An Online Scheduling Algorithm with Advance Reservation for Large-Scale Data Transfers *

Mehmet Balman[1] and Tevfik Kosar[2]

[1] *Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720*

[2] *Department of Computer Science & Engineering, University at Buffalo (SUNY), Buffalo, NY 14260*

Email: mbalman@lbl.gov, tkosar@buffalo.edu

2010

Abstract

Scientific applications and experimental facilities generate massive data sets that need to be transferred to remote collaborating sites for sharing, processing, and long term storage. In order to support increasingly data-intensive science, next generation research networks have been deployed to provide high-speed on-demand data access between collaborating institutions. In this paper, we present a practical model for online data scheduling in which data movement operations are scheduled in advance for end-to-end high performance transfers. In our model, data scheduler interacts with reservation managers and data transfer nodes in order to reserve available bandwidth to guarantee completion of jobs that are accepted and confirmed to satisfy preferred time constraint given by the user. Our methodology improves current systems by allowing researchers and higher level meta-schedulers to use data placement as a service where they can plan ahead and reserve the scheduler time in advance for their data movement operations. We have implemented our algorithm and examined possible techniques for incorporation into current reservation frameworks. Performance measurements confirm that the proposed algorithm is efficient and scalable.

Keywords: Advance Reservation, Resource Allocation, Scheduling, Data Management, Distributed Computing.

1 Introduction

Recent progress in high performance computing and distributed systems have provided collaborative science and the necessary resources for emerging computationally- and data-intensive applications. Scientific applications have especially become increasingly data intensive [29, 41] in the recent years. The SuperNova project in astronomy is producing terabytes of data per day, and a tremendous increase is expected in the volume of data in the next few years [7]. The LSST (Large Synoptic Survey Telescope) is scanning the sky for transient objects and producing more than ten terabytes of data per simulation [54]. Similarly, simulations in bimolecular engineering generate huge data-sets to be shared between geographically distributed sites. In climate research, data from every measurement and simulation is more than one terabyte. In high-energy physics [18], processing of petabytes of experimental data remains the main problem affecting quality of the real-time decision making. Often, such applications require geographically distributed resources to satisfy their immense computational requirements. The dynamic nature of interconnects between collaborating sites, heterogeneity of resources, and client/server side capacity bottlenecks (such as memory, CPU, storage capacity) necessitate provisioning (preparing network, server and client before initiating a data transfer operation for desired transfer throughput) for efficient resource utilization and performance.

A very simple use case can be explained as follows. Consider a scientific application which generates immense amount of simulation data using supercomputing resources. The generated data is stored in a temporary space and need to be moved to a data repository for further processing or archiving. Often, the data repository is located in a remote site where the generated data will be analyzed/visualized by collaborating researchers. In the remote site, another temporary space with limited lifetime may be allocated to store the data. Another application may be waiting this generated data as its input to start execution. We can allocate compute resources in advance, and we even can predict the completion time of a compute job submitted in a supercomputer queue. Therefore, users have the opportunity to have an estimation about the time their computation and analysis will finish, and the generated data will be available to be moved to the remote repository.

In current systems [36, 13], a data transfer request is managed by the scheduler without any constraints. The data transfer request is put in a queue to be scheduled after completing currently running operations. This request may be delayed because of prior long-running jobs, or it can be postponed by the scheduler to operate other short jobs. Depending on the scheduler's policy, the scheduler can initiate other jobs using some of resources shared by this job. In such a case, the number of jobs completed will increase, but the total completion time of our data transfer job will also increase. Delaying the data transfer operation, completing the transfer far after than the expected finish time, may create several problems. One common case is that other resources are allocated for further processing but they are waiting idle for the transfer operation to complete.

Delivering data placement (moving data between collaborating parties) as-a-service where users can schedule their request in advance is highly desirable. In a data placement request, users can provide a simple time constraint in which they state the earliest start time and latest completion time. Earliest start time specifies when the source data set will be ready to schedule the given task. Latest completion time specifies a desired deadline to complete the transfer operation. The scheduler

confirms the request after checking availability of resources and other tasks in the given time frame. If the request cannot be confirmed in the given time frame, the scheduler might suggest a longer time period such that latest completion time extended to satisfy the request. It is the scheduler's responsibility to satisfy given requests with the given time constraints. Future time windows are considered while accepting a request and initial decision are made in advance. On the other hand, the scheduler can accept a request that needs to be initiated instantly if there is capacity available and none of the reserved operations will be delayed.

A major challenge is to predict the completion time and also estimate the capacity of a resource involved in data transfer operations. We assume that data scheduler gathers information about the server and network capacities to control load on data servers in source and destination sites. On the other hand, predicting performance and completion time over a dynamic and shared network is quite difficult [12, 56]. Next generation research networks such as Internet2 [5] and ESnet (Energy Sciences Network) [3] provide bandwidth guaranteed on-demand data access between collaboration institutions. Advance network reservation systems such as ESnet's OSCARS enable infrastructure for data schedulers to retrieve possible future reservations [11] and allocate bandwidth between two sites for a given duration with predictable throughput [6]. Using a network interconnect in which we can reserve and guarantee bandwidth enables data scheduler to make accurate decisions and satisfy user requirements with given time constraints.

Data scheduler checks the availability of resources, and the server and the network capacity are allocated for the future time period in advance. We consider other requests reserved for future time windows and examine available capacity both in network and server resources to make a new reservation satisfying the requirements of a job submitted. If there is no available slot to execute the transfer operation with the given user constraints, the job submitted is rejected, or the latest completion time is extended and the user is notified about the possible finish time.

While scheduling a new job, we may also need to change a reservation that belongs to an already accepted job which has not started yet. In that case, we release previously allocated resources to make new reservations if possible, if there is available slot to move the job start time backwards. Conversely, data transfer jobs can be moved forward if there is enough time before its deadline. Data scheduler should operate in an opportunistic manner to maximize resource utilization and the number of jobs accepted. On the other hand, it should first take into consideration of the jobs for which it has already confirmed to satisfy their deadlines. Therefore, we make changes to the reservations of previously accepted jobs only if we guarantee completion within the given time constraints.

In this paper, we present a new data scheduling model with advance reservation and provisioning. We intend to eliminate possible long delays in completion of a transfer operation by taking advantage of bandwidth guaranteed paths and user defined time constraints, and increase utilization both in client and server sites by giving an opportunity to provision resources in advance. Users submit their jobs by stating the total volume of data needed to be transferred between source and destination nodes. In addition to resource constraints, each job is also bounded by time constraints, the earliest start time and desired latest completion time. In order to take advantage of the available network bandwidth, we should also provision other resources for storage and server capacity between collaborating parties.

Data scheduler interacts with reservation managers and data transfer nodes in order to reserve from the available capacity, to guarantee completion of jobs that are accepted, and confirmed to satisfy preferred time constraint given by the user.

In general, number of reservation options is exponential, and the scheduling problem is NP-hard. We analyze time-dependent resource assignment problem with bottleneck constraints, and present a detailed study of data transfer scheduling with resource and time conflicts. We propose an efficient heuristic for scheduling data placement operations with advance reservation. We have implemented our algorithm and examined possible techniques for incorporation into current reservation frameworks. Performance measurements confirm that the proposed algorithm is efficient and scalable.

The organization of this paper is as follows. In Section 2, we highlight some of the relevant studies in the literature. In Section 3, we analyze scheduling of data transfer operations with time and resource constraints., and we explain several common approaches used to evaluate file scheduling and resource assignment problems. In Section 4, we propose an online scheduling heuristic such that the scheduler makes decision whenever a new data transfer job is submitted, and we analyze data transfer scheduling between distributed resources with given time and resource constraints. We conclude the paper in Section 5.

2 Related Work

There are several studies concentrating on data management in scientific applications [8, 9, 55]; however, resource allocation and job scheduling considering the data requirements still remains as an open problem. There is a few work towards coordinating resource allocation and advance reservation together for data movements [48]. Existing systems fail to address issues such as scheduling according to given user requirements and priorities, taking advantage of advance resource reservation, and adapting to dynamic environment in distributed systems. Current data schedulers manage data transfer jobs by trying to optimize for performance and resource utilization [36, 13], but they do not provide advance resource reservation and coordination where users can plan ahead and allocate/reserve the data placement service for a future time.

In [26], a reservation and allocation architecture, GARA, is defined to address several problems in providing end-to-end quality of service for next generation research networks. Heterogeneity of resources requires independent control and local administration policies of individual resources. Computational elements also affect end-to-end performance and they should be managed and monitored separately while dealing with reservation elements. The GARA project aims to provide application level co-allocation by providing a reservation API in order to coordinate resources, and to allocate them in advance.

There are also several relevant studies in the literature using reinforcement learning for resource management and planning [27], and user constraints for file transfer scheduling [17]. Priority-based scheduling has been studied for real-time system [45], especially for databases to satisfy time constraints with transactions [22]. Deadline scheduling algorithms consider the time constraint of every request to ensure the deadline (completion time). We can classify real-time requests into two

categories; hard and soft transactions. In a hard real-time transaction system, the scheduler needs to guarantee the exact completion time. There is no benefit to finish the request after the deadline. In a soft real-time transaction system, the scheduler considers the time constraint and prioritizes the requests with earliest deadline in the scheduling queue [45]. In our data scheduling paradigm, we consider soft-deadline scheduling for data transfer requests. We can allocate the server and the network capacity (bandwidth); however, it is difficult to guarantee the exact completion time due to possible failures, system problems, and unpredicted instant performance degradations. The scheduler takes into account time constraint and tries to schedules the job before its deadline while making the decision to maximize the number of job accepted in the system.

There is an increase in developing projects for research networks to provide dedicated bandwidth channels. The dedicated bandwidth networks brings the ability to provision the communication channels when the data, especially large-scale massive data, is ready to be transferred [37, 43]. In order to provide high-speed on-demand data access between collaborating institutions, research institutions established production level network supporting on-demand bandwidth reservation in which bandwidth is reserved for a specific time period [6, 5, 4]. On-demand bandwidth reservation is usually supported by Multiple Protocol Label Switching (MPLS) in layer 3 [15, 23]. In layer 2, a virtual secure circuit is setup between source and destination with a specific bandwidth over the connection. Internet 2 [5] and ESnet [3, 6] provide dynamic circuit infrastructure to establish on-demand guaranteed bandwidth point-to-point connections. TeraPaths [53] controls end-sites and allows creation of secure circuits within the site to support guaranteed bandwidth service.

There are few studies in on-demand bandwidth allocation [1, 6, 2] and advance bandwidth reservation [28, 44, 16, 21, 32]. A very typical case is to represent the network topology as a graph. In addition to that, we need a proper representation for time in advance reservation. There are two common approaches; slotted time model and continuous time model. In slotted time model, time is divided into equal slots and each link keeps information about the available bandwidth in each slot. Several studies addressing slotted time model are [28, 16, 51, 46, 26]. In continuous time model, the link capacity is modeled as a time-bandwidth function. This provides better granularity and finer control in scheduling with a cost of increased complexity in implementation. We extend the contiguous time model and define two new concepts (time steps and time windows) to analyze time-dependent topologies for scheduling and resource assignment.

In [38], the proposed scheduling algorithms are classified as periodic scheduling and differentiated from instant scheduling. In instant scheduling, the scheduler makes a decision for every incoming request. In periodic scheduling, the scheduler makes decision in certain intervals where several requests in that period are considered. In [21], the scheduling algorithm considers flexibility in bandwidth and the time period in order to increase utilization. Many of the given algorithms have high computational complexity and large space requirements. We consider instant scheduling and use a practical approach that is suitable in real-life applications. Once a job has been accepted, it is always preferred over other jobs that are submitted later. Simply, we never dismiss and reject a committed job to accept other recent jobs in order to favor the optimization criteria. We have recently reported a flexible network reservation algorithm that provides alternate allocation possibilities for a single job, including earliest time for completion, or shortest transfer duration - leaving the choice to the user [11]. We have

implemented our algorithm as a new service extending the current underlying mechanisms of ESnet's OSCARS [6]. We try to come up with a near optimum allocation pattern for multiple data transfer requests with advance reservation. We necessitate a new methodology that is easily applicable to provide a solution for incorporation with other resource managers and reservation systems

3 Time and Resource Conflicts

We concentrate on scheduling data transfer operations in a time-dependent topology. The network bandwidth assignment, which has been described in our previous work [11], plays an important role designing a solution. In our scheduling model given in this study, the transfer rate is fixed and does not vary over time. This is one of the crucial features that affect the methodologies used to approach the problem. In order to elucidate the problem domain and introduce the concepts, we present the crucial decision points in the process of designing a scheduling algorithm with resource and time constraints. We give a simple example at the end of this section to make readers more familiar with the theory behind scheduling with time and resource constraints with fixed bandwidth assignment.

There are several studies in the literature [20] categorizing several research problems in data transfer scheduling [24, 50], and summarizing theoretical complexity and difficulty of those problems in several domains. In [31], authors analyze some common cases and show that there are polynomial time solutions for some very special types of the problem, though for the rest of the cases the solutions are exponential. Other than that, the general problem is proven to be NP-hard. The study given in [31] examines several network structures such as trees, bipartite graphs, networks with odd and even cycles, and provides a detailed complexity analyze through relaxing the problem by eliminating parameters such as file size and concurrency.

Data transfer scheduling with a specific start time and a particular deadline has been studied in [39, 47, 42]. The scheduling problem has been formulated as a multi-commodity flow problem, and uniform time slices have been used to model the time dependency in [42]. The objective is to maximize the total transfer throughput and data transfers can use varying bandwidth in every time slice. This problem can be generalized as a concurrent file transfer problem [47]; such that, we share the bandwidth between multiple jobs and try to utilize the network as much as possible. Using network flows to model and place a solution space to combinatorial optimization problems is a common practice [39]. On the other hand, sharing bandwidth between concurrent transfers can improve the total throughput but does not help satisfying completion time of each job. Our objective is to provide allocation of scheduling time satisfying given user constraints, not to improve only the system utilization. We would like to emphasize that multi-commodity flow does not apply to our case. We are dealing with network topologies with bottleneck constraints [11].

We can use unsplittable flow problem to model and clarify our problem domain. The unsplittable flow problem [35] is an interesting dilemma in algorithm research. We can simply describe it by t tasks with start and end time and a particular demand $d > 0$ and a profit p. If we assign a task, it requires b_i amount of bandwidth. We are given a network with available bandwidth b_t for every time t. The purpose is to find a subset of tasks to maximize the profit. Similarly if every task acquires

a cost value, objective is to minimize the cost. The unsplittable flow problem is NP-hard, and only polynomial approximation algorithms are given as a solution [10, 14, 34, 33, 19]. Interestingly, even for very special cases (i.e. planar graphs) the problem is still NP-hard.

In order to clarify the concept behind fixed bandwidth time dependent scheduling in a distributed network, consider a single network with a single line. We let only one edge connecting two nodes. In this special case of the general problem where network is a line, the unsplittable flow problem converts to a very well known optimization problem, Knapsack problem [40]. In Knapsack problem, we have a set of items each with a weight and a cost value, and we select a collection of items to maximize the total profit considering that we have a limit in total capacity. Similarly, we have start and end times for each task, and we have the bandwidth limit over the link. Beyond that, even if we have unique profit $p = 1$, and unique demand $d = 1$ for all tasks, and we set the edge capacities to a unique value, we still end up with a NP-hard problem. This special case can be generalized to maximum edge-disjoint paths problem in graph theory [14]. The problem we attempt to solve is quite hard. To the best of our knowledge, only polynomial approximation algorithms have been proposed in literature as discussed above, and there is no constant factor approximation algorithm known to solve the unsplittable flow problem.

A common approach is to design approximation algorithms in which we set priorities and rate each selection to reduce the search space. The number of possible options to examine in order to make the best selection exponentially increases in worst case. Instead of that, we rate each selection and displacement based on the priority or the cost/desire we assign to each task. Thus, we can design polynomial greedy heuristics which can solve the problem with a near optimum scheduling choice. Note that very simple but effective greedy approaches like best-fit, first-come, and earliest-deadline, use some preference/criteria to make a choice among multiple options. The design of the algorithm and deciding on a good selection criteria play important role in terms of the quality of the resulting scheduling approximation. There are many studies in the literature investigation approximation algorithms for scheduling; [30] and [25] are one of them which show benefits in designing greedy algorithms with priorities.

3.1 Analyzing the Assignment Problem

We define a sample network with three data transfer nodes connected to each other over a network, given in Figure 1. Each node has a particular capacity that it can provide maximum upload and download transfer rate. It defines the limit in server site such that total throughput is also constrained by the capacity of data transfer nodes. Each job has a volume of data need to be transferred, and a specific period of time this job need to be completed - earliest start time t^E, and latest completion time t^L.

We are bounded by edge capacity as well as node capacities. Figure 2 shows the resource conflicts in this simple example. If we have a transfer request from node n_1 to n_2 running at the same with another request from node n_3 to n_2, the total bandwidth allocation given to both should not exceed the capacity of the shared node n_2.

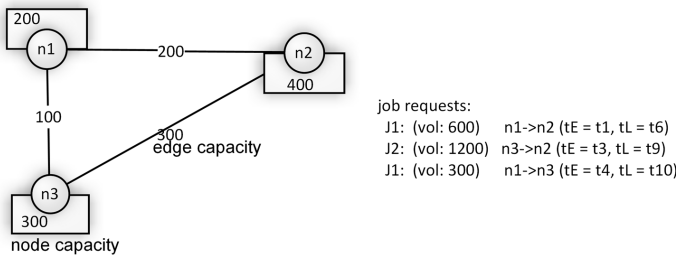

Figure 1: Sample Problem Definition

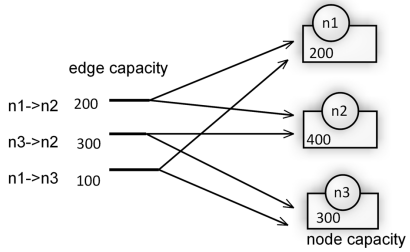

Figure 2: Node and Link - Resource Constraints

Earliest start time and latest completion time of each job defines its search interval. We focus on the search interval to find a proper allocation for the given request. Figure 4 shows time steps which are calculated according to constraints of each job. A **time step** shows the longest duration of time in which we have a stable network structure in terms of available capacity ready for reservation [11]. Figure 3 shows time windows. A **time window** is a sequence of time steps. We traverse time windows in a specific order, as shown in Figure 3. First we try to find an allocation which has shortest duration of time; or simply say which includes less time steps. Besides, we want to find an allocation with earliest completion; so, we traverse first time windows which end earlier. We further demonstrate mapping time windows to search intervals for each job in Figure 3 and 5.

The total amount of data for each job that need to be transferred characterizes the duration of the time period needed. Figure 5 shows how several time windows are eliminated in the search interval (also see Figure 5 for comparison). Further, it also illustrates the resource constraints specific to each time windows. For example, if we want to assign job J_1 into time window tw_6, we need at least a capacity of 120 allocated over the link from n_1 to n_2 which provides maximum of 200 capacity. However, we would first consider tw_3 and tw_5 if there is more capacity available since those time windows consist of less time steps (shorter duration). Figure 6 provides a more detailed view of capacities for each job to time window assignment.

We have analyzed the unsplittable flow problem. If we could solve that in a polynomial time, we would also solve this problem. Figure 7 represents the sample problem using network flows. The crucial point is that each time window may affect more than a single time step. And, those time steps need to have the same capacity allocation during the entire period of time. As an example, tw_9 and tw_5 both include ts_2. Any flow passing over these two should also consume capacity in ts_2. Even though we could represent the network structure with discrete graphs in each time step, we still need

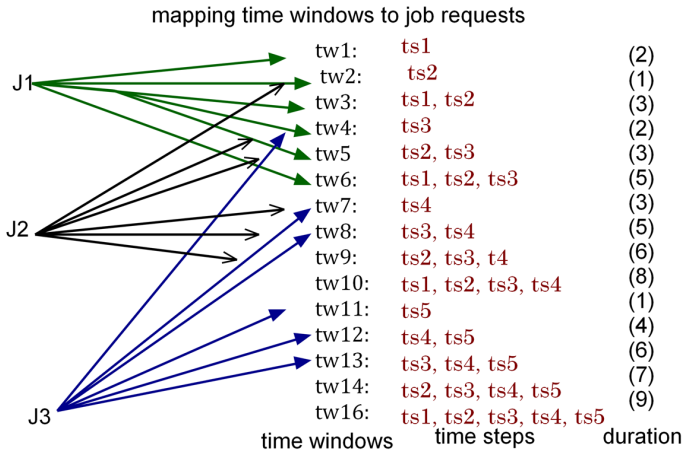

Figure 3: Time Windows

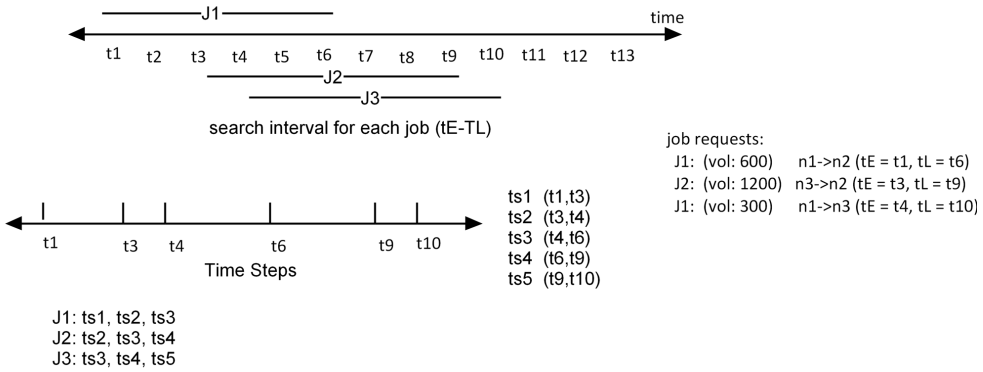

Figure 4: Time Steps and Search Interval

to consider time constraints. In other words, our problem complexity increases exponentially when we have time constraints and resource constraints together. In this case, we have resource conflicts in each time step, see Figure 2, and time conflicts for time windows as shown in Figure 7.

Figure 8 provides a table of possible assignment options that need to be considered with resource constraints given in Figure 9. In Figure 10, we present how solution space is analyzed in this sample problem. We show sample conflicts, and explain that search space is exponential. We have three possible assignment option for J_1, two for J_2, and four for J_3. Overall, we may need to consider $3 \times 2 \times 4$ choices in order to make a selection. Each assignment might affect other options, but there is no direct correlation between them.

For example, if we select tw_8 for J_2, we could assign J_3 into time window tw_{13}. tw_8 includes ts_3 and ts_4, a period of time between t_4 and t_9. The minimum capacity we can use in this time window for job J_2 is 240, but we can finish by t_8 if use 300. In such a case, we would not be able to assign J_3 into tw_{13} since there will be no capacity left at node n_3. However, if total volume of job J_3 was 200 instead of 300, we could assign it between t_8 and t_{10}.

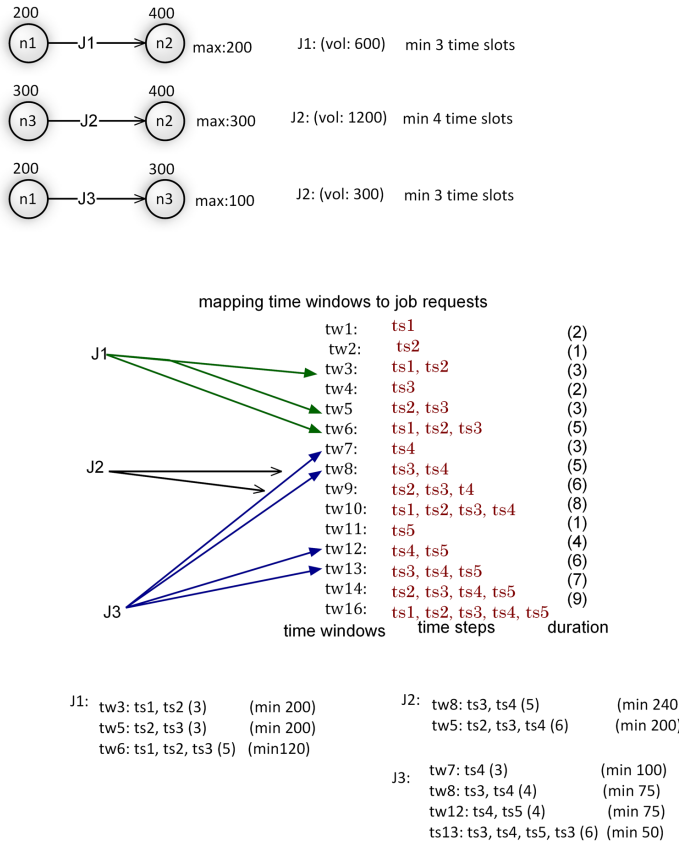

Figure 5: Mapping Jobs to Time Windows

Assume that total volume of data for job J_3 is 200. The flow from J_3 to tw_8 and tw_{12} would be 50 instead of 75. We would be able to select tw_8 for J_2, and tw_{12} for J_2 (total makes $240+50 = 290 < 300$). We could use 240 amount of bandwidth for J_3, and 50 amount of bandwidth for J_3, between t_4 and t_9. Alternatively, we could use 300 amount of bandwidth for J_3 between t_4 and t_8, and 100 amount of bandwidth for J_3 between t_8 and t_{10}. In other words, we would introduce a new point at t_8 and divide the time step ts_4 into two.

4 Scheduling with Advance Reservation

The data transfer scheduler checks other jobs in the system and considers both time and resource conflicts. Each job contains information about the total volume of data need to be transferred, source and destination end-points, and also the time period in which this data transfer operation need to be completed. Users submit data transfer jobs with a simple time constraint; an earliest time when this data will be ready to initiate the transfer, and a deadline when the user wants data transfer to be completed. In order to admit a submitted job, it has to confirm the availability of resources to complete the transfer of the data before the given deadline. If a job has been admitted, a period of time is reserved in advance with required capacity in resources along the route between these two

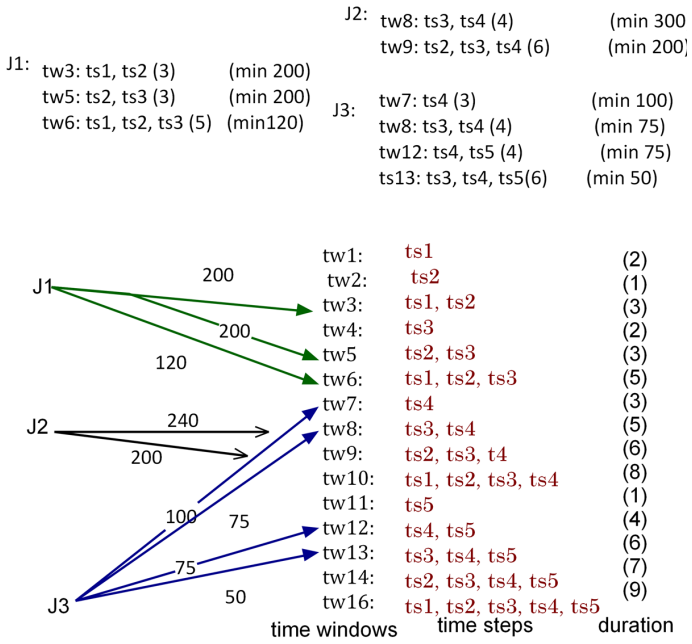

Figure 6: Minimum Capacity required for each Time Window

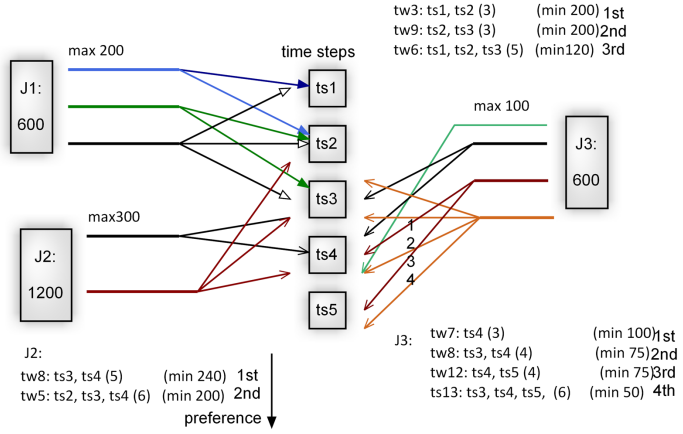

Figure 7: Time Conflicts

end-points. We consider that the scheduler has knowledge about the current and future capacity of resources that affect the end-to-end transfer performance. Users have the opportunity to give a desired period of time in which they want the transfer to be accomplished. If the scheduler cannot find a suitable time slot, it can also shift other jobs that had already have a reservation in the given time period without breaking any deadline requirements of previously admitted jobs.

An important constraint is to reserve contiguous time slots for each job such that a data transfer operation starts with fixed transfer rate and maintains this until the data volume has completely transferred. The scheduler has two main objectives. First, it ensures that no other admitted job will

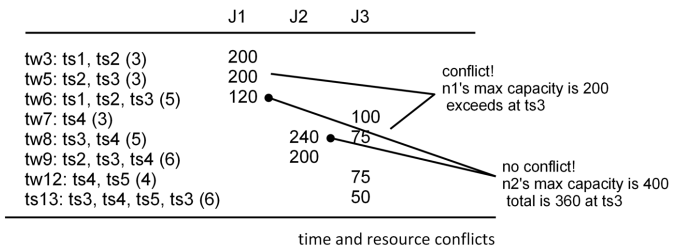

Figure 8: Time and Resource Conflicts

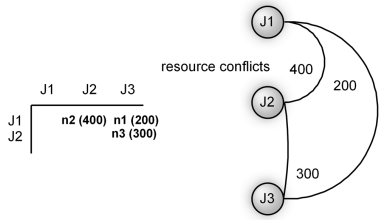

Figure 9: Resource Conflicts

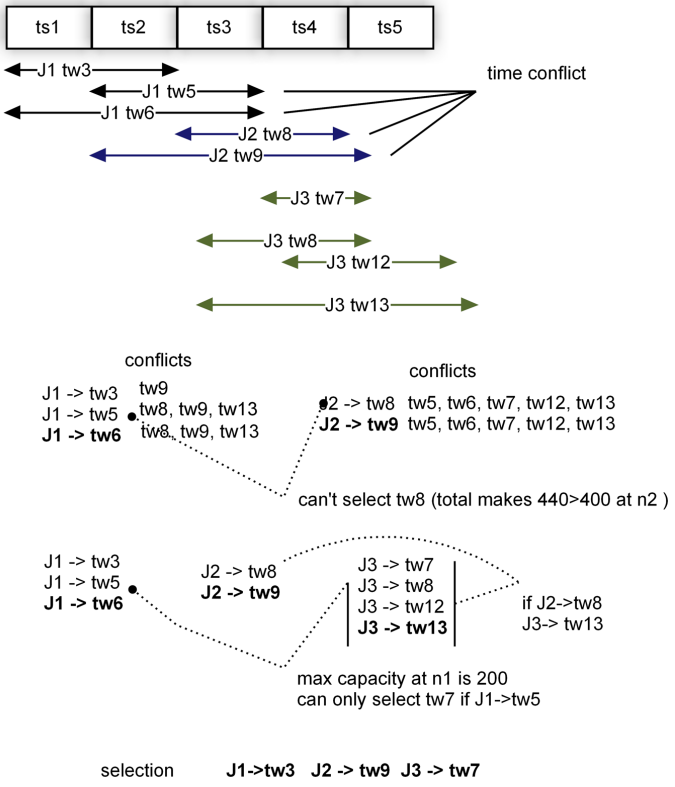

Figure 10: Assigning Jobs to Time Windows

be postponed due to making a new reservation. In addition to this fairness objective, it also tries to maximize the number of admitted jobs by moving reserved slots. The scheduler tries to be open

a suitable period of time to admit a recently submitted job by resolving time and resource conflicts. On the other hand, it also selects time slots which gives earliest completion time and with minimum interference with other admitted jobs in the system.

The connection between two end-point may span over multiple routers. As can be seen in Figure 11, data nodes are connected to the edge-routers in a network. Searching bandwidth allocation between two edge-routers and finding possible network reservation options are studied in [11].

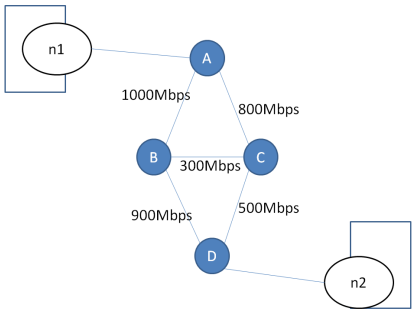

Figure 11: Connection between two data nodes

Problem Definition: We define the topology as a time-dependent directed graph $G(V, E, T)$, with a node set V of n data transfer nodes, and an edge set $E \subseteq V \times V$ with m edges, where $e_k : (v_i, v_j)$ represents a connection from v_i to v_j. For every connection between two nodes, there is a function of link capacity $u^{e_{ij}}(t)$ and $u^{e_{ji}}(t)$ where t is a variable in time domain T. In addition to that, every node has separate upload and download capacities, $u_{out}^{v_i}$ and $u_{in}^{v_i}$ respectively.

We have a dynamic network environment in which edge capacities may vary over time. On the other hand, we know the maximum upload and download capacities in each data transfer node. We consider data transfer nodes (DTNs) as specialized machine(s), with back-end storage servers and data transfer protocols, connected to the outside network with high-speed high-bandwidth interconnects.

A data transfer job is defined as $J_i = (v_i^s, v_i^d, M_i, t_i^E, t_i^L)$, where M_i is the amount of data to be transferred from source v_i to destination node v_j within the time period of (t_i^E, t_i^L). t_i^E represents the earliest possible time when this data will be ready to start the transfer operation. t_i^L represents the latest completion time the transfer operation needs to be finished. t_i^L defines a soft-deadline for the transfer operation such that the transfer operation is not interrupted if it cannot finish within the given deadline. However, the scheduler makes decisions according to the time constraints. It does not admit a job if it foresees that it is not possible to finish the transfer of the requested data before the given deadline.

If a submitted job is admitted, we set up a reservation for this job and allocate resources for a specific time period. A reservation is defined as $R_i = (v_i^s, v_i^d, \mu_i, t_i^s, t_i^e)$, where μ_i amount of bandwidth is reserved from source v_i^s to v_i^d between start time t_i^s and end time t_i^e. A reservation request is only confirmed if there exits enough capacity satisfying the allocation of μ_i bandwidth in the given time period between t_i^s and t_i^e. The total allocated bandwidth over link e_{ij} should be less than the capacity $u^{e_{ij}}(t)$ of the link for every instance of t in $[t_i^s, t_i^e]$. Similarly, the total in-coming bandwidth allocation should be less than $u_{in}^{v_i}$, and total out-going bandwidth allocation should be less than $u_{out}^{v_i}$, in the

time period of (t_i^s, t_i^e). We consider non-preemptive operations where data transfer start at t_i^s and continues till t_i^e using μ_i bandwidth from resources along the end-to-end route; consuming upload capacity of data transfer node v_i, link e_{ij}, and download capacity of v_j. The duration of the time period, and the reserved bandwidth should be enough to satisfy transferring the requested amount of data. We simply say $M_i = \mu_i \times d_i$, where d_i is the total time between t_i^s and t_i^e. Therefore, it should satisfy the requested bandwidth during the entire period of time from start to end of this transfer. The problem is to find a contiguous set of time slots such that a fixed amount of bandwidth can be allocated to satisfy the data transfer request.

4.1 Online Scheduling

We propose an online scheduler that can come up with a decision easily when a new job is submitted, so it can instruct the underlying reservation managers quickly. The main objective of the scheduler is to maximize the number of admitted jobs. Besides, it also increases the utilization by maximizing the number of jobs that use the system. With such an objective in hand, one would expect the scheduler to accept jobs with small data volume and reject or delay jobs which have large data volume. Moreover, a job interfering with many other jobs and creating time conflicts will not be preferred. Those criteria will help maximize number of admitted jobs but will result in unfairness in practice. Therefore, a crucial condition in our approach is to ensure that no other committed job will be postponed due to admitting a new reservation request.

Input: A set of admitted jobs (already in the system) and their active advance reservation
Input: A new job request J_i with earliest start t^E, latest completion t^L, volume M, source v^s and destination v^d
Get a set of time steps in the search interval $[t^E - t^L] : \{ts_1, ts_2, \ldots, ts_n\}$;
for $i = 1$ *to* n **do**
  for $j = i$ *to* 1 **do**
    Get time window $tw = tw_{j-i}$ which contains all time steps between ts_j and ts_i;
    Check available capacity for v^s, v^d and link $v^s \rightarrow$ v^d in time window tw ;
    if *the given criteria can fit into the time window* $tw = ts_j \ldots ts_i$ **then**
      Make allocation and admit the job;
Return: No reservation found;

Algorithm 1: Scheduling Algorithm: select the time window that gives earliest completion time with shortest duration.

When a new job request arrives, the scheduler checks available time slots and considers resource constraints to find a proper allocation for the new request. One major objective is to complete the given request as early as possible. It selects time slots which gives earliest completion time and with minimum interference with other admitted jobs in the system. Even though there is available resource capacity both in nodes and the link, it is always beneficial not to have many concurrent transfers running at the same time. Furthermore, we would prefer to complete a job as soon as possible. We prefer allocating higher bandwidth for a shorter duration instead of allocating lower bandwidth for a longer duration. Data transfers which takes longer and which run on resources shared concurrently with other jobs, would have higher failure probability. Algorithm 1 gives a greedy heuristic in which

data transfer operations are scheduling in submission order.

We enhance the scheduling approach given in Algorithm 1. If there is no availability, we try to open a suitable period of time to admit a recently submitted job by moving previously made allocations to resolve time and resource conflict Our approach, inspired from Gale-Shapley [52] and N-queen [49] algorithms, is to design an effective methodology which can easily be implemented and deployed in practice. When a new job request cannot find a suitable time slot to make a reservation, it competes with previously admitted jobs to move their reservations and open a proper reservation time for itself.

The outline of our scheduling methodology is as follows. When a new request arrives, we first evaluate its time and resource constraints, and we try to find a reservation satisfying given criteria. We search through possible time windows and make a reservation with the preference of selecting the one which gives earliest completion and shortest transfer duration. If there is no contiguous period of time with enough resource capacity in the given search interval between t^E and t^L, we start exploring possible options to move previously made time reservations to open a contiguous time slot that could satisfy the resource requirements of the new job request.

In this phase, we search over time windows for the new request, and look for jobs with less preference value that have allocation in the time window we are traversing. If there are jobs which

Input: A set of admitted jobs (already in the system) and their active advance reservations
Input: A new job request J_i with earliest start t^E, latest completion t^L, volume M, source v^s and destination v^d
Get all time windows which contains all time steps between t^E and t^L ;
Check available capacity for v^s, v^d and link $v^s \rightarrow v^d$ in time window tw ;
Search time windows to find a reservation satisfying given criteria ;
Time window search sequence : According to earliest completion time but time window with shortest duration preferred first ;
if *A suitable time window satisfying resource constraints found* **then**
  Make allocation and admit the job;
else
  for *All time windows that could satisfy the new request* **do**
    For time window tw, search if there is a job with less preference;
    Omit jobs which are flagged not to be displaced ;
    Request that are already started or already displaced in the current search sequence are also omitted ;
    if *There is a job J_k with less preference* **then**
      if *We can make a reservation for J_i by Displacing J_k* **then**
        Displace J_k and Make a reservation for J_i ;
        J_k and J_i are flagged ;
        Run Algorithm 2 for J_k to find a new reservation ;
if *All jobs in the system (including J_i) are scheduled* **then**
  Admit the new job J_i;
  Accept the new allocation (job-to-resource) mapping by committing the final Reservation Set ;
else
  Rollover to the initial state (job-to-resource mapping) ;

Algorithm 2: Scheduling Algorithm: displace previously admitted jobs if necessary to open space for the new request.

have less preference for this time period, we select the job with minimum preference. We move out this job and take over its time allocation to make a temporary reservation for the new request. The job which recently moved out from its allocated time space starts competing with other jobs to find a new slot. This recursive operation continues until no reservation left to shift out in worst-case. Algorithm 2 gives a glimpse of the methodology used to search and find a new advance resource assignment, in order to accept a new job, and displace a previously admitted jobs if necessary to open space for the new request

4.2 Evaluation

For each new job, we divide the search interval into time steps. The search interval $[t^E, t^L]$ of a job is the time period between earliest start time t^E and latest completion time t^L in which the data need to be transmitted. A time step represents the longest duration of time in which we have a stable discrete status in terms of available capacity over the link and the data transfer nodes. The set of confirmed reservations in the system characterizes time steps since each reservation modifies available capacity in the topology.

If there are r committed reservations falling into the period, there can be maximum $2r+1$ different time steps in the worst-case. If s is the total number of time steps, there are $(s \times (s+1))/2$ time windows since time windows are subsequent combinations of time steps. We search through these time windows in a sequential order to check whether we can satisfy the requested allocation in that time window. Overall, the worst-case complexity is bounded by $O(r^2)$. Time steps are associated with reservations and the total number linearly scales with the number of reservations in the system. Therefore, worst-case complexity for Algorithm 1 is $O(n^2)$.

A new data transfer request is only admitted only if we could allocate time and resource capacity in advance without breaking the constraints of previously admitted jobs. In Algorithm 2, if we can still find a space for all previously admitted jobs and the new request, we admit the new request and make the temporarily made reservations permanent. Otherwise, we roll back all temporary reservations and return back to the previous state. We try and execute the same search procedure for other possible time windows that this new request can reserve. If we succeed in none of them, we could not end up finding a schedule satisfying all admitted job and this new request, we either reject the new request or suggest a new latest completion time.

Assume that there are already n jobs in the system which have already been admitted. When we receive the $(n+1)^{th}$ job, and we could not confirm a reservation just by looking time windows it can span over, we try to displace other jobs to open space for this recent request. We sequentially traverse time windows that can satisfy given criteria, and try to find a job with less preference that already has allocation in the time window we are considering. As it has been described above, this recursive process will end when we cannot place a previously admitted job. Therefore, there can be maximum n tries. Thus, total complexity is bounded by number of jobs and number of time windows, $O(n \times s^2)$. In a very extreme case where all jobs fall into same search interval, complexity of Algorithm 2 is $O(n^3)$ in worst-case.

4.3 Preference Metric

Assigning a preference value is important in the design of a greedy heuristic algorithm. Even though we assign random ranks to each transfer request, the algorithm in general will conclude with a scheduling decision. Since no previously admitted job will be displaced in order to allocate resources for a new request, we guarantee that scheduler will eventually come up with a decision that satisfies users by making reservations based on their criteria. However, we would like to have a good selection metric in order to have an efficient algorithm. Therefore, we define the following preference metrics.

- P_1: $(t^L - t_i^e)/(t^L - t^E)$. The first metric defines time left to complete the job before its deadline, proportional to the duration of the search interval between earliest start time t^E and latest completion time t^L. A job that has lower P_1 value has higher preference.
- P_2: $ts_{J_i}^{num}/ts_{J_i}^{total}$, where ts^{num} is the total number of time steps in the current time window assigned to the job that we are examining, and ts^{total} is the total number of time steps that this job can span over to make its reservations. We prefer to assign a job to a time window which includes less time steps. Therefore, we favor a job which has already been using more time steps compared to the total number of time steps it can cover. A job assigned to a time window with higher preference has better chance to have its transfer overlapping with other transfer operations.
- P_3: tw^{id}/tw^{num}, where tw^{id} is the index of the current assigned time window, and tw^{num} is the total number of time windows associated for this job. For a recently arrived job tw^{id} represents the current time window we are evaluating to allocate. We compare its preference with other jobs that are already using this time window. A job with higher P_3 value is more close to its deadline; so, it has higher preference.
- P_4: $(t^L - t_i^s)/(t^L - t^E)$. The last metric is related to the start time. A job that has started earlier relative to the search interval ($[t^E, t^L]$) has lower preference. We favor the jobs that has higher P_5 value. For P_2, P_3, and P_4, higher value means higher preference. Those jobs with lower preference metric are likely to be displaced to open up space for the jobs with higher preference values. For P_1, higher value means more time to deadline, so less preference.

We have implemented Algorithm 1 and Algorithm 2, and developed a simple simulator that generates random topologies. In order to test the performance of the given metrics under heavy system load, we generated random jobs with search interval, time period between latest completion time and earliest start time, limited to a maximum of two days. User parameters such as data volume, source and destination data transfer nodes, earliest start time and latest end time are all set randomly. Available node capacity, and network capacity connecting data nodes are also random. Default capacity of data nodes are generated between 5Gbps and 10Gbps, available network bandwidth is between 1Gbps minimum and 10Gbps maximum. In this experimental setup, there are no separate upload and download capacities. A connection capacity between two nodes is randomly generated and it is shared for each direction.

Table 1: Test results for 1000 nodes
n_j: number of submitted jobs, n_r: number of rejected jobs, n_i: iteration count, t: elapsed time in millisecs

	Set 1				Set 2			
	n_j	n_r	n_i	t	n_j	n_r	n_i	t
Alg^1	500	343	500	4	500	282	500	5
	1000	710	1000	18	1000	648	1000	22
	1500	1112	1500	42	1500	1038	1500	52
	2000	1528	2000	78	2000	1469	2000	96
	2500	1951	2500	127	2500	1895	2500	157
Alg^2	500	341	509	5	500	280	523	7
P_1	1000	699	1069	28	1000	638	1091	34
	1500	1108	1568	61	1500	1021	1682	84
	2000	1520	2248	130	2000	1448	2590	195
	2500	1936	2903	233	2500	1869	3250	333
Alg^2	500	341	504	5	500	278	517	7
P_2	1000	702	1055	27	1000	641	1065	33
	1500	1108	1562	61	1500	1023	1660	82
	2000	1518	2216	128	2000	1449	2569	195
	2500	1936	2882	233	2500	1870	3217	324
Alg^2	500	341	504	5	500	278	519	7
P_3	1000	702	1054	28	1000	641	1070	33
	1500	1108	1562	61	1500	1023	1661	82
	2000	1518	2219	128	2000	1449	2549	192
	2500	1936	2884	233	2500	1870	3214	324
Alg^2	500	342	503	5	500	281	514	7
P_4	1000	709	1011	25	1000	648	1020	30
	1500	1109	1530	59	1500	1031	1553	74
	2000	1524	2074	114	2000	1461	2195	156
	2500	1944	2653	201	2500	1883	2719	253

Table 2: Test results for 1000 nodes
n_j: number of submitted jobs, n_r: number of rejected jobs, n_i: iteration count, t: elapsed time in millisecs

	Set 1				Set 3			
	n_j	n_r	n_i	t	n_j	n_r	n_i	t
Alg^1	2000	1445	2000	79	2000	44	2000	312
	2500	1817	2500	119	2500	98	2500	710
Alg^2	2000	1427	2124	120	2000	10	2818	561
P_1	2500	1803	2621	175	2500	12	3279	1532
Alg^2	2000	1430	2104	118	2000	11	2814	562
P_2	2500	1804	2596	174	2500	12	7556	4711
Alg^2	2000	1430	2104	117	2000	10	2806	576
P_3	2500	1804	2596	172	2500	12	3417	1762
Alg^2	2000	1436	2047	111	2000	19	2136	428
P_4	2500	1810	2538	167	2500	30	3078	1362

We have defined three test sets. In *Set 1*, data transfer jobs are generated randomly. In *Set 2*, jobs have higher chance to get accepted since their data sizes are proportional to the maximum network bandwidth connecting source and destination. In *Set 3*, we calculate the data volume size by multiplying total duration between t^E and t^L by the minimum link capacity in the topology. In *Set 3*, we will have the highest acceptance rate, and in *Set 1* we will have highest number of rejected jobs.

For each job, we also generate a submission time. We simulate a real-life scenario where data transfer operations are submitted independently. We have performed minimum 50 test runs for each case. We took average of the test results from measurements, and rounded them to the nearest integer. These test are conducted on a mid-range workstation with 2.5GHz Intel CPU and 4G RAM. We have collected number total elapsed time along with the number of scheduling iterations to measure the effectiveness of Algorithm 2. A search sequence is triggered in each iteration. In Algorithm 1, the iteration count is equal to the number of jobs. In Algorithm 2, it is higher than the number of jobs

submitted since we might displace other jobs and initiate a new search to come up with a better scheduling decision. Table 1 and Table 2 show our initial results.

Table 3: Test results for 100 nodes and 250 jobs
n_r: number of rejected jobs, n_i: iteration count, t: elapsed time in millisecs

		n_r	n_i	t
$Set1$	Alg^1	182	250	1
	Alg^2 P_3	172	299	2
	Exponential-ALL	172	17577	283
$Set2$	Alg^1	158	250	1
	Alg^2 P_3	155	329	3
	Exponential-ALL	155	126508	2466

According to experimental results, P_2 and P_3 are better than others preference metrics in general. The number of iterations is mostly bounded by the number of nodes - far away from the worst case scenario $O(n^3)$. Furthermore, total time required to make scheduling decisions for all jobs submitted is less than a second. The scheduling algorithm is efficient and easily applicable to real-life scenarios. On the other hand, we also compare the competitiveness of the greedy heuristic. We implemented a special case in which all possible assignments are examined for best scheduling decision. The number of reservation options increases exponentially. Therefore, we could only test this special case for small number of jobs. As can be seen in Table 3, Algorithm 2 with preference metric P_3 gives same results in a very efficient manner. Experimental results verify that our proposed approach produces near optimum results.

5 Conclusion

We have developed a new scheduling model considering resource allocation in client sites and bandwidth allocation on network link connecting resources. Our model provides a basis for provisioning end-to-end high performance data transfers in which users submit their jobs with time and resource constraints to make an advance schedule. The focus of our work is to optimize data scheduling in the wide-area backbone. Other projects such as TeraPaths and LamdaStation address reservations between the clients and edge routers, whereas OSCARS addresses reservation between edge routers. Our scheduler interacts with reservation managers and queries resource availability for a certain period of time. Once we displace a previously admitted job in order to open space for a new request, we require specific interface to communicate with the reservation managers and temporarily hold a reservation. If the scheduler commits the new state, it will communicate and ensure that all temporary reservations in end points are also replaced. Current reservation systems do not provide such capabilities at this moment.

We have analyzed time-dependent resource assignment problem with bottleneck constraints, and presented a detailed study of data transfer scheduling with resource and time conflicts. We have proposed an efficient heuristic for scheduling data placement operations with advance reservation. We have implemented our algorithm and examined possible techniques for incorporation into current reservation frameworks. Performance measurements confirmed that our proposed algorithm is efficient and scalable.

Acknowledgments

This project is in part sponsored by the National Science Foundation under award numbers CNS-0846052, CNS-0619843, and OCI-0926701, and in part supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract no. DE-AC02-05CH11231.

References

[1] Advance Network Testbed for Research and Development. http://www.jgn.nict.go.jp/english/.

[2] Dynamic Resource Allocation via GMPLS Optical Networks. http://dragon.maxgigapop.net/.

[3] Energy Sciences Network. http://www.es.net.

[4] High speed TransAtlantic network for the LHC Community. http://lhcnet.caltech.edu/.

[5] Internet2. www.internet2.edu.

[6] OSCARS: On-demand secure circuits and advance reservation system. www.es.net/oscars.

[7] G. Aldering and SNAP Collaboration. Overview of the supernova/acceleration probe (snap). http://www.citebase.org/abstract?id=oai:arXiv.org:astro-ph/0209550, 2002.

[8] Bill Allcock, Ian Foster, Veronika Nefedova, Ann Chervenak, Ewa Deelman, Carl Kesselman, Jason Lee, Alex Sim, Arie Shoshani, Bob Drach, and Dean Williams. High-performance remote access to climate simulation data: A challenge problem for data grid technologies. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 46–46, New York, NY, USA, 2001. ACM Press.

[9] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. *Parallel Computing. 2001.*, 2001.

[10] Yossi Azar and Oded Regev. Strongly Polynomial Algorithms for the Unsplittable Flow Problem. In *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization (IPCO*, pages 15–29, 2001.

[11] Mehmet Balman, Evangelos Chaniotakisy, Arie Shoshani, and Alex Sim. A flexible reservation algorithm for advance network provisioning. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[12] Mehmet Balman and Tevfik Kosar. Dynamic Adaptation of Parallelism Level in Data Transfer Scheduling. *International Workshop on Adaptive Systems in Heterogeneous Environments (ASHEs 2009), Fukuoka, Japan*, 2009.

[13] Mehmet Balman and Tevfik Kosar. Data Scheduling for Large Scale Distributed Applications. In *the 5th ICEIS Doctoral Consortium, In conjunction with the International Conference on Enterprise Information Systems (ICEIS'07)*. Funchal, Madeira-Portugal, June, 2007.

[14] Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 702–709, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[15] Uyless N. Black. *MPLS and Label Switching Networks (2nd Edition)*. Printece-Hall series on advance communicaiton technologies.

[16] Lars-Olof Burchard. Networks with advance reservations: Applications, architecture, and performance. *J. Netw. Syst. Manage.*, 13(4):429–449, 2005.

[17] Alexandra Carpen-Amarie, Mugurel Andreica, and Valentin Cristea. An algorithm for file transfer scheduling in grid environments. http://arxiv.org/pdf/0901.0291, 2009.

[18] Cern. The world's largest particle physics laboratory. http://public.web.cern.ch, 2006.

[19] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. In *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 51–66, London, UK, 2002. Springer-Verlag.

[20] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers in a distributed network. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 254–266, New York, NY, USA, 1983. ACM.

[21] R. Cohen, N. Fazlollahi, and D. Starobinski. Graded channel reservation with path switching in ultra high capacity networks. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pages 1–10, Oct. 2006.

[22] Sang Son Department and Sang H. Son. A priority-based scheduling algorithm for real-time databases.

[23] Eduard Escalona, Salvatore Spadaro, Jaume Comellas, and Gabriel Junyent. Advance reservations for service-aware gmpls-based optical networks. *Comput. Netw.*, 52(10):1938–1950, 2008.

[24] Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *J. of Scheduling*, 12(2):199–224, 2009.

[25] Barak Farzad, Neil Olver, and Adrian Vetta. A priority-based model of routing, 2008.

[26] Ian Foster, Carl Kesselman, Craig Lee, Bob Lindell, Klara Nahrstedt, and Alain Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *In Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[27] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid using reinforcement learning. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 3:1314–1315, 2004.

[28] R. Guerin and A. Orda. Networks with advance reservations: the routing perspective. INFOCOMM 2000, 2000.

[29] Tony Hey and Anne Trefethen. The Data Deluge: An e-science perspective. *Grid Computing: Making the Global Infrastructure a Reality. Chichester, UK: John Wiley & Sons, Ltd.*, pages 809–824, 2003.

[30] Sandy Irani and Vitus Leung. Scheduling with conflicts on bipartite and interval graphs. *J. of Scheduling*, 6(3):287–307, 2003.

[31] Edward G. Coffman Jr., M. R. Garey, David S. Johnson, and Andrea S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(4):743–780, 1985.

[32] Eun-Sung Jung, Yan Li, Sanjay Ranka, and Sartaj Sahni. An evaluation of in-advance bandwidth scheduling algorithms for connection-oriented networks. In *ISPAN '08: Proceedings of the The International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 133–138, Washington, DC, USA, 2008. IEEE Computer Society.

[33] Jon M. Kleinberg. Single-source unsplittable flow. In *In Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 68–77, 1996.

[34] Stavros G. Kolliopoulos and Clifford Stein. Improved approximation algorithms for unsplittable flow problems (extended abstract). In *In Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 426–435, 1997.

[35] Petr Kolman. A note on the greedy algorithm for the unsplittable flow problem. *Information Processing Letters*, 88(3):101 – 105, 2003.

[36] Tevfik Kosar and Mehmet Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, Vol.25 No.4, pp.406-413, 2009.

[37] Zhaoming Li, Qiang Song, and Ibrahim Habib. Cheetah virtual label switching router for dynamic provisioning in ip optical networks. *Optical Switching and Networking*, 5(2-3):139–149, 2008. Advances in IP-Optical Networking for IP Quad-play Traffic and Services.

[38] Yunyue Lin and Qishi Wu. On design of bandwidth scheduling algorithms for multiple data transfers in dedicated networks. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 151–160, New York, NY, USA, 2008. ACM.

[39] Florin Manea and Calina Ploscaru. Solving a combinatorial problem with network flows, 2004.

[40] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[41] The Office of Science Data-Management Challenge. Report from the doe office of science data-management workshops, March-May 2004.

[42] Kannan Rajah, Sanjay Ranka, and Ye Xia. Scheduling bulk file transfers with start and end times. *Comput. Netw.*, 52(5):1105–1122, 2008.

[43] N. S. V. Rao, W. R. Wing, S. M. Carter, and Q. Wu. Ultrascience net: network testbed for large-scale science applications. *Communications Magazine, IEEE*, 43(11):S12–S17, 2005.

[44] N.S.V. Rao, Qishi Wu, Song Ding, S.M. Carter, W.R. Wing, A. Banerjee, D. Ghosal, and B. Mukherjee. Control plane for advance bandwidth scheduling in ultra high-speed networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–5, April 2006.

[45] Biju K Raveendran, Sundar Balasubramaniam, and S Gurunarayanan. Evaluation of priority based real time scheduling algorithms: choices and tradeoffs. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 302–307, New York, NY, USA, 2008. ACM.

[46] Olov Schelen and Stephen Pink. An agent-based architecture for advance reservations. *Local Computer Networks, Annual IEEE Conference on*, 0:451, 1997.

[47] Farhad Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.

[48] Arie Shoshani, Alexander Sim, and Junmin Gu. *Storage Resource Managers: Essential Components for the Grid*. Kluwer Academic Publishers, 2003.

[49] Rok Sosic and Jun Gu. A polynomial time algorithm for the n-queens problem. *SIGART Bull.*, 1(3):7–11, 1990.

[50] Sebastien Soudan, Bin Bin Chen, and Pascale Vicat-Blanc Primet. Flow scheduling and endpoint rate control in gridnetworks. *Future Gener. Comput. Syst.*, 25(8):904–911, 2009.

[51] Savera Tanwir, Lina Battestilli, Harry Perros, and Gigi Karmous-Edwards. Dynamic scheduling of network resources with advance reservations in optical grids. *Int. J. Netw. Manag.*, 18(2):79–105, 2008.

[52] Chung-Piaw Teo, Jay Sethuraman, and Wee-Peng Tan. Gale-shapley stable marriage problem revisited: Strategic issues and applications. *Manage. Sci.*, 47(9):1252–1267, 2001.

[53] TeraPaths. Configuring end-to-end virtual network paths with qos guarantees.

[54] J. A. Tyson. Large Synoptic Survey Telescope: Overview. In J. A. Tyson and S. Wolff, editors, *Survey and Other Telescope Technologies and Discoveries. Edited by Tyson, J. Anthony; Wolff, Sidney. Proceedings of the SPIE, Volume 4836, pp. 10-20 (2002).*, pages 10–20, December 2002.

[55] Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 75–80, 2004.

[56] Esma Yildirim, Mehmet Balman, and Tevfik Kosar. Dynamically tuning level of parallelism in wide area data transfers. In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 39–48, New York, NY, USA, 2008. ACM.