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

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

[1] Dept. of Computer Science and Engineering
{khannag, saday}@cse.ohio-state.edu
[2] Dept. of Biomedical Informatics
{umit,kurc}@bmi.osu.edu, Joel.Saltz@osumc.edu
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 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 insertion scheduling and has much lower scheduling overhead. We conclude that the heuristic scheme is a better fit for larger workloads and systems.

1 Introduction

Data centers consisting of collections of storage and compute clusters provide a viable environment for hosting large scientific datasets and providing analysis services. Scientific datasets are typically stored as a set of files, distributed across multiple storage nodes. Data analysis is carried out by downloading subsets of datasets from storage systems to compute systems. Analysis tasks are then executed on local data. A data center should be able to support efficient execution of batches of analysis tasks, in which a task requests a set of files and the sets of files requested by different tasks may overlap (i.e., tasks may share files). Efficient execution of such a batch of tasks involves addressing two key problems. The first problem is the mapping of tasks to compute nodes such that the volume of overall data transfer is minimized. The second one is the transfer of files from storage nodes to compute nodes. The staging of files should be carefully scheduled and executed to minimize the contention, while accounting for the topology and the heterogeneity of bandwidths in the system.

* This research was supported in part by the National Science Foundation under Grants #CCF-0342615 and #CNS-0403342.

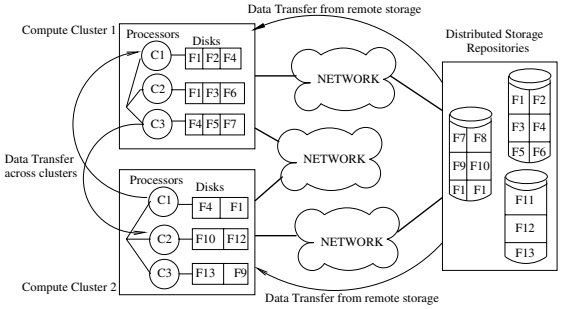


Fig. 1. Scheduling problem.

In our earlier work, we looked at the problem of scheduling and mapping a batch of data-intensive tasks [1]. This paper addresses the file transfer scheduling problem, given a mapping of tasks to nodes. In other words, it focuses on the second phase of the overall problem. We propose two approaches. The first one formulates the problem using 0-1 Integer Programming (IP) by employing the concept of time-expanded networks [2]. 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 the insertion scheduling heuristic [1]. Our results show that the IP formulation results in better schedules, but introduces high scheduling overhead. The second approach performs as well as the insertion scheduling and also takes much less time to compute a schedule thereby making it a good choice for larger workloads and systems.

2 Problem Definition and Related Work

Problem Definition: We target batches consisting of independent sequential tasks. Each task requests a subset of files from a dataset and can be executed on any of the compute nodes. The files required by a task should be staged to the node where the task is allocated. We assume a single port model wherein multiple requests to the same node are serialized. A heterogeneous multi-cluster environment consisting of compute and storage nodes is represented by graph $G = (V, E)$, referred to here as a *platform graph*. Here, V is the set of nodes and E is the set of edges. We assume that the graph G is connected. We employ a store and forward model of file transfer which implies that if a file f_ℓ needs to be transferred from a node v_i to a non-adjacent node v_j, the file is routed along one of the multiple possible paths between v_i and v_j. 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 scheduler is a set of two tuples $R = \{< f_\ell, v_i >\}$ representing that the file f_ℓ needs to be transferred to the node v_i. The initial mapping of files to nodes (storage and/or compute nodes, if the file has been replicated on a

compute node for a previous request) is represented by the set $D = \{< f_\ell, v_j >\}$, which means that the file f_ℓ is initially present on the node v_j. *Our objective is to find and efficiently execute a schedule that will minimize the total file transfer time. The schedule 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 to be transferred and the file transfer start time.* The file transfer scheduling optimization problem is $NP - complete$. Please refer to the technical report [3] for the proof. Fig. 1 shows an illustration of the problem.

Related Work: Giersch et al. [4] address scheduling of a collection of tasks sharing files onto heterogeneous clusters. Their work proposed extensions to the MinMin heuristic [5] to lower the scheduling cost. In our past work, we looked at the problem of scheduling a batch of data-intensive tasks on homogeneous clusters [1]. Our prime focus was to address the first phase of the overall problem that is to accomplish task mapping. GridFTP [6] is a protocol which enables high performance data movement by employing techniques like multiple TCP streams per transfer and striped transfers. In contrast, our work is complementary to GridFTP and can be applied in conjunction with it.

3 Scheduling Schemes

3.1 Insertion Scheduling Based Approach

Giersch et al. [4] employ an insertion scheduling scheme to schedule file transfers. In our past work [1], 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 (TCT) to transfer a file f_ℓ 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. At each step, the algorithm chooses a file, destination node pair $< f_\ell, v_k >$ and schedules the transfer of file f_ℓ to node v_k. To accomplish this, it finds the expected 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 expected transfer completion time. This process is then repeated until all the file transfers have been scheduled. For a platform graph $G = (V, E)$ and an input request set R, the complexity of insertion scheduling is $O(|R|^2 \times |V| \times (|E| + |V| log(|V|)))$ [3].

3.2 0-1 Integer Programming-based Approach

In the following discussion we use subscripts i and j for nodes, e for edges, ℓ 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. In our formulation, we make use of the concept of time-expanded networks [2]. A time-expanded network captures

the temporal aspects of network flow such that flows over time in the original network can be treated as flows in the time-expanded network. Let T^* denote the upper bound on the total completion time of all the file transfers. For each file f_ℓ to be transferred, we construct a time expanded network $G'_l = (V'_l, E'_l)$ as follows. For each node v_i in the system and each time $t = 0, ..., T^*$, we add a vertex v_{it} to the graph G'_l. For an edge $e = \{v_i, v_j\}$ connecting any two nodes v_i and v_j, $Time_{lij}$ represents the transfer time of file f_ℓ on the link $e = \{v_i, v_j\}$. We add a directed edge $(v_{it}, v_{jt'})$ to the time expanded network G'_l if $t' \leq T^*$, where $t' = t + Time_{lij}$. The objective function of the 0-1 IP scheme is to the minimize the overall file transfer time $FileTransferTime = \sum_{(\forall t)} Busy_t$ under a set of constraints. It solves for the following set of variables: 1) $Busy_t$, which is a binary variable. $Busy_t = 1$, if there is a file transfer which is finished at time t or a later point in time. 2) $X_{\ell it}$, which is a binary variable. $X_{\ell it} = 1$, if file f_ℓ is available on node v_i at time t, and 0 otherwise. 3) $Y_{\ell e}$, which is a binary variable. $Y_{\ell e} = 1$, if the edge e in the time expanded network G'_l is used to transfer the file f_ℓ, and 0 otherwise. The constraints for 0-1 IP are:

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

$$(\forall \ell)(\forall i, < f_\ell, v_i > \in D) X_{\ell i 0} = 1 \tag{1}$$

A file f_ℓ is present on a node v_i at time t either if it is already present on the node at time $t-1$ or due to the file transfer of the file f_ℓ to the node v_i from one of the nodes v_j such that the file transfer is finished at time t. $I_{\ell it}$ is the set of directed edges incident on the node v_{it} in the time-expanded network G'_l.

$$(\forall \ell)(\forall i)(\forall t) X_{\ell it} = (\sum_{(\forall e, e \in I_{\ell it})} Y_{\ell e}) + X_{\ell it-1} \tag{2}$$

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

$$(\forall \ell)(\forall i, < f_\ell, v_i > \in R) X_{\ell i T^*} = 1 \tag{3}$$

A file f_ℓ 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, at most one outgoing arc is allowed from a node v_i at time t. $O_{\ell it}$ is the set of directed edges outgoing from the node v_{it} in the time-expanded network G'_l.

$$(\forall \ell)(\forall t)(\sum_{(\forall e, e \in O_{\ell it})} Y_{\ell e}) \leq X_{\ell it} \tag{4}$$

A file f_ℓ once staged to a node v_i remains available on the node.

$$(\forall \ell)(\forall i)(\forall t) X_{\ell it} \leq X_{\ell it+1} \tag{5}$$

Each node v_i can be involved in at most 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.

Algorithm 1 Maximum Weighted Matching based Scheduling Heuristic

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

while there exists a pending request **do**
  for each pending request $< f_\ell, v_i >$ **do**
    Run the Modified Dijkstra's algorithm on Graph G for the request $< f_\ell, v_i >$. Let $Path_{\ell i}$ denote the file transfer path which yields the earliest completion time for the request.
  Create a file transfer graph $G' = (V', E')$ as follows.
  for each pending request $< f_\ell, v_i >$ **do**
    Let nodes v_{i1} and v_{i2} comprise the first hop of the file transfer path $Path_{\ell i}$.
    $V' = V' \cup \{v_{i1}, v_{i2}\}$.
    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
  Run the Max-weighted matching algorithm on the Graph G' to get a Matching
  Schedule the chosen set of edges belonging to the Matching

$$(\forall t)(\forall i)(\sum_{(\forall \ell)(\forall e, e \in C_{\ell i t})} Y_{\ell e}) \leq Busy_t \tag{6}$$

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.

$$(\forall t) Busy_t \geq Busy_{t+1} \tag{7}$$

3.3 Max-Weighted Matching Based Scheduling Scheme (MMSS)

The MMSS is an iterative algorithm and employs max-weighted matching as illustrated in Algorithm 1. For a 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. 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.

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_ℓ 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_ℓ onto the node v_i. This step requires running a variant of Dijkstra's shortest path algorithm on G to find which one of the multiple possible sources to stage the file from. 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. 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 matching on the file transfer graph to obtain a set of non-contending ready file transfers and schedules them. In this work, we modify the Dijkstra's algorithm to take into account the wait times of the source and the destination nodes as well as the link bandwidths. Once a file transfer is scheduled between a source and a destination node, the wait time on both the nodes is incremented by the expected file transfer time, which is simply the size of the file divided by the bandwidth. Therefore, the transfer completion time for an unscheduled transfer between a source-destination pair is the sum of the earliest idle time (which is simply the maximum of the wait times on the two nodes) and the expected file transfer time. This procedure works iteratively until all the file transfers have been scheduled.

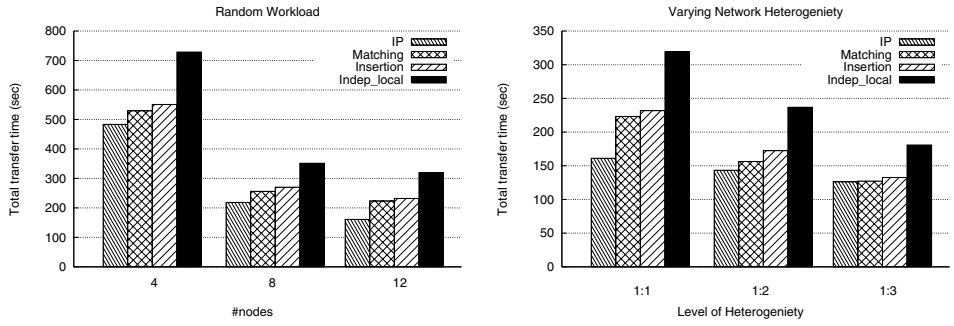
We employ Gabow's $O(|V|^3)$ implementation of the Edmond's algorithm for computing maximal matching on graphs [7]. The worst case complexity of the matching based heuristic is $O(|R| \times (|V|^4))$. For furthur details, please refer to the technical report [3]. The number of input requests $|R|$ is typically orders of magnitude higher than the number of vertices $|V|$. Therefore, in practice, the matching based heuristic is expected to perform much faster than the insertion scheduling approach presented in Section 3.1.

4 Experimental Results

For experimental evaluation, we used both randomly generated workloads as well as workloads derived from two application classes: satellite data processing (**SAT**) and biomedical image analysis (**IA**) [1, 8]. For **IA**, we implemented a program to emulate studies that involve analysis on images obtained from MRI and CT scans. A 1 Terabyte dataset was created which emulates a study involving 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 were distributed among all the storage nodes in a round robin fashion. To generate datasets for **SAT**, we employed an emulator developed in [8]. For **SAT**, the 250GB dataset was distributed across the storage nodes using a Hilbert-curve based declustering method. Each file in the dataset was 50 MB. To generate the input file request set for the two application domains, we apply our task-mapping technique [1] to map a batch of tasks onto a set of compute nodes. Since each task is associated with a set of files, the task mapping provides information about the destination nodes for each file.

In addition to the three schemes; the integer programming (IP), the graph matching based approach ($Matching$), and the insertion scheduling approach ($Insertion$), we implemented a base scheme, referred to here as $Indep\_local$.

This is a relatively simpler scheduling scheme where each destination node knows 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. In the experiments, IP uses the feaspump solver [9], available through the NEOS Optimization Server [10] to compute the schedule. The upper bound T^* defined in Section 3.2 was set to be value obtained by $Matching$. Since the feaspump solver gives feasible solutions which may not be optimal, we apply binary search in conjunction with the solver to get the optimal value of the objective function.



(a) (b)

Fig. 2. (a) Performance of all schemes for a randomly generated workload, (b) Performance of all schemes with varying network heterogeneity

Fig 2(a) compares the different schemes in terms of the overall file transfer time (in seconds). These experiments were conducted on randomly generated workloads. The initial distribution of files on the nodes was also chosen randomly. The input request set consisted of 50 file transfers each involving 1GB files. The results show that IP results in the best schedule. This is because IP is able to integrate the global information of the input request set and the platform topology into the objective function. $Matching$ performs quite similar to $Insertion$; it yields a schedule that minimizes end-point contention, since the graph matching ensures that each at step, a set of non-conflicting transfers are chosen. $Indep\_local$ performs the worst as expected.

Fig 2(b) shows the performance of the schemes when network heterogeneity is varied. This experiment was conducted using 12 nodes by employing the workload used in Figure 2(a). Since the workload was random, each of the 12 nodes could possibly act as sources for some files and destinations for others. We abstracted the platform graph (see Section 2) 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. These are denoted by $(1:2)$ and $(1:3)$ in the results; $(1:1)$ corresponds to a homogeneous network case. On the cluster machine used for the experiments, the

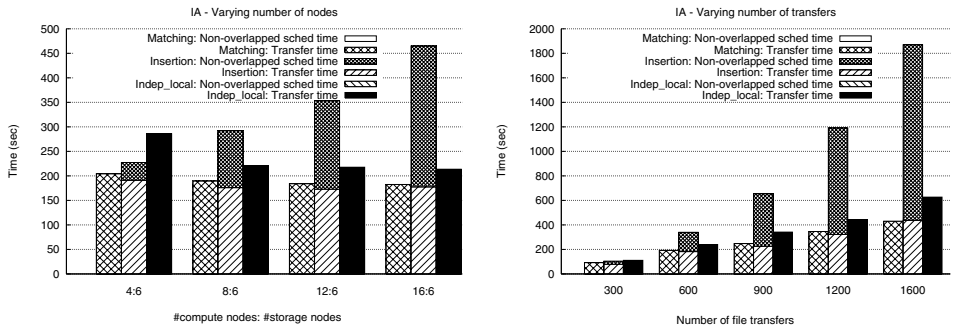


Fig. 3. Performance of different schemes for IA workload with (a) varying number of nodes, (b) varying number of file transfers

network heterogeneity 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 experimental results show that the performance gap between IP and the other approaches decreases with increasing heterogeneity. At low heterogeneity, IP performs better because it explores a much larger search space 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.

Figure 3 shows the scalability results with varying number of compute nodes and varying number of input requests. Since IP takes too long to execute even for moderately-sized workloads, we show results for the other three schemes only. 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, 16 compute nodes and 6 storage nodes. Note that the Figure 3 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. The 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 completed. 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. $Insertion$ generates the entire schedule once at the beginning followed by the transfer of files. For $Matching$, on the other hand, the schedule is generated iteratively 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. Figure 3(a) shows that $Matching$ performs significantly better than $Insertion$ in terms of the end-to-end execution time. This is because, non-overlapped schedul-

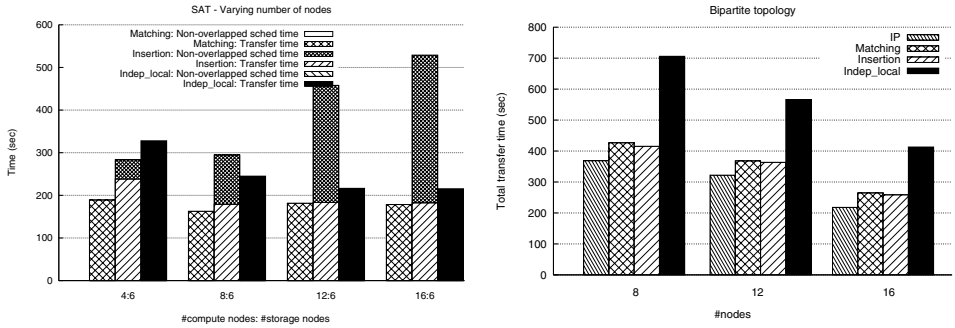


Fig. 4. (a) Performance of different schemes for SAT workload with varying number of nodes, (b) Performance of all schemes by employing a bipartite platform graph

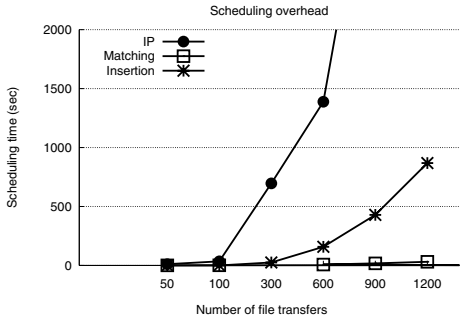


Fig. 5. Scheduling overhead for all schemes

ing time in *Matching* is very small. In terms of the total file transfer time, the performance of *Matching* is quite close to *Insertion*. Figure 3(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 4(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 *Matching* outperforms *Insertion* by upto 20% in terms of the total file transfer time. In terms of the end-to-end execution time, *Matching* does significantly better. Fig 4(b) shows the results in terms of total file transfer time on a bipartite platform graph for all the schemes. The bipartite topology was emulated by having two distinct subsets of nodes with links only across the two sets. We employed a randomly generated workload with multiple destination node mappings for each file. The results 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 contention due to multiple simultaneous requests for the same file. Fig 5 shows the scheduling times for various schemes. The scheduling time shown is the actual time spent in generating the schedule. IP has a high scheduling overhead for larger configurations, due to its exponential complexity. The scheduling time of $Insertion$ is higher than that of $Matching$, as expected.

5 Conclusions

We proposed two strategies for collectively scheduling a set of file transfer requests made by a batch of data-intensive tasks on heterogeneous systems - one approach employs 0-1 Integer Programming and the other employs max-weighted matching. The results show that the IP formulation results in the best overall file transfer time. However, it suffers from high scheduling time. The matching based approach results in slightly higher file transfer times, but is much faster than the IP based approach. Moreover, the matching based approach is able to match the performance of Insertion scheduling with a much lower scheduling overhead. Our conclusion is that the IP based approach is attractive for small workloads, while the matching based approach is preferable for large scale workloads.

References

1. Khanna, G., Vydyanathan, N., Kurc, T., Catalyurek, U., Wyckoff, P., Saltz, J., Sadayappan, P.: A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O. In: Proc. of CCGrid'05 - Volume 2. (2005) 792–799
2. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research **6** (1958) 419–433
3. Khanna, G., Catalyurek, U., Kurc, T., Sadayappan, P., Saltz, J.: Scheduling file transfers for data-intensive jobs on heterogeneous clusters. Technical Report OSU-CISRC-1/07-TR05, CSE Dept., The Ohio State University (2007)
4. Giersch, A., Robert, Y., Vivien, F.: Scheduling tasks sharing files from distributed repositories. In: Proc. of Europar'04 - volume 3149 of LNCS. (2004) 246–253
5. Ibarra, O.H., Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. ACM **24** (1977) 280–289
6. Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M.: The globus striped gridftp framework and server. In: Proc. of SuperComputing'05. (2005)
7. Gabow, H.N.: An efficient implementation of edmonds' algorithm for maximum matching on graphs. J. ACM **23** (1976) 221–234
8. Uysal, M., Kurc, T.M., Sussman, A., Saltz, J.: 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, LNCS, Vol. 1511, Springer-Verlag (1998) 243–258
9. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. **104** (2005) 91–104
10. Czyzyk, J., Mesnier, M.P., Moré, J.J.: The neos server. IEEE Comput. Sci. Eng. **5** (1998) 68–75