, we briefly discuss the concept of the time-expanded network [10] and its construction in the context of our problem. Time-expanded networks have been defined in the context of network flows over time. The simplest version of a flows over time problem involves a network with capacities and transfer times assigned to its edges and the goal is to push the maximum amount of flow from a source node to a sink node within a given time $T$. A time-expanded network captures the temporal aspects of this problem in such a manner that flows over time in the original network can be treated as just flows in the time-expanded network. Our problem has a similar flavor to a multi-commodity flow over time problem in that it involves transfer of multiple different files from multiple sources to multiple destinations so as to minimize transfer completion time.

**Construction of the time-expanded network in our context.** Let $F$ is the set of files belonging to the file transfer request set $R$. Let $T^*$ denote the upper bound on the total completion time of all the file transfers. For each file $f_\ell \in F$, we construct a time expanded network $G'_l = (V'_l, E'_l)$ as follows. For each node $v_i \in V$ and each time $t = 0, ..., T^*$, we add a vertex $v_{it}$ to the graph $G'_l$. For each undirected edge $e = \{v_i, v_j\}, e \in E$ connecting any two nodes $v_i$ and $v_j$, $Time_{lij}$ represents the transfer time of file $f_\ell$ on the link $e = \{v_i, v_j\}$. For each edge $e = \{v_i, v_j\}$ and

| | F1 | F2 | F3 | F4 |
|---|---|---|---|---|
| t=0 | 1→2 | | | |
| t=1 | 2→3 | | 0→1 | |
| t=2 | | | | 2→1 |
| t=3 | | | 1→2 | |
| t=4 | | 3→2 | | |
| t=5 | | 2→1 | | |
| t=6 | | 1→0 | | |

a→b denotes the file transfer from node a to node b

(a)

| | F1 | F2 | F3 | F4 |
|---|---|---|---|---|
| t=0 | | 3→2 | 0→1 | |
| t=1 | 1→2 | | | |
| t=2 | 2→3 | | | |
| t=3 | | 2→1 | | |
| t=4 | | 1→0 | | |
| t=5 | | | | 2→1 |
| t=6 | | | 1→2 | |

a→b denotes the file transfer from node a to node b

(b)

Figure 3: (a) Schedule obtained by Insertion scheduling. and (b) Schedule obtained by the matching based approach.

time instants $t$ and $t'$ such that $t' = t + Time_{lij}$, we add the directed edge $(v_{it}, v_{jt'})$ to the time expanded network $G'_l$ if $t' \leq T^*$.

### 5.2.1 0-1 IP formulation

The objective function of the IP formulation is to the minimize the overall file transfer time under a set of constraints. It solves for the following set of variables. Let $X_{\ell it}$ be a binary variable where $X_{\ell it} = 1$, if file $f_\ell$ is available on node $v_i$ at time $t$, and 0 otherwise. Let $Y_{\ell e}$ be a binary variable where $Y_{\ell e} = 1$, if the edge $e$ in the time expanded network $G'_l$ is used to transfer the file $f_\ell$, and 0 otherwise. Let $Busy_t$ be a binary variable where $Busy_t = 1$, if there is a file transfer which is finished at time $t$ or a later point in time.

**Constraints**

At t=0, certain files are present on certain nodes.

$$(\forall \ell)(\forall i, < f_\ell, v_i > \in D) X_{\ell i0} = 1 \tag{1}$$

Let $I_{\ell it}$ be the set of directed edges incident on the node $v_{it}$ in the time-expanded network $G'_l$. Let $O_{\ell it}$ be the set of directed edges outgoing from the node $v_{it}$ in the time-expanded network $G'_l$.

$$I_{\ell it} = (\forall j, j \neq i)(\forall t', t' \leq t)(e = (v_{jt'}, v_{it}), e \in E_{exp,l}) \tag{2}$$

$$O_{\ell it} = (\forall j, j \neq i)(\forall t', t' \geq t)(e = (v_{it}, v_{jt'}), e \in E_{exp,l}) \tag{3}$$

A file $f_\ell$ is present on a node $v_i$ at time t either if its already present on the node at time $t-1$ or due to the file transfer of the file $f_\ell$ to the node $v_i$ from one of the nodes $v_j$ already having the file such that the file transfer is finished at time $t$.

$$(\forall \ell)(\forall i)(\forall t) X_{\ell it} = (\sum_{(\forall e, e \in I_{\ell it})} Y_{\ell e}) + X_{\ell it-1} \tag{4}$$

At time $t = T^*$, each file must be present at its respective destination nodes.

$$(\forall \ell)(\forall i, < f_\ell, v_i > \in R) X_{\ell i T^*} = 1 \tag{5}$$

A file $f_\ell$ can be transferred from the node $v_i$ at time $t$ only if its present on the node $v_i$ at time $t$ . In addition, atmost one outgoing arc is allowed from a node $v_i$ at time $t$ .

$$(\forall \ell)(\forall t)(\sum_{(\forall e, e \in O_{\ell it})} Y_{\ell e}) \leq X_{\ell it} \tag{6}$$

A file $f_\ell$ once staged to a node $v_i$ remains available on the node.

$$(\forall \ell)(\forall i)(\forall t) X_{\ell it} \leq X_{\ell it+1} \tag{7}$$

The aforementioned constraints are defined for each of the time-expanded network corresponding to each unique file. The interaction between the different time-expanded networks comes from the following capacity constraints.

Each node $v_i$ can be involved in atmost one send or receive at a time $t$. Let $C_{\ell it}$ be the set of all incoming and outgoing arcs of the time-expanded network $G'_l$ that would make the node $v_i$ busy during the time $[t, t+1)$. Note that this includes all arcs that start at time $t' \leq t$, end at a time $t' \geq (t+1)$, and having $v_i$ as its source or target node.

$$(\forall t)(\forall i)(\sum_{(\forall \ell)(\forall e, e \in C_{\ell it})} Y_{\ell e}) \leq Busy_t \tag{8}$$

**Objective function.**

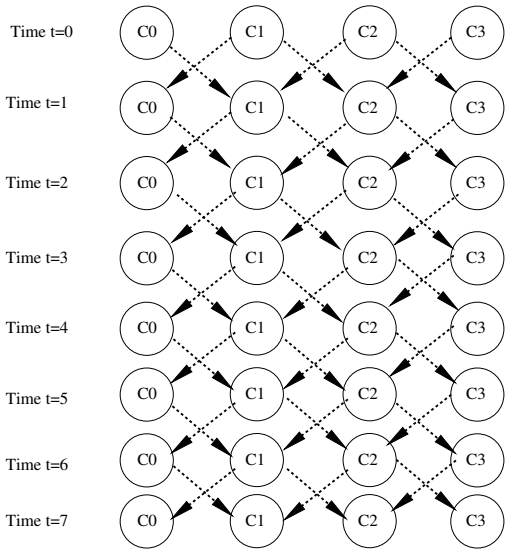
The objective is to minimize the total file transfer time $FileTransferTime$.

$$FileTransferTime = \sum_{(\forall t)} Busy_t \tag{9}$$

The objective function is such that the network may be busy for, say, 5 time steps with $Busy_1 = ... = Busy_5 = 1$, be idle for the next 10 time steps, $Busy_6 = ... = Busy_{15} = 0$, and finishing the transfer in the next 2 time steps, $Busy_{16} = Busy_{17} = 1$. This would lead to objective value 7, which is seemingly wrong since the network is busy even at time $t = 17$. To address this problem, we introduce the following constraint.

$$(\forall t) Busy_t \geq Busy_{t+1} \tag{10}$$

The IP formulation effectively exploits the global information comprising of the files to be transferred and the topology of the network to yield the entire schedule. Figure 4(a) shows the directed time-expanded network for file $F1$ for the problem instance of Figure 2. Since the overall file transfer completion time obtained by the Insertion scheduling heuristic is 7 time units, therefore we set $T^*$ to be 7. The IP formulation gives an overall completion time of 6 units as shown by the resultant schedule in the Figure 4(b).




(a)

| | F1 | F2 | F3 | F4 |
|---|---|---|---|---|
| t=0 | | 3 → 2 | 0 → 1 | |
| t=1 | 1 → 2 | | | |
| t=2 | | | | 2 → 1 |
| t=3 | | 2 → 1 | | |
| t=4 | 2 → 3 | 1 → 0 | | |
| t=5 | | | 1 → 2 | |

a → b denotes the file transfer from node a to node b

(b)

Figure 4: (a) Time expanded Network for file F1. and (b) Schedule obtained by the IP based approach.

## 5.3 Max-Weighted Matching Based Scheduling Scheme (MMSS)

The MMSS is an iterative, dynamic algorithm and employs max-weighted matching heuristic as illustrated in Algorithm 1. For a given undirected graph $G = (V, E)$, we define the set $M \in E$ as a matching of Graph $G$, if no two edges in $M$ have a common vertex. For edge-weighted graphs, the weight of the matching is the sum of the weights of the edges which form the matching. A maximum weighted matching is defined as the matching of maximum weight.

The proposed scheduling algorithm is a dynamic scheduling algorithm that uses max-weighted matching as a building block to realize the schedule. The input as mentioned in Section 5.1 is a file transfer request set which comprises of a set of two tuples . Each two tuple $< f_\ell, v_i >$ denotes a $< fileid, destination >$. Algorithm 1 outlines the matching heuristic. The goal of the algorithm is to minimize the overall file transfer completion time.

The algorithm proceeds in iterations. In each iteration, the algorithm creates a file transfer graph $G' = (V, E')$ whose vertices $v' \in V$ correspond to the nodes in the system and whose edges $e'$ correspond to file transfers. Each input request can possibly consist of multiple hops, i.e., a set of intermediate nodes can be used to transfer the file to its final destination. An input request $< f_\ell, v_i >$ is considered as pending, if the file $f_\ell$ is not yet present on the node $v_i$. For each such pending request, the algorithm computes the path $Path_{\ell i}$ of file transfer which yields the minimum transfer time for the file $f_\ell$ onto the node $v_i$. This step requires running a variant of Dijkstra's shortest path algorithm on the graph $G$ to find which one of the multiple possible sources to stage the file from. We modify the Dijkstra's algorithm to take into account the wait times of the source and the destination nodes as well as the bandwidth of the links. The file transfer corresponding to the first hop of the path $Path_{\ell i}$ is then added as an edge to $G'$ between the corresponding pair of vertices in $G'$. Note that for a multi-hop request, the first hop changes with time as the file gets closer to its destination node. Since multiple file transfer requests can be

associated with the same source and destination node, therefore, each pair of vertices in the file transfer graph can possibly have multiple edges between them. The weight of an edge in the file transfer graph corresponding to an input request is $\frac{1}{TCT}$ where $TCT$ is the expected minimum completion time of the request. The idea behind this weight assignment is to give higher priority to file transfers which can finish early. Finally, the algorithm employs max-weighted graph matching on the file transfer graph to obtain a set of non-contending ready file transfers and schedules them. This procedure works iteratively until all the file transfers have been scheduled.

**Algorithm 1** Maximum Weighted Matching based Scheduling Heuristic

**Require:** Topology denoted by $G = (V, E)$ and an input request set consisting of $< f_\ell, v_i >$ pairs

1: **while** there exists a pending request **do**
2: **for** each pending request $< f_\ell, v_i >$ **do**
3: Run the Modified Dijkstra's algorithm. Let $Path_{\ell i}$ denote the file transfer path which yields the earliest completion time for the request $< f_\ell, v_i >$.
4: Create a file transfer graph $G' = (V', E')$ as follows.
5: **for** each pending request $< f_\ell, v_i >$ **do**
6: Let nodes $v_{i1}$ and $v_{i2}$ comprise the first hop of the file transfer path $Path_{\ell i}$.
7: $V' = V' \cup \{v_{i1}, v_{i2}\}$.
8: Add an edge with weight $\frac{1}{TCT}$ between $v_{i1}$ and $v_{i2}$ in $G'$. Here, $TCT$ denotes the minimum completion time of the request
9: Run the Max-weighted matching algorithm on the Graph $G'$ to get a Matching
10: Schedule the chosen set of edges belonging to the Matching

Fig 3(b) shows the schedule obtained by running the matching based algorithm on the aforesaid problem instance. The file transfer completion time obtained is 7 units of time.

### 5.3.1 Complexity Analysis of the Matching Heuristic:

Edmonds et al. [7] proposes a $O(|V|^4)$ algorithm for finding maximal matchings in graphs. We employ Gabow's implementation of the Edmond's algorithm for computing maximal matching on graphs [11]. The complexity of the Gabow's implementation is $O(|V|^3)$.

Before we go furthur, we analyze an iterative graph matching procedure for a Graph $G$. The analysis is used to compute the run-time complexity of the matching based scheduling approach.

Algorithm 2 shows an iterative matching algorithm which at each step, chooses a different set of edges of a Graph $G$ such that those edges constitute a matching and marks all those edges. The set of edges $e \in E$ chosen at each step constitute disjoint sets.

**Theorem 3** *Given a Graph $G = (V, E)$ with atmost one edge for every vertex pair , the number of times a maximal matching algorithm needs to be run on the Graph $G$ in order to cover all its edges is $O(|V|)$. In other words, the number of iterations of the while loop in Algorithm 2 is $O(|V|)$.*

**Proof** Consider a specific instance where the Graph $G = (V, E)$ is a clique. A clique consists of $\frac{|V| \times (|V|-1)}{2}$ edges. The set of edges of a clique is basically the union of $|V|-1$ sets of $\frac{|V|}{2}$ edges each

**Algorithm 2** Iterative Matching

**Require:** Graph $G = (V, E)$

1: Unmark all the edges of the graph $G$
2: **while** There exists an edge $e \in E$ which is unmarked **do**
3: Create a Graph $G' = (V', E')$ where $V' = V$ and $E'$ consists of edges $e \in E$ such that e is unmarked
4: Run a maximal matching algorithm on the Graph $G'$ to get a Matching $M$
5: Set the chosen set of edges which constitute the Matching $M$ as marked

such that for each edge set of size $\frac{|V|}{2}$, no two edges belonging to it share a common vertex. Each such edge set, therefore, constitutes a matching. Each step of the algorithm 2, therefore, chooses $\frac{|V|}{2}$ edges which form a matching. Since the number of such edge sets is $|V| - 1$, therefore, the number of steps required to cover all its edges is $|V| - 1$. For general graphs $G = (V, E)$, where $E \leq |V|^2$, the complexity can therefore be atmost $O(|V|)$.

**Corollary 1** *Given a Graph $G = (V, E)$ with atmost $|K|$ edges for every vertex pair , the number of times a maximal matching algorithm needs to be run on the Graph $G$ in order to cover all its edges is $O(|K| \times |V|)$.*

For a file transfer input request set $R$ and a platform graph $G = (V, E)$, each input request can atmost require $O(|V|)$ hops in the platform graph. Therefore, the total number of file transfer edges can atmost be $O(|R| \times |V|)$. In the worst case, each file transfer request can involve the the transfer of files through the same set of edges in the graph $G$. Therefore, there can be atmost $O(|R|)$ file transfer request edges between any pair of vertices in the worst case. By applying the aforesaid theorem and the corollary, we can prove that the matching algorithm is run atmost $O(|R| \times |V|)$ to cover all the file transfer request edges thereby obtain the complete schedule. Therefore, the worst case complexity of the matching based scheduling heuristic is $O((|R|) \times (|V|^4))$.

The number of file transfer requests $|R|$ is typically orders of magnitude higher than the number of vertices $|V|$ in the platform graph $G$. Therefore, in practice, the matching based heuristic is expected to perform much faster as compared to the Insertion scheduling approach explained in Section 5.1.

# 6 Experimental Results

In this section, $IP$ refers to the integer programming approach proposed in Section 5.2, $Matching$ refers to the graph matching based approach proposed in Section 5.3 and $Insertion$ refers to the insertion scheduling approach explained in Section 5.1. We implement another heuristic which acts as a baseline scheduling scheme for comparison. The heuristic is called as $Indep\_Local$ and is a relatively simpler scheduling scheme where each destination node knows the the set of files it needs and makes requests for each of them one by one. The destination nodes acting as clients do not interact with each other before making their respective requests and the centralized scheduler ensures that contending requests are serialized.

The proposed IP approach used an optimization technique [9] in conjunction with the well known solver called ILOG-CPLEX [1] available through the NEOS Optimization Server [6]. For

the purpose of the experiments, the upper bound on the overall file transfer completion time $T^*$ was set to be value obtained by the $Matching$ approach. For fair comparison, the scheduling time of the Integer Programming approach equals the sum of the time taken by the $Matching$ approach and the time taken by the IP solver. Since the feaspump solver gives feasible solutions which need not be optimal, therefore we apply binary search in conjunction with the solver to get the optimal value of the objective function.

For evaluation, we compared the performance of the various scheduling schemes under a varying set of scenarios covering multiple job-file sharing patterns and different topologies. We consider three different kinds of topologies, namely fully connected topologies which can be composed with Infiniband fabrics, bipartite topologies and random topologies. For the workloads, we employ both randomly generated workloads as well as workloads derived from 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 [23]. The application [5] 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 **IA**), 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.

For SAT, the 250GB dataset was distributed across the storage nodes using a Hilbert-curve based declustering method [8]. Each file in the dataset was 50 MB. For IA, the 1 Terabyte dataset corresponded to a dataset of 2000 patients and images acquired over several days from MRI and CT scans. The sizes of images were 10 MB and 100 MB for MRI and CT scans, respectively. Images for each patient were distributed among all the storage nodes in a round robin fashion.

To generate the input file request set for the two application domains, we apply the hypergraph partitioning based task-mapping technique [19] to map a batch of tasks onto a set of compute nodes. Since each task is associated with a set of files it needs, therefore, the task mapping provides information about the respective destination nodes for each file. Moreover, since the task partitioning is locality conscious, therefore, the number of different destination nodes for each file is very low.

We conducted our experiments using a memory/storage cluster at the Department of Biomedical Informatics at the Ohio State University. The cluster consists of 64 nodes with an aggregate 0.5 TBytes of physical memory and 48TB of disk storage. These nodes are connected to each other through Infiniband. We also used simulations to understand the performance of the various scheduling schemes on different topologies. We ran our simulations using the **Simgrid Toolkit** [4, 21]. This toolkit implements event-driven simulation of applications on heterogeneous distributed systems. It models a resource by two performance characteristics: latency (time to

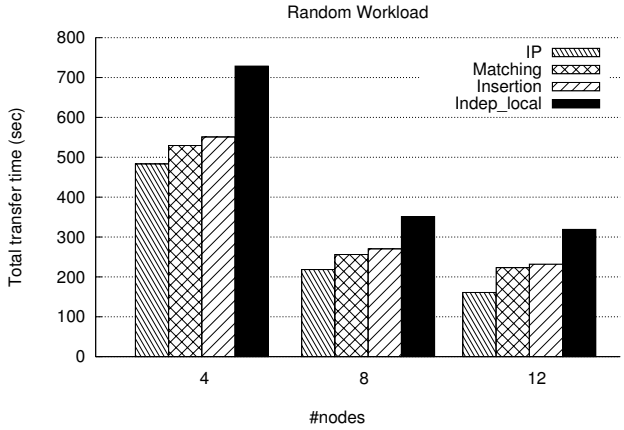
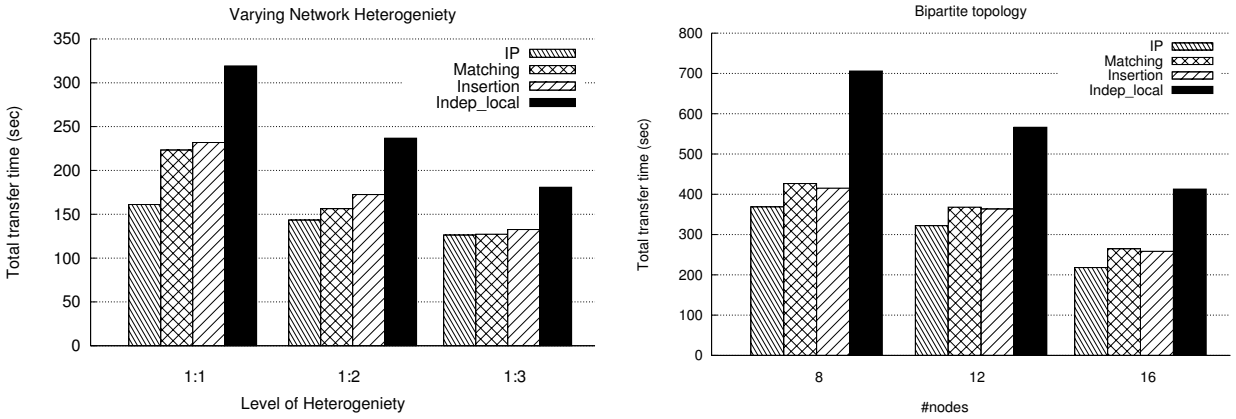



Figure 5: Performance of all schemes for a randomly generated workload



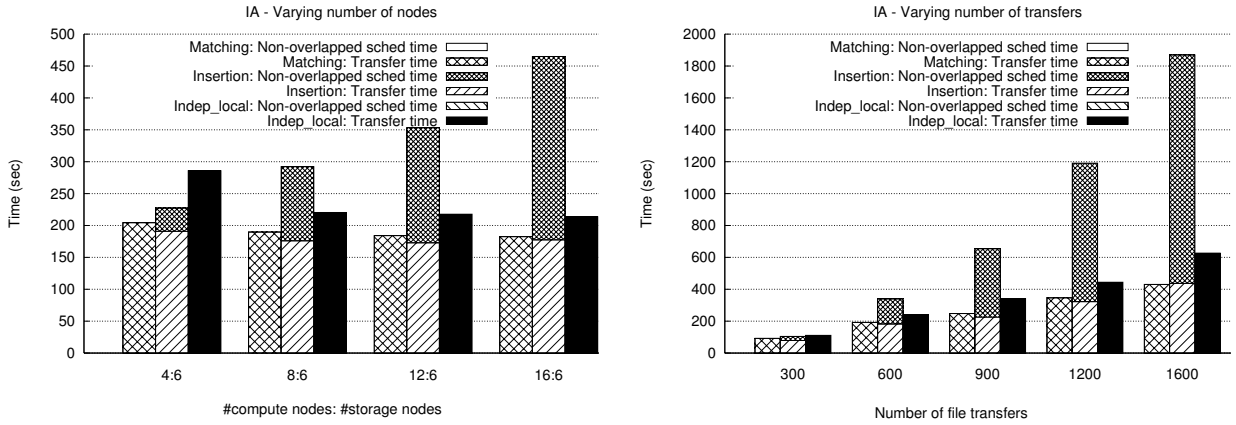

(a) (b)

Figure 6: (a) Performance of all schemes with varying network heterogeneity, (b) Performance of all schemes by employing a bipartite platform graph

access the resource) and service rate (number of work units performed per time unit.

Fig 5 shows the comparison across the scheduling schemes in terms of the overall file transfer time(secs). These experiments were conducted using 4, 8, 12 compute nodes on randomly generated file request workloads by employing Mpich via tcp(ethernet). The initial distribution of files on the nodes was also chosen as random. The input request set consisted of around 50 file transfers each involving 1GB files. The results show that the $IP$ scheme performs the best. This is because the IP formulation is able to integrate the global information of the file transfer request set and the platform topology information by incorporating it into its goal function of minimizing the overall file transfer time. It therefore, acts as a lower bound on the overall file transfer time. The matching based approach $Matching$ performs quite similar to the previously proposed insertion scheduling approach $Insertion$. This is because, the matching based approach leads to a contention minimizing schedule since the graph matching ensures that each at step, a set of non-conflicting file transfers are chosen to execute. $Indep\_local$ performs the worst as expected.

Fig 6(a) shows the performance of the algorithms in terms of overall file transfer time (secs)



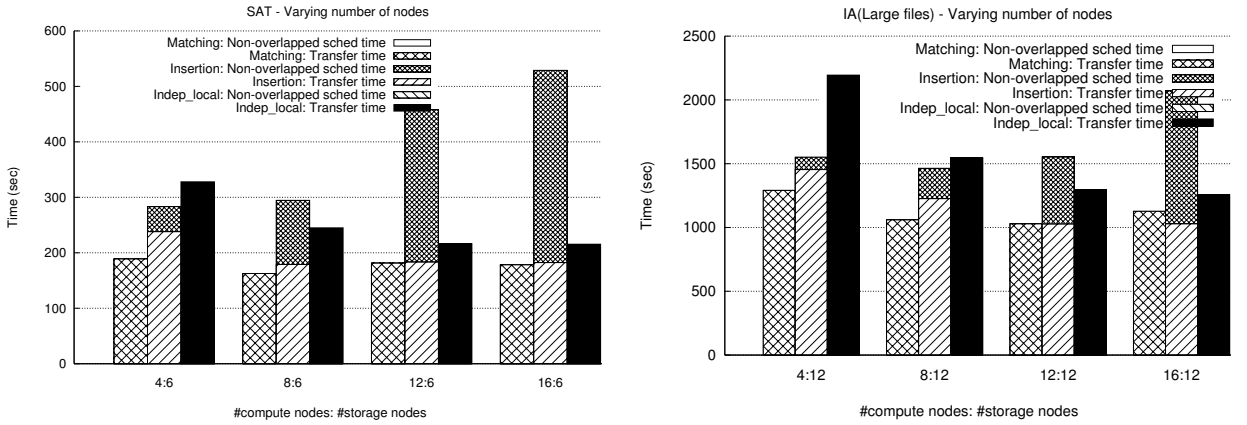

(a) (b)

Figure 7: (a) Performance of different schemes for IA workload with varying number of nodes, (b) Performance of different schemes for IA workload with varying number of file transfers

on configurations with different degrees of network heterogeneity. This experiment was conducted over 12 nodes of the cluster by employing Mpich via tcp(ethernet). The workload used for this experiment is the same one corresponding to the results shown in Fig 5. Since the workload was random, each of the 12 nodes could possibly act as sources for some files and destinations for others. $(1:1)$ corresponds to the network homogeneous case while $(1:2)$ and $(1:3)$ correspond to network heterogeneity cases. We abstracted the platform graph as a fully-connected network and emulated heterogeneity by randomly choosing half of the links to have double and triple the communication bandwidth as compared to the remaining links, respectively for the $(1:2)$ and the $(1:3)$ cases. The emulation is achieved by transferring proportionally smaller amounts of data on the faster links followed by locally padding the rest of bytes to the file. The results show that the performance gap between the $IP$ approach and the other approaches decreases with increasing levels of network heterogeneity. At low heterogeneity, $IP$ performs better because it explores a much larger search space of efficient solutions thereby achieving a better global solution. However, as the extent of heterogeneity increases, the search space of efficient solutions becomes more and more restricted to faster links and all the schemes take that into account.

Fig 6(b) shows the performance of the scheduling schemes on a bipartite topology platform graph. This experiment was conducted over 8, 12, 16 nodes of the cluster by employing the Infiniband interconnect. The bipartite topology was emulated by abstracting the topology as two distinct subsets of nodes with interconnection links only across the two sets. The workload for this experiment was a randomly chosen workload with the input request set consisting of multiple destination node mappings for each file. The size of each file in the workload was equal to 1GB. The result show expected trends except that the performance of $Indep\_Local$ is much worse than the other approaches. This is because each file needs to be sent to multiple different destinations, thereby leading to increased end-point contention due to multiple simultaneous requests for the same file.

Figure 7 shows the scalability results with varying number of compute nodes and varying number of input requests for IA. Since $IP$ takes too long to execute even for moderately-sized



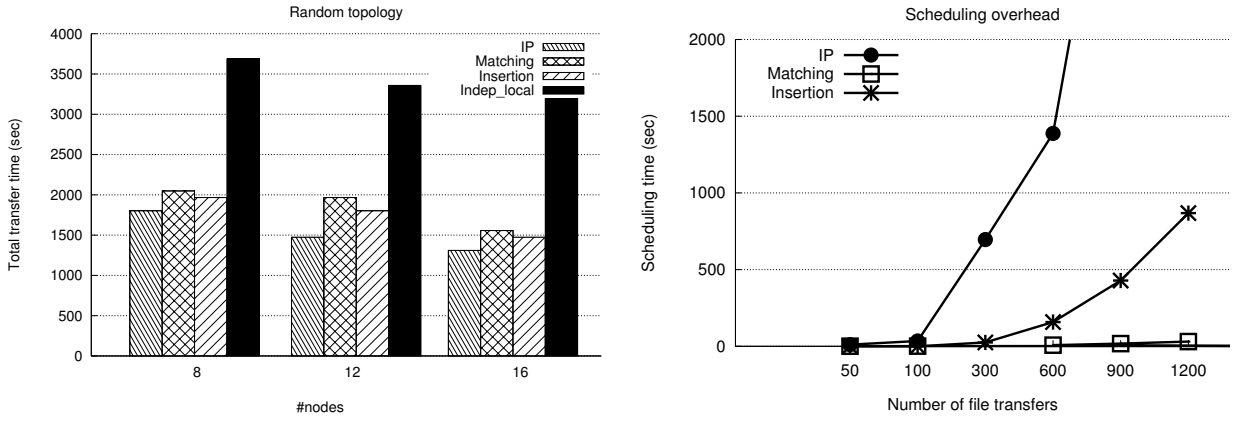

(a) (b)

Figure 8: (a) Performance of different schemes for SAT workload with varying number of nodes, (b) Performance of different schemes for IA workload (large files)with varying number of nodes

workloads, therefore in this figures, we show results only for the other three schemes. To analyze the scalability of $Matching$ with respect to the number of compute nodes, we ran experiments with an IA workload consisting of around 250 tasks over 4, 8, 12 compute nodes and 6 storage nodes. Note that the Figure 7 shows the performance in terms of two metrics, namely the total file transfer time and the non-overlapped scheduling time. The non-overlapped scheduling time is the difference between the end-to-end execution time and the total file transfer time, where end-to-end execution time is defined as the elapsed time between the instant when the scheduler accepts a batch of requests to the instant when all the requests have been finished. In other words, the non-overlapped scheduling time is the perceived scheduling overhead.

For $Insertion$, the end-to-end execution time is simply the sum of the scheduling time and the total file transfer time. This is because, for $Insertion$, the centralized scheduler generates the entire schedule once at the beginning followed by the transfer of files. However, $Matching$ is a dynamic scheduling approach wherein the scheduler generates the schedule in an iterative fashion while the file transfers are taking place. Therefore, the non-overlapped scheduling time is negligible and the end-to-end execution time closely matches the overall file transfer time. The base heuristic $Indep\_local$ also has negligible scheduling overhead. Figure 7(a) shows that $Matching$ performs significantly better than $Insertion$ in terms of the end-to-end execution time. This is because, non-overlapped scheduling time in $Matching$ is very small. In terms of the total file transfer time, the performance of $Matching$ is quite close to $Insertion$. Figure 7(b) shows the results with increasing number of requests for an IA workload. We observe that $Matching$ is able to perform much better than $Insertion$. This is because $Insertion$ has a quadratic dependence of its complexity on the number of requests as opposed to $Matching$ which has a linear dependence.

Fig 8(a) shows the performance results for a SAT workload in terms of the total file transfer time and the non-overlapped scheduling time. We observe that the $Matching$ scheme outperforms $Insertion$ by upon 20% in terms of the total file transfer time. In terms of the end-to-end execution time, $Matching$ does significantly better than $Insertion$. Fig 8(b) shows the performance results for a larger IA workload involving transfer of 1GB and 200MB files initially distributed over 12




(a) (b)

Figure 9: (a) Performance of all schemes by employing a random platform graph, (b) Scheduling overhead for all schemes

storage nodes. The number of compute nodes is varied from 4, 8, 12 to 16 nodes. The results show expected trends.

Fig 9(a) shows the performance in terms of total file transfer time of the scheduling schemes on a random topology platform graph consisting of 8, 12 and 16 nodes respectively. These were obtained by using simulations involving transfer of 1GB files over 100Mbit/sec fast ethernet. The results show that $IP$ approach performs the best as expected. The performance of the $Matching$ heuristic is able to match the performance of the Insertion scheduling approach. Fig 9(b) shows the scheduling times for various schemes. The scheduling time shown is the actual time spent in generating the schedule. Essentially, it is the sum of the overlapped and the non-overlapped scheduling times.$IP$ has a high scheduling overhead for larger configurations, due to the exponential complexity of the search. The scheduling time of $Insertion$ is higher than that of $Matching$ for larger sized workloads, as expected.

# 7 Conclusions

The paper developed two strategies for collectively scheduling a set of file transfer requests made by a batch of data-intensive tasks on heterogeneous systems - one approach formulates the problem using 0-1 Integer Programming by employing the concept of time-expanded networks and another based on using max-weighted graph matching. The performance results show that the IP formulation results in the best overall file transfer time. However, it suffers from high scheduling time. The graph matching based approach results in slightly higher file transfer completion times, but is much faster than the IP based approach. Moreover, the matching based approach is able to match the performance of a previously proposed Insertion scheduling approach and at the same time is much faster than it. Our conclusion is that the IP based approach is attractive for small workloads, while the matching based approach is preferable for large scale workloads and system configurations.

# References


[1] Ilog cplex 9.1. 2004. http://www.ilog.com/.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The globus striped gridftp framework and server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2005. IEEE Computer Society.

[3] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, page 5. IBM Press, 1998.

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

[5] 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.

[6] J. Czyzyk, M. P. Mesnier, and J. J. Moré. The neos server. *IEEE Comput. Sci. Eng.*, 5(3):68–75, 1998.

[7] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[8] 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.

[9] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Math. Program.*, 104(1):91–104, 2005.

[10] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

[11] H. N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(2):221–234, 1976.

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

[13] A. Giersch, Y. Robert, and F. Vivien. Scheduling tasks sharing files on heterogeneous master-slave platforms. In *PDP'2004, 12th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 364–371, 2004.

[14] I. Holyer. The np-completeness of edge-colouring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

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

[16] 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.

[17] G. Khanna, T. Kurc, U. Catalyurek, P. Sadayappan, and J. Saltz. A data locality aware online scheduling approach for i/o-intensive jobs with file sharing. In *Proceedings of the 12th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2006)*, 2006.

[18] G. Khanna, N. Vydyanathan, U. Catalyurek, T. Kurc, S. Krishnamoorthy, J. Saltz, and P. Sadayappan. Task scheduling and file replication for data-intensive jobs with batch-shared i/o. In *Proceedings of the The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, 2006.

[19] G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. A hypergraph partitioning based approach for scheduling of tasks with batch-shared i/o. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 792–799, Washington, DC, USA, 2005. IEEE Computer Society.

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

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

[22] 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.

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

[24] N. Vydyanathan, G. Khanna, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan. Scheduling of tasks with batch-shared I/O on heterogeneous systems. In *Heterogeneous Computing Workshop (HCW'06)*, Apr. 2006.