Advance Resource Provisioning in Bulk Data Scheduling

Mehmet Balman
Lawrence Berkeley National Laboratory
Berkeley, CA 94720, USA Email:mbalman@lbl.gov

***Abstract*—Today's scientific and business applications generate massive data sets that need to be transferred to remote sites for sharing, processing, and long term storage. Because of increasing data volumes and enhancement in current network technology that provide on-demand high-speed data access between collaborating institutions, data handling and scheduling problems have reached a new scale. In this paper, we present a new data scheduling model with advance resource provisioning, in which data movement operations are defined with earliest start and latest completion times. We analyze time-dependent resource assignment problem, and propose a new methodology to improve the current systems by allowing researchers and higher-level meta-schedulers to use data-placement as-a-service, so they can plan ahead and submit transfer requests in advance. In general, scheduling with time and resource conflicts is NP-hard. We introduce an efficient algorithm to organize multiple requests on the fly, while satisfying users' time and resource constraints. We successfully tested our algorithm in a simple benchmark simulator that we have developed, and demonstrated its performance with initial test results.**

***Keywords*-scheduling with constraints, bulk data movement, time-dependent graphs, network virtualization, Gale-Shapley algorithm**

I. Introduction

Scientific and business applications require geographically distributed resources to satisfy their large compute and storage requirements. In addition to increasing data volumes and computational needs, many science projects require cooperative work. For example, in climate research, many researchers around the world work together to model complex natural ecosystems [1], [2]. Collaboration between multiple institutions requires complex middleware to orchestrate the use of storage and network resources, and to manage end-to-end processing of data.

In the age of high-bandwidth networks [3], [4], [5], challenges facing large-scale data transfer is a reality. One of the open problems is the dearth of robust, economic data scheduling models for time-sensitive data movements in future research infrastructures. Current data scheduling systems [6], [7] manage data transfer jobs by trying to optimize for performance and resource utilization. However, they do not take time constraints into account when running scheduling algorithms.

Many scientific analysis programs need recurrent and time sensitive movement of large-scale datasets. Current network technology provides on-demand high-speed data access between collaborating institutions [8]. Users are able to reserve a specific amount of the available bandwidth for predictable performance and timely completion of data transfer operations. As a result, we need novel mechanisms to organize advance reservation requests intelligently. A data movement operation is also affected by client/server side performance (such as memory, CPU, storage bottlenecks). In addition to reservation of network bandwidth for desired transfer throughput, we also necessitate provisioning of server capacity before initiating a data movement operation.

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 provide the total volume of data needs to be transferred between source and destination points. In addition to resource constraints, each data transfer job includes time constraints in which users state the earliest start time and latest completion time. Earliest start time specifies when the source data set will be ready to start transferring data. Latest completion time specifies a desired deadline to complete the transfer operation.

In our previous research, we presented a bulk data scheduling model with advance reservation and provisioning [9]. In this model, users can plan ahead and reserve time in advance for their data movement operations. The data scheduler interacts with the data transfer nodes to provision server capacity, and network reservation systems [8], [10], [11] to allocate network bandwidth for guaranteed completion of data movement operations. In this paper, we extend the methodology introduced in [9] with detailed analysis and experimental results. In general, the number of reservation options, to allocate server and network capacity in a given time interval, is exponential, so the scheduling problem is NP-hard. Therefore, we propose an efficient heuristic for scheduling data placement operations with advance reservation. We successfully tested our algorithm in a simple benchmark simulator that we have developed, and demonstrated its performance with initial test results. Performance measurements confirm that the proposed algorithm is efficient and scalable.

II. Motivating Remarks

A very simple use case can be explained as follows. Consider a scientific application that generates very large amount of simulation data using supercomputing resources. The generated data needs to be analyzed for validation and verification by collaborating researchers. In most cases, post-processing of the data is performed by a different team located in a remote site [12]. In addition to running the simulation program that generated the data, verification and validation steps also necessitate allocation of many compute hours and reservation of fast temporary storage space, in order to locally process and run analysis programs. The data analysis phase will be waiting for the generated data to arrive. If data transfer completes later than the expected time, allocated resources will stay idle. Therefore, advance planning for timely completion of data movement operations is crucial, in order to utilize allocation of resources efficiently.

Today's best-effort networks give equal priority to all data flows and do not distinguish high-priority and time sensitive data movement operations with the rest of competing data flows in the network. In order to support Quality of Service (QoS) requirements of data-intensive scientific applications, next generation research networks such as Internet2 [13] and ESnet (Energy Sciences Network) [14] provide bandwidth guaranteed on-demand data access between collaboration sites. Network reservation systems such as ESnet's OSCARS enable necessary infrastructure to allocate bandwidth in advance. A specific amount of the available bandwidth between two sites is reserved for a given duration [8], [15], in order to guarantee completion of time sensitive data movement operations. Advance bandwidth allocation helps satisfy QoS requirements, enables predictable performance, and helps make accurate planning for allocation of resources.

There are several studies concentrating on data management in scientific applications [16], [17], [18]; however, most of the existing systems fail to address issues such as scheduling according to given user requirements, time constraints, and priorities. In current systems [6], [12], a data transfer request is managed by a scheduler without time 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.

In our proposed model, data scheduler confirms requests after checking availability of resources and other submitted tasks in the given time frame. It considers future time windows and examines available capacity both in network and server resources to make a new reservation that satisfies the requirements of a submitted transfer job. Once a job is accepted, a temporary reservation is made. While scheduling new jobs, we may also need to change the temporary reservations that belongs to the jobs that are accepted but 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 completion deadline. Data scheduler should operate in an opportunistic manner to maximize 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.

III. Problem Definition

We define the overall 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 node v_i to node 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, $u^{v_i}_{out}$ and $u^{v_j}_{in}$ respectively.

We have a dynamic network environment in which edge capacities may vary over time. The connection between two end-points may span over multiple routers in the network, as can be seen in Figure 1. Searching available network bandwidth between end-points and finding possible network reservation options are studied in [15]. In this study, we specifically focus only on the total available bandwidth between end-points. e_{ij} represents the virtual connection from v_i to v_j. $u^{e_{ij}}(t)$ represents the time-dependent available bandwidth information from v_i to v_j. On-demand reservation systems such as OSCARS are able to make bandwidth reservations between two edge-routers [8], [15]. In our scheduling model, data scheduler interacts directly with a network reservation manager and queries available bandwidth information between two end-points.

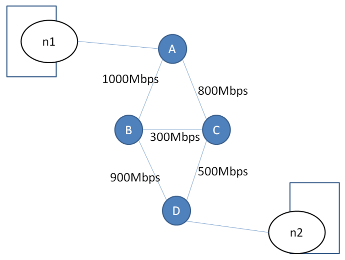

Figure 1. Connection between two data nodes

We consider data transfer nodes (DTNs) [19] as specialized machine(s), with back-end storage servers and high-speed data transfer protocols, connected to the outside network with high-bandwidth interconnects. We assume that we know the maximum upload and download capacities in each data transfer node.

A data transfer job is defined as $J_i = (v^s_i, v^d_i, M_i, t^E_i, t^L_i)$, where M_i is the amount of data to be transferred from source v^s_i to destination node v^d_i within the time period of $[t^E_i, t^L_i]$. t^E_i represents the earliest possible time when this data will be ready to start the transfer operation. t^L_i represents the latest completion time the transfer operation needs to be finished. Data 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 $u_{in}^{v_i^d}$, and total outgoing bandwidth allocation should be less than $u_{out}^{v_i^s}$, 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^d, bandwidth of link e_{ij}, and download capacity of v_i^s. 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. We can simplify the problem as finding a contiguous set of time slots such that a fixed amount of bandwidth can be allocated to satisfy the data transfer request.

Data transfer scheduler checks other jobs in the system and considers both time and resource conflicts. In order to admit a job, it has to confirm the availability of bandwidth between end-points and upload/download capacities in data transfer nodes, within the given time constraints. If a job has been admitted, an initial decision is made on when this job will start/end. In addition to the period of time reserved, temporary resource (network bandwidth and server capacity) reservations are also created in advance. 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 a contiguous time slot for each job. A data transfer operation starts with fixed transfer rate and maintains this until the entire data has completely transferred. Current network reservations systems provide guaranteed bandwidth for a contiguous period of time between two endpoints [8], [11]. In order to benefit from on-demand bandwidth allocation systems and provide predictable guaranteed performance, we also enforce this constraint in our scheduling model.

The scheduler has two main objectives. First, it ensures that no other admitted job will be postponed because of accepting a new job. In addition to this fairness objective, it also tries to maximize the number of admitted jobs by moving reserved slots appropriately. It tries to create suitable periods of time to accept jobs by resolving time and resource conflicts. On the other hand, it also selects time slots that gives earliest completion time with minimum interference with other admitted jobs in the system.

IV. Time and Resource Conflicts

We define a sample network with three data transfer nodes connected to each other over a network, given in Figure 2. Each node has particular upload and download capacities. Maximum upload and download rates define the limit in server site; such that, total transfer 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 job need to be completed - earliest start time t^E, and latest completion time t^L.

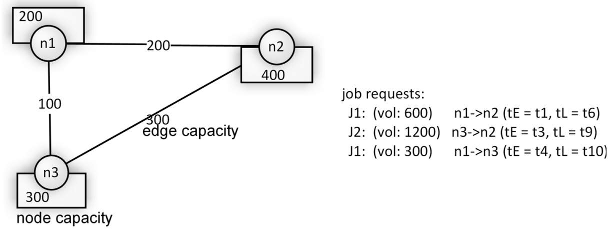
200
n1
200
n2
400
100
300
edge capacity
n3
300
node capacity
job requests:
J1: (vol: 600) n1->n2 (tE = t1, tL = t6)
J2: (vol: 1200) n3->n2 (tE = t3, tL = t9)
J1: (vol: 300) n1->n3 (tE = t4, tL = t10)

Figure 2. Sample Problem Definition

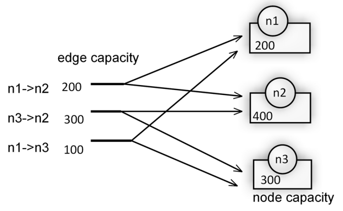

Figure 3. Node and Link - Resource Constraints

We are bounded by edge capacity as well as node capacities. Figure 3 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 node n_2.

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 time constraints of jobs. We divide the search interval into time steps to analyze the time-dependent topology graph. Available network and server capacities are fixed in each time step. A time step shows the longest duration of time in which we have a stable network topology [15]. A time window is a sequence of time steps. We traverse time

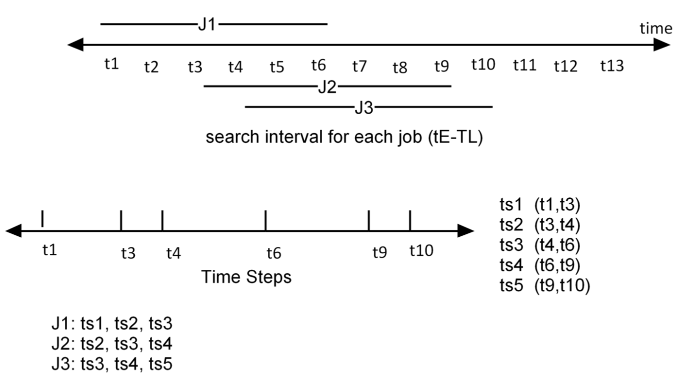

Figure 4. Time Steps and Search Interval

windows in a specific order, as shown in Figure 5. 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.

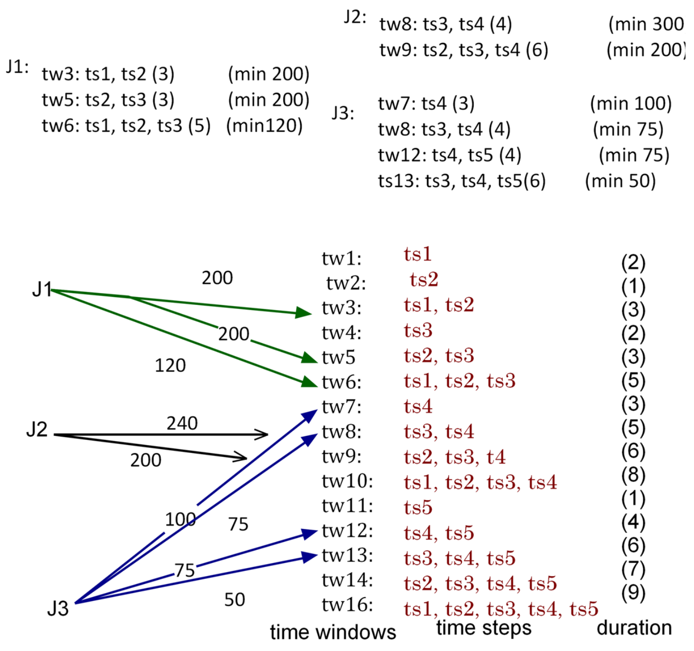
J1:
tw3: ts1, ts2 (3) (min 200)
tw5: ts2, ts3 (3) (min 200)
tw6: ts1, ts2, ts3 (5) (min120)
J2:
tw8: ts3, ts4 (4) (min 300
tw9: ts2, ts3, ts4 (6) (min 200)
J3:
tw7: ts4 (3) (min 100)
tw8: ts3, ts4 (4) (min 75)
tw12: ts4, ts5 (4) (min 75)
ts13: ts3, ts4, ts5(6) (min 50)
J1
J2
J3
200
200
120
240
200
100
75
75
50
tw1: ts1 (2)
tw2: ts2 (1)
tw3: ts1, ts2 (3)
tw4: ts3 (2)
tw5 ts2, ts3 (3)
tw6: ts1, ts2, ts3 (5)
tw7: ts4 (3)
tw8: ts3, ts4 (5)
tw9: ts2, ts3, t4 (6)
tw10: ts1, ts2, ts3, ts4 (8)
tw11: ts5 (1)
tw12: ts4, ts5 (4)
tw13: ts3, ts4, ts5 (6)
tw14: ts2, ts3, ts4, ts5 (7)
tw16: ts1, ts2, ts3, ts4, ts5 (9)
time windows
time steps
duration

Figure 5. Minimum Capacity required for each Time Window

Figure 5 shows how several time windows are eliminated in the search interval. Further, it also illustrates the resource constraints specific to each time windows. For example, if we want to assign job J_1 to 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 table of possible assignment options

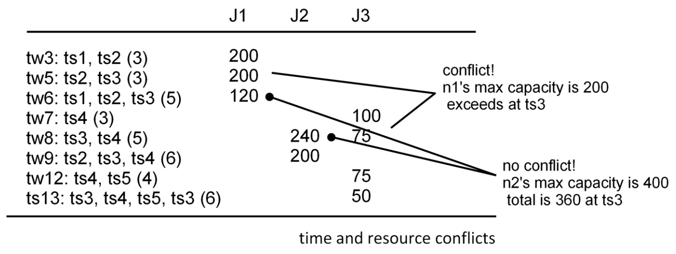

Figure 6. Time and Resource Conflicts

that need to be considered with resource constraints given in Figure 7. In Figure 8, we present how solution space is analyzed in this sample problem. We show the 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. In other words, our problem complexity increases exponentially when we have time constraints and resource constraints together.

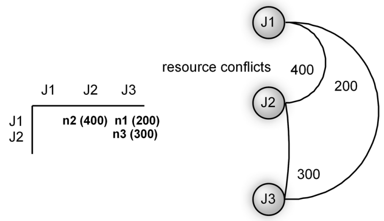

Figure 7. Resource Conflicts

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.

We can use unsplittable flow problem to model and clarify our problem domain. The unsplittable flow problem [20] is an interesting dilemma in algorithm research. The unsplittable flow problem is NP-hard, and only polynomial approximation algorithms are given as a solution [21], [22], [23], [24], [25]. Interestingly, even for very special cases (i.e. planar graphs), the problem is still NP-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.

There are many studies in the literature investigation approximation algorithms for scheduling; [26] and [27] are some of them which show benefits in designing greedy

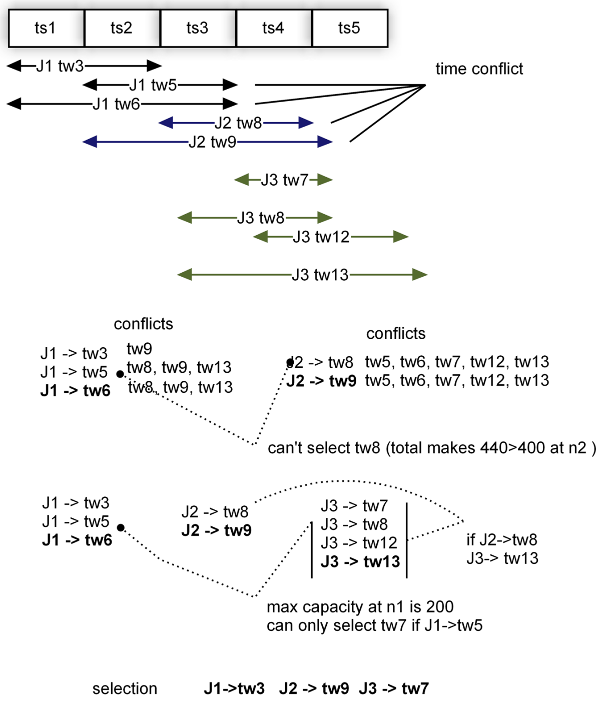

Figure 8. Assigning Jobs to Time Windows

algorithms with priorities. The number of possible options increases exponentially in worst case. A common approach is to set priorities for the possible options that need to be examined. We rate each option based on the priority or the cost/desire we assign, and we select the best to reduce the search space. 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. Deciding on a good selection criteria plays an important role in designing polynomial greedy heuristics for near optimum scheduling results.

V. Online Scheduling

We propose an online scheduler that makes decision when a new job is submitted. Our approach is to design an effective methodology that is easy to implement and deploy in practice. Algorithm 1 gives a simple greedy heuristic. The problem in Algorithm 1 is that data transfer operations are scheduled in submission order. In order to maximize the number of jobs admitted, we introduce Algorithm 2. Inspired from Gale-Shapley stable marriage problem [28] and N-queen [29] algorithm, we developed an efficient methodology to organize multiple requests on the fly, while satisfying users' time and resource constraints. When a new job request cannot find a suitable time slot to make a reservation, it competes with previously admitted jobs to move their reserved slots and open a proper reservation time for itself.

```
Input: A set of admitted jobs (already in the system) and
       their active advance reservations
Input: A new job $J_i$ 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 $tw_{j-i}$ which contains all time
        steps between $ts_j$ and $ts_i$;
        Check available capacity for $v^s$, $v^d$, and $v^s \rightarrow v^d$
        link in time window $tw_{j-i}$;
        if the given criteria can fit into the time window
        $tw_{j-i} = ts_j \ldots ts_i$ then
            Make allocation and admit the job;
Return: No reservation found;
```

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

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 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 in the time window we are traversing. If there are jobs which have less preference, 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 reservations.

A. 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 status in terms of available capacities.

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

```
Input: A set of admitted jobs (already in the system) and
       their active advance reservations (Reservation Set)
Input: A new job $J_i$ with earliest start $t^E$, latest completion
       $t^L$, volume $M$, source $v^s$ and destination $v^d$
Search all time windows between $t^E$ and $t^L$ ;
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 each time window $tw$, search if there is a job
        with less preference;
        Exclude jobs which are flagged; already started
        request 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
        Reject the new job;
        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.

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, we admit the new request and make the temporarily 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 fit. If we succeed in none of them, we reject the new request.

Assume that there are already n admitted jobs in the system. When we receive the $(n+1)^{th}$ job, and we could not confirm a reservation just by looking time windows it can span over. In that case, we try to displace other jobs to open space for this recent request. We sequentially traverse time windows that can satisfy given criteria, and we try to find a job with less preference with an allocation in the time window we are considering. As it has been described above, this recursive process will end when we cannot replace 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, $O(n \times s^2)$. Therefore, worst case complexity of Algorithm 2 is $O(n^3)$.

VI. Preference Metric

Since previously admitted jobs will not be rejected in order to allocate resources for a new request, we guarantee that scheduler will eventually generate correct results satisfying user's given constraints. Even if we assign random ranks to each transfer request, the algorithm in general will conclude with a scheduling decision. However, we would like to have a good selection criteria 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 could use to make a reservation. We prefer to assign a job to a time window that consists of less time steps. Therefore, we favor a job which spans over more time steps, compared to the total number of time steps it can cover. A job with a higher $P2$ value has more chance to have its transfer overlapping with other transfer operations. Thus, a job with high P_2 value has higher preference.
- P_3: tw^{id}/tw^{num}, where tw^{id} is the index (according to the search order given in Figure 5) of the current assigned time window, and tw^{num} is the total number of possible time windows associated with this job. For a recently arrived job, tw^{id} represents the current time window we are evaluating. 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 its search interval ($[t^E, t^L]$) has lower preference. We favor the jobs that has higher P_4 value.

For P_2, P_3, and P_4, higher value means higher preference. Those jobs with lower preference 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 it means 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 a search interval (time period between latest completion

Table I
TEST RESULTS FOR 500 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 II
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

time and earliest start time) limited to a maximum of two days. User parameters such as data volumes, source and destination data transfer nodes, earliest start times and latest completion times are all set randomly. Available node capacities, and the network capacities 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.

We have defined three test sets. In *Set 1*, data transfer jobs are generated randomly. In *Set 2*, we generate data sizes proportional to the maximum network bandwidth connecting source and destination. *Set 2* will have higher number of accepted jobs. Set 1 and Set 2 might also include jobs trying to allocate resources far beyond the capacity of the network. In *Set 3*, we calculate the data volume size by multiplying total duration (time between t^E and t^L) by the minimum link capacity. *Set 3* will have the highest acceptance rate.

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 the 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 I and Table II show our initial results.

Table III
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
*Set*1	Alg^1	182	250	1
	Alg^2 P_3	172	299	2
	Exponential-ALL	172	17577	283
*Set*2	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 III, 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.

VII. RELATED WORK

Current research networks bring the ability to provision the communication channels when the data, especially large-scale massive data, is ready to be transferred [30], [31]. On-demand bandwidth reservation is usually supported by Multiple Protocol Label Switching (MPLS) in layer 3 [32],

[33]. In layer 2, a virtual secure circuit is setup between source and destination with a specific bandwidth over the connection. Internet 2 [13] and ESnet [14], [8] provide dynamic circuit infrastructure to establish on-demand guaranteed bandwidth connection between two endpoints.. TeraPaths [34] controls end-sites and allows creation of secure circuits within the site to support guaranteed bandwidth service. TeraPaths address reservations between the clients and edge routers, whereas OSCARS addresses reservation between edge routers.

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 [15]. In this study, we try to come up with a near optimum allocation pattern for multiple data transfer jobs with advance reservation.

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 [35]. In our data scheduling paradigm, we consider soft-deadline scheduling for data transfer requests. 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. We can allocate the server and the network capacity; however, it is difficult to guarantee the exact completion time due to possible failures, system problems. A job is not admitted if it is not possible to finish the transfer of the requested data before the given deadline.

There are several studies summarizing theoretical complexity of data transfer scheduling [36], [37], [38]. In [39], 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 [39] examines several network structures such as trees, bipartite graphs, networks with odd and even cycles, and provides a detailed complexity analysis 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 [40], [41], [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 [41]; 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 find a solution space for combinatorial optimization problems, is a common practice [40]. 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 [15].

VIII. Conclusion

In this study, 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. Our proposed model provides a basis for provisioning end-to-end high performance data transfers in which users submit their jobs with time and resource constraints. We have developed an efficient heuristic for scheduling data placement operations with advance reservation. We have implemented our algorithm and showed its performance with initial test results.

Acknowledgement

We would like to thank Prof. Tevfik Kosar (University at Buffalo-SUNY) for insightful discussions, valuable assistance, and helpful comments.

References

[1] D. N. Williams et al., “Earth System Grid Center for Enabling Technologies: Building a Global Infrastructure for Climate Change Research,” Scientific Discovery through Advanced Computing Conference Series, SciDAC’10 conference proceedings , 2010.

[2] ——, “Earth System Grid Center for Enabling Technologies: A Data Infrastructure for Data-Intensive Climate Research,” Scientific Discovery through Advanced Computing Conference Series, SciDAC’11 conference proceedings , 2011.

[3] M. Balman and S. Byna, “Open problems in network-aware data management in exa-scale computing and terabit networking era,” in *Proceedings of the first international workshop on Network-aware data management*, ser. NDM ’11, 2011, pp. 73–78. [Online]. Available: http://doi.acm.org/10.1145/2110217.2110229

[4] M. Balman, “Memznet: Memory-mapped zero-copy network channel for moving large datasets over 100gbps network (poster),” in *ACM/IEEE international Conference For High Performance Computing, Networking, Storage and Analysis*, 2012.

[5] M. Balman, E. Pouyoul, Y. Yao, E. W. Bethel, B. Loring, M. Prabhat, J. Shalf, A. Sim, and B. L. Tierney, “Experiences with 100gbps network applications,” in *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing, in conjunction with the International Symposium on High Performance Distributed Computing*, 2012.

[6] T. Kosar and M. Balman, “A new paradigm: Data-aware scheduling in grid computing,” *Future Generation Computer Systems*, vol. Vol.25 No.4, pp.406-413, 2009.

[7] Kosar, T.; Balman, M.; Yildirim, E.; Kulasekaran, S.; Ross, B., “Stork data scheduler: mitigating the data bottleneck in e-science,” in *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 369, issue 1949, pp. 3254-3267*, 2011.

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

[9] M. Balman, "Data Transfer Scheduling with Advance Reservation and Provisioning," *Ph.D. Dissertation, Louisiana State University, Baton Rouge, LA*.

[10] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston, "Intra and interdomain circuit provisioning using the oscars reservation system," in *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, Oct. 2006, pp. 1–8.

[11] M. Balman, E. Chaniotakis, A. Shoshani, and A. Sim, "Advance Network Reservation and Provisioning for Science," UK e-science All-hands Meeting, 2009.

[12] M. Balman and T. 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.

[13] "Internet2," www.internet2.edu.

[14] "Energy Sciences Network," http://www.es.net.

[15] M. Balman, E. Chaniotakisy, A. Shoshani, and A. Sim, "A flexible reservation algorithm for advance network provisioning," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.4

[16] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, and D. 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*, 2001, pp. 46–46.

[17] 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.

[18] S. Venugopal, R. Buyya, and L. 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*, 2004, pp. 75–80.

[19] "Science DMZ: Data transfer nodes," http://fasterdata.es.net/science-dmz/DTN/.

[20] P. Kolman, "A note on the greedy algorithm for the unsplittable flow problem," *Information Processing Letters*, vol. 88, no. 3, pp. 101 – 105, 2003.

[21] Y. Azar and O. Regev, "Strongly Polynomial Algorithms for the Unsplittable Flow Problem," in *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization (IPCO*, 2001, pp. 15–29.

[22] N. Bansal, Z. Friggstad, R. Khandekar, and M. 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*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009, pp. 702–709.

[23] S. G. Kolliopoulos and C. Stein, "Improved approximation algorithms for unsplittable flow problems (extended abstract)," in *In Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997, pp. 426–435.

[24] J. M. Kleinberg, "Single-source unsplittable flow," in *In Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, 1996, pp. 68–77.

[25] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, "Approximation algorithms for the unsplittable flow problem," in *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*. London, UK: Springer-Verlag, 2002, pp. 51–66.

[26] S. Irani and V. Leung, "Scheduling with conflicts on bipartite and interval graphs," *J. of Scheduling*, vol. 6, no. 3, pp. 287–307, 2003.

[27] B. Farzad, N. Olver, and A. Vetta, "A priority-based model of routing," 2008.

[28] C.-P. Teo, J. Sethuraman, and W.-P. Tan, "Gale-shapley stable marriage problem revisited: Strategic issues and applications," *Manage. Sci.*, vol. 47, no. 9, pp. 1252–1267, 2001.

[29] R. Sosic and J. Gu, "A polynomial time algorithm for the n-queens problem," *SIGART Bull.*, vol. 1, no. 3, pp. 7–11, 1990.

[30] Z. Li, Q. Song, and I. Habib, "Cheetah virtual label switching router for dynamic provisioning in ip optical networks," *Optical Switching and Networking*, vol. 5, no. 2-3, pp. 139–149, 2008, advances in IP-Optical Networking for IP Quad-play Traffic and Services.

[31] 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*, vol. 43, no. 11, pp. S12–S17, 2005.

[32] U. N. Black, *MPLS and Label Switching Networks (2nd Edition)*. Printece-Hall series on advance communicaiton technologies, 2002.

[33] E. Escalona, S. Spadaro, J. Comellas, and G. Junyent, "Advance reservations for service-aware gmpls-based optical networks," *Comput. Netw.*, vol. 52, no. 10, pp. 1938–1950, 2008.

[34] "TeraPaths: Configuring End-to-End Virtual Network Paths with QoS Guarantees," racf.bnl.gov/terapath.

[35] B. K. Raveendran, S. Balasubramaniam, and S. Gurunarayanan, "Evaluation of priority based real time scheduling algorithms: choices and tradeoffs," in *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2008, pp. 302–307.

[36] 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*. New York, NY, USA: ACM, 1983, pp. 254–266.

[37] G. Even, M. M. Halldórsson, L. Kaplan, and D. Ron, "Scheduling with conflicts: online and offline algorithms," *J. of Scheduling*, vol. 12, no. 2, pp. 199–224, 2009.

[38] S. Soudan, B. B. Chen, and P. Vicat-Blanc Primet, "Flow scheduling and endpoint rate control in gridnetworks," *Future Gener. Comput. Syst.*, vol. 25, no. 8, pp. 904–911, 2009.

[39] E. G. C. Jr., M. R. Garey, D. S. Johnson, and A. S. LaPaugh, "Scheduling file transfers," *SIAM J. Comput.*, vol. 14, no. 4, pp. 743–780, 1985.

[40] F. Manea and C. Ploscaru, "Solving a combinatorial problem with network flows," *Journal of Applied Mathematics and Computing*, 2004.

[41] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," *J. ACM*, vol. 37, no. 2, pp. 318–334, 1990.

[42] K. Rajah, S. Ranka, and Y. Xia, "Scheduling bulk file transfers with start and end times," *Comput. Netw.*, vol. 52, no. 5, pp. 1105–1122, 2008.