Advance Reservation and Scheduling for Bulk Transfers in Research Networks

Kannan Rajah, Sanjay Ranka and Ye Xia

Abstract

Data-intensive e-science collaborations often require the transfer of large files with predictable performance. To meet this need, we design novel admission control and scheduling algorithms for bulk data transfer in research networks for e-science. Due to their small sizes, the research networks can afford a centralized resource management platform. In our design, each bulk transfer job request, which can be made in advance to the central network controller, specifies a start time and an end time. If admitted, the network guarantees to complete the transfer before the end time. However, there is flexibility in how the actual transfer is carried out, that is, in the bandwidth assignment on each allowed paths of the job on each time interval, and it is up to the scheduling algorithm to decide this. To improve the network resource utilization or lower the job rejection ratio, the network controller solves optimization problems in making admission control and scheduling decisions. Our design combines the following elements into a cohesive optimization-based framework: advance reservation, multi-path routing, and bandwidth reassignment via periodic re-optimization. We evaluate our algorithm in terms of both network efficiency and the performance level of individual transfer. We also evaluate the feasibility of our scheme by studying the algorithm execution time.

I. Introduction

The advance of communication and networking technologies, together with the computing and storage technologies, is dramatically changing the ways how scientific research is conducted. A new term, *e-science*, has emerged to describe the "large-scale science carried out through distributed global collaborations enabled by networks, requiring access to very large scale data collections, computing resources, and high-performance visualization" [1]. Well-quoted e-science (and the related grid computing [2]) examples include high-energy nuclear physics (HEP), radio astronomy, geoscience and climate studies.

The need for transporting large volume of data in e-science has been well-argued [3], [4]. For instance, the HEP data is expected to grow from the current petabytes (PB) (10^{15}) to exabytes (10^{18}) by 2012 to 2015. In particular, the Large Hadron Collider facility at CERN is expected to generate petabytes of experimental data every year,

The authors are with the Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL.

Ye Xia is the corresponding author. Email: yx1@cise.ufl.edu, Phone: 352-392-2714, Fax:352-392-2714

This work was supported in part by the National Science Foundation (NSF) under Grant ITR 0325459 and 0427110. Any findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF. The authors would like to thank Rick Cavanaugh and Paul Avery for several discussions and insights.

for each experiment. In addition to the large volume, as noted in [5], e-scientists routinely request schedulable high-bandwidth low-latency connectivity with known and knowable characteristics. Instead of relying on the public Internet, which has unpredictable service performance, national governments are sponsoring a new generation of optical networks to support e-science. Examples of such research and education networks include the Internet2 related National Lambda Rail [6] and Abilene [7] networks in the U.S., and CA*net4 [8] in Canada.

To meet the need of e-science, this paper studies *admission control* (AC) and *scheduling* algorithms for high-bandwidth data transfers (also known as jobs) in research networks. The results will not only advance the knowledge and techniques in that area, but also compliment the protocol, architecture and infrastructure projects currently underway in support of e-science and grid computing [9], [10], [11], by providing more efficient network resource reservation and management algorithms. Our AC and scheduling algorithms handle two classes of jobs, *bulk data transfer* and those that require a *minimum bandwidth guarantee* (MBG). Bulk transfer is not sensitive to the network delay but may be sensitive to the delivery deadline. It is useful for distributing high volumes of scientific data, which currently often relies on ground transportation of the storage media. The MBG class is useful for realtime rendering or visualization of data remotely. In our framework, the algorithms for handling bulk transfer also contain the main ingredients of those for handling the MBG class. For this reason, we will only focus on bulk transfer.

One distinguishing feature in this study is that each job request can be made in advance and can specify a start time and an end time. The reservation-based approach gives the network users more predictability and control over their work schedule and is deemed very useful by the e-science community [12]. If a job is admitted, as determined by the admission control algorithm, the network guarantees that it will finish the data transfer for the job before the requested end time. The challenge is how to provide this guarantee while maintaining efficient utilization of the network resources and keeping the request rejection ratio low. (If a request is rejected, there are many possible follow-up scenarios depending on the design. The simplest is that the user of the request may modify the end time and re-submit the request. The re-submission process can be automated and repeated by the user-side software agent.)

The need for efficient network resource utilization is especially relevant in the context of advance reservation and large file sizes or long-lasting flows. As argued in [13], there is an undesirable phenomenon known as *bandwidth fragmentation*. The simplest example of bandwidth fragmentation occurs when the interval between the end time of one job and the beginning of another job is not long enough for any other job request. Then, the network or relevant links will be idle on that interval. If there are too many of these unusable intervals or if their durations are long, the job rejection ratio is likely to be high while the network utilization remains low. Over-provisioning the network capacity may not be the right solution due to the high cost, time delay or other practical constraints.

The solution advocated in this paper for reducing the job rejection ratio and increasing the network utilization efficiency is to bring in more flexibilities in how the data are transferred. The process of determining the manner of data transfer is known as *scheduling*. For instance, one can take advantage of the elastic nature of bulk data and have the network transferring the data at time-varying bandwidth instead of a constant bandwidth. Another example is to use multiple paths for each job. In order to achieve the greatest flexibilities, this paper formulates the AC/scheduling problems as optimization problems. A centralized network controller is used to administer AC and scheduling, including solving the optimization problems. Different from the public Internet, research networks typically have far less than 1000 core nodes in the backbone. Hence, it possible to use a centralized network controller for making AC and scheduling decisions, setting up network paths, and reserving the allocated bandwidth or optical circuits. One advantage of the centralized approach is that resource reservation and allocation decisions are made based on a global view of the network and on all the job requests. It is possible to manage the network resources as a whole and make trade-offs among all the jobs in the network. The result is greatly improved efficiency in network resource utilization.

Recently, some authors have begun to study AC and scheduling for bulk transfer with advance reservations [14], [15], [16], [17], [18], [19], [13], [20], [21]. Compared with these earlier studies, our work distinguishes itself for its comprehensiveness in bringing several important ingredients together under a single optimization framework with well-defined objectives. These include (1) periodic admission control for handling continuous arrivals of job requests rather than one-shot admission control, (2) admission control and scheduling for the whole network rather than for each link separately, (3) multi-path routing, (4) time-varying bandwidth assignment for each job, (5) dynamic bandwidth re-assignment at each AC/scheduling instance, which leaves more room to accept new requests, and (6) a novel time discretization scheme (i.e., the congruent time-slice structures) that allows the admission of new requests and bandwidth re-allocation to existing jobs while not violating the end-time requirements of the existing jobs. As will be reviewed in Section V, other studies in this area only incorporate a subset of the features from the above list.

The rest of the paper is organized as follows. The main technical contribution of this paper is to describe a suite of algorithms for AC and scheduling (Section II) and compare their performance (Section IV). A key methodology is the discretization of time into a time slice structure so that the problems can be put into the linear programming framework. A highlight of our scheme is the introduction of non-uniform time slices (Section III), which can dramatically shorten the execution time of the AC and scheduling algorithms, making them practical. The related work is shown in Section V and the conclusion is drawn in Section VI.

II. Admission Control and Scheduling Algorithms

A. The Setup

For easy reference, notations and definitions frequently used in this paper are summarized in Appendix I. The network is represented as a (directed) graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. The capacity of a link (edge) $e \in E$ is denoted by C_e. Job requests arrive at the network following a random process. Each bulk transfer request i is a 6-tuple $(A_i, s_i, d_i, D_i, S_i, E_i)$, where A_i is the arrival time of the request, s_i and d_i are the source and destination nodes, respectively, D_i is the size of the file, S_i and E_i are the requested start time and end time, where $A_i \leq S_i \leq E_i$. In words, request i, which is made at time $t = A_i$, asks the network to transfer a file of size D_i from node s_i to node d_i on the time interval $[S_i, E_i]$. A bulk transfer request may optionally specify a minimum bandwidth and/or a maximum bandwidth. In practice, even more parameters can be added if needed, such as an estimated range for the demand size or for the end times when the precise information is unknown [22]. For ease of presentation, we will ignore these options. But, they usually can be incorporated into our optimization-based AC/scheduling framework by modifying the formulations of the optimization problems. The approach of using a centralized network controller has an advantage here for an evolving system, since, to accommodate new types of parameters or functions, the only necessary changes are at the central controller's software. The user-side software will be updated only if the user needs the new parameters or functions.

In the basic scheme, AC and scheduling are done periodically after every τ time units, where τ is a positive number. More specifically, at time instances $k\tau$, $k = 1, 2, ...$, the controller collects all the new requests that arrived on the interval $[(k-1)\tau, k\tau]$, makes the admission control decision first, and then, schedules the transfer of all jobs. Both AC and scheduling must take into account the *old jobs*, i.e., those jobs that were admitted earlier but remain unfinished. The admission of new jobs is formulated as a feasibility problem subject to the constraint that the old jobs must retain their performance guarantee. To increase the admission rate, this step takes into account the possibility that the bandwidth of each old job on different routes can be reassigned. In the second step, scheduling, the network controller assigns the actual bandwidth to all jobs in the system, including the old jobs, on the allowed paths so as to optimize a performance objective. Examples that we consider in this paper are to minimize the worst case link utilization or to minimize an objective that encourages earlier completion of the jobs. The bandwidth assignment is time-varying. The value of τ should be small enough so that new job requests can be checked for admission and scheduled as early as possible[1]. However, τ should be more than the computation time required for AC and scheduling.

1) The Time Slice Structure: At each scheduling instance, $t = k\tau$, the timeline from t onward is partitioned into time slices, i.e., closed intervals on the timeline, which are not necessarily uniform in size. The significance of the time slice is that the bandwidth (rate) assignment to each job is done at the slice level. That is, the bandwidth assigned to a particular path of a job remains constant for the entire time slice, but it may change from slice to slice.

A set of time slices, $\mathcal{G}_k$, is said to be *anchored at* $t = k\tau$ if all slices in $\mathcal{G}_k$ are mutually disjoint and their union forms an interval $[t, t']$ for some t'. The set $\{\mathcal{G}_k\}_{k=1}^{\infty}$ is called a *slice structure* if each $\mathcal{G}_k$ is a set of slices anchored at $t = k\tau$, for $k = 1, ..., \infty$.

Definition 1: A slice structure $\{\mathcal{G}_k\}_{k=1}^{\infty}$ is said to be **congruent** if the following property is satisfied for every pair of positive integers, k and k', where $k' > k \geq 1$. For any slice $s' \in \mathcal{G}_{k'}$, if s' overlaps in time with a slice s, $s \in \mathcal{G}_k$, then $s' \subseteq s$.

In words, any slice in a later anchored slice collection must be completely contained in a slice of any earlier collection, if it overlaps in time with the earlier collection. Alternatively speaking, if slice $s \in \mathcal{G}_k$ overlaps in time with $\mathcal{G}_{k'}$, then either $s \in \mathcal{G}_{k'}$ or s is partitioned into multiple slices all belonging to $\mathcal{G}_{k'}$.

One example of a congruent slice structure is the *uniform slices (US)*, where the timeline is divided into equal-

[1]In this scheme, a request generally needs to wait a duration no longer than τ for the admission decision. We will comment on how to conduct realtime admission control later.

sized time slices of duration τ (coinciding with the AC/scheduling interval length). The set of slices anchored at any $t = k\tau$ is all the slices after t. Figure 1 shows the US at two time instances $t = \tau$ and $t = 2\tau$. In this example, $\tau = 4$ time units. The arrows point to the scheduling instances. The two collections of rectangles are the time slices anchored at $t = \tau$ and $t = 2\tau$, respectively. It is easy to check the congruent property of this slice structure.

Nearly all prior works that discretize the timeline use the US. The motivation for defining the more general concept of the congruent slice structure is as follows. Although easy to understand, the US is not necessarily an ideal slice structure to use because, in our linear programming formulation of the AC and scheduling problems, the number of time slices is positively related to the number of variables, and in turn to the execution time of our algorithms. We face a problem of covering a long enough segment of the timeline for advance reservations with a small number of slices, say 100. As an example, to cover a 30-day reservation period with 100 slices, the slice size in the US is 7.2 hours, too coarse for small to medium sized jobs whose requested time windows for data transfer are well under one hour. In this paper, we advocate a congruent slice structure with non-uniform slice sizes, the *nested slices (NS)*. The NS contains different classes of time slices with exponentially (geometrically) increasing sizes. Suppose the current time $t = k\tau$ is a scheduling instance. The timeline near t is divided into fine slices. The timeline away from t is divided into increasingly larger slices. Later, as time progresses, say to $k'\tau$, some coarse time slices will become close to the new current time, $k'\tau$, and will be divided into fine slices, which will belong to $\mathcal{G}_{k'}$. As will be demonstrated later, the NS can cover a large portion of the timeline using a small number of slices without sacrificing the performance (e.g., job rejection ratio). Fig. 5 shows a three-level nested slice structure. More detailed description about the NS is deferred to Section III.

The AC and scheduling algorithms introduced in this paper apply to any congruent slice structure. When a non-uniform slice structure is used, the congruent property is the key to the existence of algorithms that allow the network to keep the commitment to the old jobs admitted earlier while admitting new jobs. The reason is that, in solving the admission control problem, the bandwidth allocation (on each allowed path of each job) on each time slice is assumed to be constant. When a time slice is divided into finer slices at a later time, the old jobs are still admissible since one can keep the bandwidth on the finer slices at the same constant.[2] This will be further explained in Section III. For ease of presentation, we use the uniform slices as an example to explain the AC and scheduling algorithms.

At any AC/scheduling time $t = k\tau$, let the time slices anchored at t, i.e., those in $\mathcal{G}_k$, be indexed $1, 2, \ldots$ in increasing order of time. Let the start and end times of slice i be denoted by $ST_k(i)$ and $ET_k(i)$, respectively, and let its length be $LEN_k(i)$. We say a time instance $t' > t$ *falls into* slice i if $ST_k(i) < t' \leq ET_k(i)$. The index of the slice that t' falls in is denoted by $I_k(t')$.

At $t = k\tau$, let the set of jobs in the system yet to be completed be denoted by $\mathcal{J}_k$. $\mathcal{J}_k$ contains two types of jobs, those new requests (also known as new jobs) made on the interval $((k-1)\tau, k\tau]$, denoted by $\mathcal{J}_k^n$, and those old jobs admitted at or before $(k-1)\tau$, denoted by $\mathcal{J}_k^o$. The old jobs have already been admitted and should not

[2]However, one can often do better by varying the bandwidth on the finer slices.

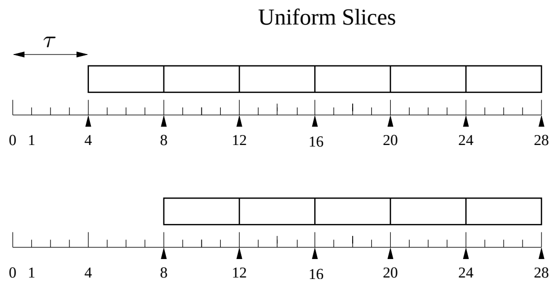


Fig. 1. Uniform time slice structure

be rejected by the admission control conducted at t. But some of the new requests may be rejected.

2) Rounding of the Start and End Times: With the time slice structure and the advancement of time, we adjust the start and end times of the requests. The main objective is to align the start and end times on the slice boundaries. After such rounding, the start and the end times will be denoted as $\hat{S}_i$ and $\hat{E}_i$, respectively. For a new request i, let the requested response time be $T_i = E_i - S_i$. We round the requested start time to be the maximum of the current time or the end time of the slice in which the requested start time S_i falls, i.e.,

$$\hat{S}_i = \max\{t, ET_k(I_k(S_i))\}. \tag{1}$$

For rounding of the requested end time, we allow two policy choices, the *stringent policy* and the *relaxed policy*. Which one is used in practice is a policy issue, left to the decision of the network manager. In the stringent policy, if the requested end time does not coincide with a slice boundary, it is rounded down, subject to the constraint that $\hat{E}_i > \hat{S}_i$ [3]. This constraint ensures that there is at least one-slice separation between the rounded start time and the rounded end time. Otherwise, there is no way to schedule the job. In the relaxed policy, the end time is first shifted by T_i with respect to the rounded start time, and then rounded up. More specifically,

stringent

$$\hat{E}_i = \begin{cases} ET_k(I_k(\hat{S}_i) + 1) & \text{if } ST_k(I_k(E_i)) \leq \hat{S}_i \\ E_i & \text{else if } ET_k(I_k(E_i)) = E_i \\ ST_k(I_k(E_i)) & \text{otherwise.} \end{cases} \tag{2}$$

relaxed

$$\hat{E}_i = ET_k(I_k(\hat{S}_i + T_i))$$

Figure 2 shows the effect of the two policies after three jobs are rounded.

[3]In the more sophisticated non-uniform slice structure introduced in Section III, we allow the end time to be re-rounded at different scheduling instances. This way, the rounded end time can become closer to the requested end time, as the slice sizes become finer over time.

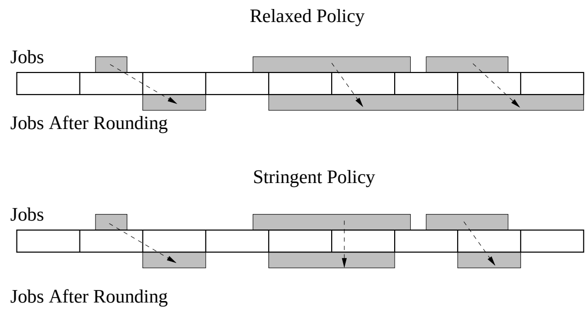


Fig. 2. Two rounding policies. The unshaded rectangles are time slices. The shaded rectangles represent jobs. The top ones show the requested start and end times. The bottom ones show the rounded start and end times.

If a job i is an old one, its rounded start time $\hat{S}_i$ is replaced by the current time t. The remaining demand is updated by subtracting from it the total amount of data transferred for job i on the previous interval, $((k-1)\tau, k\tau]$.

By definition, the slice set anchored at each $t = k\tau$, $\mathcal{G}_k$, contains an infinite number of slices. In general, only a finite subset of $\mathcal{G}_k$ is useful to us. Let M_k be the index of the last slice in which the rounded end time of some jobs falls. That is, $M_k = I_k(\max_{i \in \mathcal{J}_k} \hat{E}_i)$. Let $\mathcal{L}_k \subset \mathcal{G}_k$ be the collection of time slices $1, 2, ..., M_k$. We call the slices in $\mathcal{L}_k$ as the *active time slices*. We will also think of $\mathcal{L}_k$ as an array (instead of a set) of slices when there is no ambiguity. Clearly, the collection $\{\mathcal{L}_k\}_{k=1}^{\infty}$ inherits the congruent property from $\{\mathcal{G}_k\}_{k=1}^{\infty}$. Therefore, it is sufficient to consider $\{\mathcal{L}_k\}_{k=1}^{\infty}$ for AC and scheduling.

B. Admission Control

For each pair of nodes s and d, let the collection of allowable paths from s to d be denoted by $P_k(s, d)$. In general, the set may vary with k. For each job i, let the *remaining demand* at time $t = k\tau$ be denoted by $R_k(i)$, which is equal to the total demand D_i minus the amount of data transferred until time t.

At $t = k\tau$, let $J \subseteq \mathcal{J}_k$ be a subset of the jobs in the systems. Let $f_i(p, j)$ be the total flow (total data transfer) allocated to job i on path p, where $p \in P_k(s_i, d_i)$, on time slice j, where $j \in \mathcal{L}_k$. As part of the admission control algorithm, the solution to the following feasibility problem is used to determine whether the jobs in J can all be admitted.

AC(k, J)

$$\sum_{j=1}^{M_k} \sum_{p \in P_k(s_i,d_i)} f_i(p,j) = R_k(i), \quad \forall i \in J \tag{3}$$

$$\sum_{i \in J} \sum_{\substack{p \in P_k(s_i,d_i) \\ p: e \in p}} f_i(p,j) \leq C_e(j) LEN_k(j), \quad \forall e \in E, \forall j \in \mathcal{L}_k \tag{4}$$

$$f_i(p,j) = 0, \quad j \leq I_k(\hat{S}_i) \text{ or } j > I_k(\hat{E}_i), \quad \forall i \in J, \forall p \in P_k(s_i,d_i) \tag{5}$$

$$f_i(p,j) \geq 0, \quad \forall i \in J, \forall j \in \mathcal{L}_k, \forall p \in P_k(s_i,d_i). \tag{6}$$

(3) says that, for every job, the sum of all the flows assigned on all time slices for all paths must be equal to its remaining demand. (4) says that the capacity constraints must be satisfied for all edges on every time slice. Note that the allocated rate on path p for job i on slice j is $f_i(p,j)/LEN_k(j)$, where $LEN_k(j)$ is the length of slice j. The rate is assumed to be constant on the entire slice. Here, $C_e(j)$ is the remaining link capacity of link e on slice j. (5) is the start and end time constraint for every job on every path. The flow must be zero before the rounded start time and after the rounded end time. [4]

Recall that we are assuming every job to be a bulk transfer for simplicity. If job i is of the MBG class and requests a minimum bandwidth B_i between the start and end times, then the remaining capacity constraint (3) will be replaced by the following minimum bandwidth guarantee condition.

$$\sum_{p \in P_k(s_i,d_i)} f_i(p,j) \geq B_i, \quad \forall j \in \mathcal{L}_k. \tag{7}$$

The AC/scheduling algorithms are triggered every τ time units with the AC part before the scheduling part. AC examines the newly arrived jobs and determines their admissibility. In doing so, we need to ensure that the earlier commitments to the old jobs are not broken. This can be achieved by adopting one of the following AC procedures.

1) *Subtract-Resource (SR)*: An updated (remaining) network is obtained by subtracting the bandwidth assigned to old jobs on future time slices, from the link capacity. Then, we determine a subset of the new jobs that can be accommodated in this remaining network. This method is helpful to perform quick admission tests [5]. However, it runs the risk of rejecting new jobs that can actually be accommodated by reassigning the flows to the old jobs on different paths and time slices.

[4]The current research networks generally use routers over optical transmission technologies instead of using optical switches alone. Routers can split or aggregate traffic before transmission. Hence, the problem in this paper is fine-grained bandwidth assignment rather than wavelength assignment, as would be the case in a wavelength-based circuit-switched optical network. It is possible to reserve an end-to-end wavelength path in the current research networks. But, our formulation of the bandwidth assignment problem will be unaffected since we can simply remove the reserved wavelength from the link capacity. We defer the wavelength assignment problem in an all optical network to future research.

[5]We can perform realtime admission with this method.

2) *Reassign-Resource (RR)*: This method attempts to reassign flows to the old jobs. First, we cancel the existing flow assignment to the old jobs on the future time slices and restore the network to its original capacity. Then, we determine a subset of the new jobs that can be admitted along with all the old jobs under the original network capacity. This method is expected to have a better acceptance ratio than SR. However, it is computationally more expensive because the flow assignment is computed for all the jobs in the system, both the old and the new.

The actual admission control is as follows. In the SR scheme, the remaining capacity of link e on slice j, $C_e(j)$, is computed by subtracting from C_e (the original link capacity), the total bandwidth allocated on slice j for all paths crossing e, during the previous run of the AC/scheduling algorithms (at $t = (k-1)\tau$). In the RR scheme, simply let $C_e(j) = C_e$, for all e and j.

In the SR scheme, we list the *new* jobs, $\mathcal{J}_k^n$, in a sequence, $1, 2, ..., m$. The particular order of the sequence is flexible, possibly dependent on some customizable policy. For instance, the order may be arbitrary, or based on the priority the jobs, or based on increasing order of the request times. In a more sophisticated, price-based scheme, the network controller can order the jobs based on the amount of payment per unit of data transferred that a job requester is willing to pay. We apply a binary search to the sequence to find the last job j, $1 \leq j \leq m$, in the sequence such that all jobs before and including it are admissible. That is, j is the largest index for which the subset of the new jobs $J = \{1, 2, ..., j\}$ is feasible for **AC(**k**,** J**)**. All the jobs after j are rejected.

In the RR scheme, at time $t = k\tau$, *all* the jobs are listed in a sequence where the old jobs $\mathcal{J}_k^o$ are ahead of the new jobs $\mathcal{J}_k^n$ in the sequence. The order among the old jobs is arbitrary. The order among the new jobs is again flexible. Denote this sequence as $1, 2, ..., m$, in which jobs 1 through l are the old ones. We then apply a binary search to the sequence of *new* jobs, $l+1, l+2, ..., m$, to find the last job j, $l < j \leq m$, such that all jobs before and including it are admissible. That is, j is the largest index for which the resulting subset of the jobs $J = \{1, 2, ..., l, l+1, ..., j\}$ is feasible for **AC(**k**,** J**)** under the original network capacity.

Discussion The binary search technique assumes a pre-defined list of jobs and identifies the first j jobs that can be admitted into the system without violating the deadline constraints. The presence of an exceptionally large job with unsatisfiable demand will cause other jobs following it to be rejected, even though it may be possible to accommodate them after removing the large job. The rejection ratio tends to be higher when the large job lies closer to the head of the list. An interesting problem is how to admit as many new jobs as possible, after all the old jobs are admitted. This combinatorial problem appears to be quite difficult. One can always use a standard integer programming formulation and solution for it. We do not know any solution techniques that run faster than the integer programming techniques. But, a solution to this problem is orthogonal to the main issues addressed in this paper and, once found, can always be incorporated into our general AC/scheduling framework.

We now comment on the computation complexity for the admission control, **AC(**k**,** J**)**. If standard linear programming techniques are used, such as the Simplex method, the practical computation time depends on the number of variables and the number of constraints. In **AC(**k**,** J**)**, the number of variables is no more than $|J| \times M \times P$. Here, P is the maximum number of paths allowed for any job. M is the maximum number of (future) time slices that need to be considered. It depends on how far into the future advance reservations can be allowed, e.g., three months, and on the type of the congruent slice structure used. The value of $|J|$ depends on whether SR or RR is used. In the former case, it is equal to the number of new job requests that have arrived on an interval of length τ; in the latter case, it is equal to all the jobs in the system, including both the old jobs and the new requests. The number of non-trivial constraints is no more than $|J| + |E| \times M$, where $|E|$ is the number of edges in the network. To reduce the execution time of the admission control algorithm, we need to limit the number of paths allowed per job, the number of time slices and the number of jobs that need to be considered. In Section IV-C, we show by experimental results that having 4 to 10 paths per job is generally sufficient to achieve near-optimal performance for research networks. If ever needed, SR is a way of reducing the number of jobs that need to be considered. What remains is how to reduce the number of time slices while not sacrificing performance by much. Section III is dedicated to that purpose. Section IV will continue to address the complexity issue in terms of the algorithm execution time obtained experimentally.

C. Scheduling Algorithm

Given the set of admitted jobs, $\mathcal{J}_k^a$, which always includes the old jobs, the scheduling algorithm allocates flows to these jobs to optimize a certain objective. We consider two objectives, **Quick-Finish** (QF) and **Load-Balancing** (LB). Given a set of admissible jobs J, the problem associated with the former is

Quick-Finish(k, J)

$$\min \quad \sum_{j \in \mathcal{L}_k} \gamma(j) \sum_{i \in J} \sum_{p \in P_k(s_i, d_i)} f_i(p, j) \tag{8}$$

subject to $(3) - (6)$.

In the above, $\gamma(j)$ is a weight function increasing in j, which is chosen to be $\gamma(j) = j + 1$ in our experiments. In this problem, the cost increases as time increases. The intention is to finish a job early rather than later, when it is possible. The solution tends to pack more flows in the earlier slices but leaves the load light in later slices. The problem associated with the LB objective is,

Load-Balancing(k, J)

$$\max \quad Z \tag{9}$$

$$\text{subject to} \sum_{j=1}^{M_k} \sum_{p \in P_k(s_i, d_i)} f_i(p, j) = Z R_k(i), \quad \forall i \in J \tag{10}$$

$$(4) - (6).$$

Let the optimal solution be Z^* and $f_i^*(p, j)$ for all i, j, and p. The actual flows assigned are $f_i^*(p, j)/Z^*$. Note that (10) ensures that $f_i^*(p, j)/Z^*$ satisfies (3). Also, $Z^* \geq 1$ must be true since J is admissible. Hence, $f_i^*(p, j)/Z^*$'s are a feasible solution to the **AC(k, J)** problem. The **Load-Balancing(k, J)** problem above is written in the maximizing concurrent throughput form. It reveals its load-balancing nature when written in the equivalent minimizing congestion form. For that, make a substitution of variables, $f_i(p, j) \Leftarrow f_i(p, j)/Z$, and let $\mu = 1/Z$.

We have,

$$
\begin{aligned}
&\textbf{Load-Balancing-1}(k,\ J) \\
&\min \quad \mu \qquad\qquad (11) \\
&\text{subject to} \sum_{i \in J} \sum_{\substack{p \in P_k(s_i, d_i) \\ p: e \in p}} f_i(p, j) \leq \mu C_e(j) LEN_k(j), \\
&\qquad\qquad\qquad \forall e \in E, \forall j \in \mathcal{L}_k \qquad\qquad (12) \\
&\qquad (3), (5) \text{ and } (6).
\end{aligned}
$$

Hence, the solution minimizes the worst link congestion across all time slices in $\mathcal{L}_k$.

The scheduling algorithm is to apply $J = \mathcal{J}_k^a$ to **Quick-Finish(**k**,** J**)** or **Load-Balancing(**k**,** J**)**. This determines an optimal flow assignment to all jobs on all allowed paths and on all time slices. Given the flow assignment $f_i(p, j)$, the allocated rate on each time slice is denoted by $x_i(p, j) = f_i(p, j)/LEN_k(j)$ for all $j \in \mathcal{L}_k$. The remaining capacity of each link on each time slice is given by,

$$
C_e(j) = \begin{cases} C_e - \sum_{i \in \mathcal{J}_k^a} \sum_{\substack{p \in P_k(s_i, d_i) \\ p: e \in p}} x_i(p, j) & \text{if SR} \\ C_e & \text{if RR.} \end{cases} \qquad (13)
$$

Finally, the complexity of the scheduling algorithms can be analyzed similarly as for the admission control algorithm. The general conclusion is also similar.

D. Putting It Together: The AC and Scheduling Algorithms

In this section, we integrate various algorithmic components and present the complete AC and scheduling algorithms.

On the interval $((k-1)\tau, k\tau]$, the system keeps track of the new requests arriving on that interval. It also keeps track of the status of the old jobs. If an old job is completed, it is removed from the system. If an old job is serviced on the interval, the amount of data transferred for that job is recorded. At $t = k\tau$, the steps described in Algorithm 1 are taken.

Finally, in Figure 3, we show a very simple example of the AC and scheduling algorithms at work. The network has only one link with a capacity of 10 Gbps. The US is used and the AC/scheduling interval length is $\tau = 100s$. QF is used for scheduling. The top figure shows the job requests. The sizes of job 1 and 2 are 3 terabits and 500 gigabits, respectively. The requested start and end times are $100s$ and $700s$ for job 1; and $200s$ and $300s$ for job 2. In this case, job 1 is admitted at $t = 100s$. The middle figure shows the schedule at $t = 100s$. At $t = 200s$, job 2 is also admitted. The bottom figure shows the schedule at $t = 200s$. Note that, by $t = 200s$, 1 terabits of data have already been transferred for job 1. Note also how the bandwidth assignment for job 1 is changed $t = 200s$,

Algorithm 1 Admission Control and Scheduling

1: Construct the anchored slice set at $t = k\tau$, $\mathcal{G}_k$.

2: Construct the job sets $\mathcal{J}_k$, $\mathcal{J}_k^o$ and $\mathcal{J}_k^n$, which are the collection of all jobs, the collection of old jobs, and the collection of new jobs in the system, respectively.

3: For each old job i, update the remaining demand $R_k(i)$ by subtracting from it the amount of data transferred for i on the interval $((k-1)\tau, k\tau]$. Round the start times as $\hat{S}_i = t$.

4: For each new job l, let $R_k(l) = D_l$. Round the requested start and end time according to (1) and (2), depending on whether the stringent or relaxed rounding policy is used. This produces the rounded start and end times, $\hat{S}_l$ and $\hat{E}_l$.

5: Derive $M_k = I_k(\max_{i \in \mathcal{J}_k} \hat{E}_i)$. This determines the finite collection of slices $\mathcal{L}_k = \{1, 2, ..., M_k\}$, the first M_k slices of $\mathcal{G}_k$.

6: Perform admission control as in Algorithm 2. This produces the list of admitted jobs $\mathcal{J}_k^a$.

7: Schedule the admitted jobs as in Algorithm 3. This yields the flow amount $f_i(p, j)$ for each admitted job $i \in \mathcal{J}_k^a$, over all paths for job i, and all time slices $j \in \mathcal{L}_k$.

8: Compute the remaining network capacity by (13).

Algorithm 2 AC - Step 6 of Algorithm 1

1: **if** Subtract-Resource is used **then**

2:     Sequence the *new* jobs ($\mathcal{J}_k^n$) in the system. Denote the sequence by $(1, 2, ..., m)$.

3:     Find the last job j in the sequence so that the set of jobs $J = \{1, 2, ..., j\}$ is admissible by **AC(**k**,** J**)**.

4: **else if** Reassign-Resource is used **then**

5:     Sequence *all* the jobs ($\mathcal{J}_k$) in the system, so that the old jobs ($\mathcal{J}_k^o$) are ahead of the new jobs ($\mathcal{J}_k^n$). Denote the sequence of jobs by $(1, 2, ..., l, l+1, ..., m)$, where the first l jobs are the old jobs, followed by the new jobs.

6:     Apply binary search to the subsequence of new jobs $(l+1, l+2, ..., m)$. Find the last job j in the subsequence so that the set of jobs $J = \{1, 2, ..., j\}$ is admissible by **AC(**k**,** J**)**.

7: **end if**

8: Return the admissible set, $\mathcal{J}_k^a = J$.

Algorithm 3 Scheduling - Step 7 of Algorithm 1

1: **if** Quick-Finish is preferred **then**

2:     Solve **Quick-Finish(**k**,** $\mathcal{J}_k^a$**)**

3: **else**

4:     Solve **Load-Balancing(**k**,** $\mathcal{J}_k^a$**)**

5: **end if**

when compared to that at $t = 100s$. This is in response to the admission of job 2, which has a stringent end time requirement. Furthermore, it can be seen that the bandwidth assignment for job 1 is time-varying.

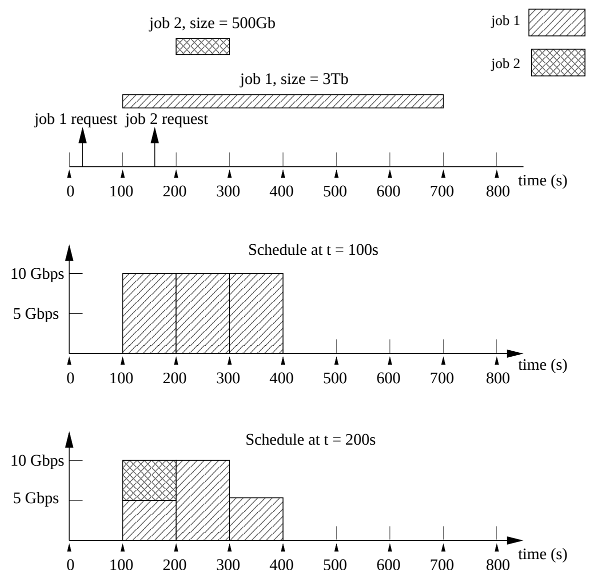


Fig. 3. An AC and scheduling example for a network with one link with a capacity 10 Gbps.

III. Non-Uniform Slice Structure

As discussed in Section II-A.1 and Section II-B, the number of time slices directly affects the number of variables in our AC and scheduling linear programs, and in turn the execution time of our algorithms. We face a problem of covering a large enough segment of the timeline for advance reservations with a small number of slices, say about 100. In this section, we will design a new slice structure with non-uniform slice sizes. They contain a geometrically (exponentially) increasing subsequence, and therefore, are able to cover a large timeline with a small number of slices. The key is that, as time progresses, coarse time slices will be further divided into finer slices. The challenge is that the slice structure must remain congruent.

Recall that the congruent property means that, if a slice in an earlier anchored slice set overlaps in time with a later anchored slice set, it either remains as a slice, or is partitioned into smaller slices in the later slice set. The definition is motivated by the need for maintaining consistency in bandwidth assignment across time. As an example, suppose at time $(k-1)\tau$, a job is assigned a bandwidth x on a path on the slice j_{k-1}. At the next scheduling instance $t = k\tau$, suppose the slice j_{k-1} is partitioned into two slices. Then, we understand that a bandwidth x has been assigned on both slices. Without the congruent property, it is likely that a slice, say j_k, in the slice set anchored at $k\tau$ cuts across several slices in the slice set anchored at $(k-1)\tau$. If the bandwidth assignments at $(k-1)\tau$ are different for these latter slices, the bandwidth assignment for slice j_k is not well defined just before the AC/scheduling run at time $k\tau$.

A. Nested Slice Structure

In the nested slice structure, there are l types of slices, known as level-i slices, $i = 1, 2, ..., l$. Each level-i slice has a duration Δ_i, with the property that $\Delta_i = \kappa_i \Delta_{i+1}$, where $\kappa_i > 1$ is an integer, for $i = 1, ..., l-1$. Hence, the slice size increases at least geometrically as i decreases. For practical applications, a small number of levels suffices. We also require that, for i such that $\Delta_{i+1} \leq \tau < \Delta_i$, τ is an integer multiple of Δ_{i+1} and Δ_i is an integer multiple of τ. This ensures that each scheduling interval contains an integral number of slices and that the sequence of scheduling instances does not skip any level-j slice boundaries, for $1 \leq j \leq i$.

The nested slice structure can be defined by construction. At $t = 0$, the timeline is partitioned into level-1 slices. The first j_1 level-1 slices, where $j_1 \geq 1$, are each partitioned into level-2 slices. This removes j_1 level-1 slices but adds $j_1\kappa_1$ level-2 slices. Next, the first j_2 level-2 slices, where $j_2 \leq j_1\kappa_1$, are each partitioned into level-3 slices. This removes j_2 level-2 slices but adds $j_2\kappa_2$ level-3 slices. This process continues until, in the last step, the first j_{l-1} level-$(l-1)$ slices are partitioned into $j_{l-1}\kappa_{l-1}$ level-l slices. Then, the first j_{l-1} level-$(l-1)$ slices are removed and $j_{l-1}\kappa_{l-1}$ level-l slices are added at the beginning. In the end, the collection of slices at $t = 0$ contains $\sigma_l \triangleq j_{l-1}\kappa_{l-1}$ ($\triangleq$ means "defined as") level-l slices, $\sigma_{l-1} \triangleq j_{l-2}\kappa_{l-2} - j_{l-1}$ level-$(l-1)$ slices, ..., $\sigma_2 \triangleq j_1\kappa_1 - j_2$ level-2 slices, and followed by an infinite number of level-1 slices. The sequence of j_i's must satisfy $j_2 \leq j_1\kappa_1$, $j_3 \leq j_2\kappa_2$, ..., $j_{l-1} \leq j_{l-2}\kappa_{l-2}$. This collection of slices is denoted by $\mathcal{G}_0$.

As an example, to cover a maximum of 30-day period, we can take $\Delta_1 = 1$ day, $\Delta_2 = 1$ hour, and $\Delta_3 = 10$ minutes. Hence, $\kappa_1 = 24$ and $\kappa_2 = 6$. The first two days are first divided into a total 48 one-hour slices, out of which the first 8 hours are further divided into 48 10-minute slices. The final slice structure has 48 level-3 (10-minute) slices, 40 level-2 (one-hour) slices, and as many level-1 (one-day) slices as needed, in this case, 28. The total number of slices is 116.

For the subsequent scheduling instances, the objective is to maintain the same number of slices at each level as in $\mathcal{G}_0$ (since it is what the system designer wants). But this cannot be done while satisfying the slice congruent property. Hence, we allow the number of slices at each level to deviate from σ_j, for $j = 2, ..., l$. This can be done in various ways. Let z_j be the current number of level-j slices at $t = k\tau$, for $j = 1, 2, ..., l$. Set $z_1 = \infty$.

1) **At-Least-σ:** For j from l down to 2, if the number of slices at level j, z_j, is less than σ_j, bring in (and remove) the next level-$(j-1)$ slice and partition it into κ_{j-1} level-j slices. This scheme maintains at least σ_j and at most $\sigma_j + \kappa_{j-1} - 1$ level-j slices for $j = 2, ..., l$.
2) **At-Most-σ:** In this scheme, we try to bring the current number of slices at level j, z_j, to σ_j, for $j = 2, ..., l$, subject to the constraint that new slices at level j can only be created if t is an integer multiple of Δ_{j-1}. More specifically, at $t = k\tau$, the following is repeated for j from l down to 2. If t is *not* an integer multiple of Δ_{j-1}, then nothing is done. Otherwise, if $z_j < \sigma_j$, we try to create level-j slices out of a level-$(j-1)$ slice. In the creation process, if a level-$(j-1)$ slice exists, then bring in the first one and partition it. Otherwise, we try to create more level-$(j-1)$ slices, provided t is an integer multiple of Δ_{j-2}. Hence, a recursive slice-creation process may be involved.

Fig. 4 and 5 show a two-level and three-level nested slice structure, respectively, under the At-Most-σ design. In the special but typical case of $\sigma_j > \kappa_{j-1}$, for $j = 2, ..., l$, the At-Most-σ algorithm can be simplified as follows. For j from l down to 2, if $z_j \leq \sigma_j - \kappa_{j-1}$, bring in (and remove) the next level-$(j-1)$ slice and partition it into κ_{j-1} level-j slices. This scheme maintains at least $\sigma_j - \kappa_{j-1}$ and at most σ_j level-j slices for $j = 2, ..., l$.

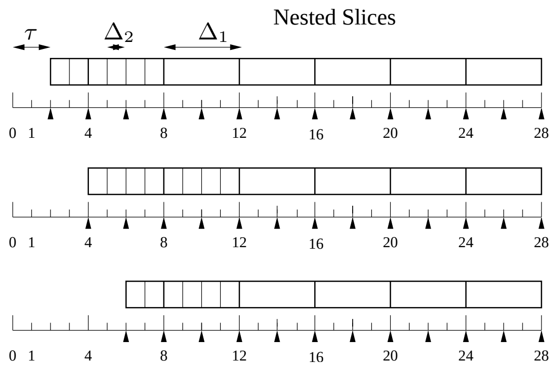


Fig. 4. Two-level nested time-slice structure. $\tau = 2$, $\Delta_1 = 4$ and $\Delta_2 = 1$. The anchored slice sets shown are for $t = \tau, 2\tau$ and 3τ, respectively. At-Most-σ Design. $\sigma_2 = 8$.

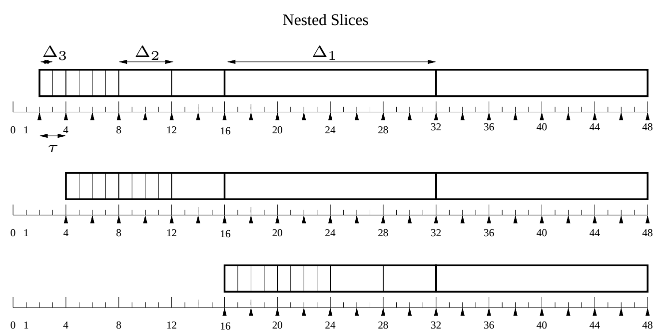


Fig. 5. Three-level nested time-slice structure. $\tau = 2$, $\Delta_1 = 16$, $\Delta_2 = 4$ and $\Delta_3 = 1$. The anchored slice sets shown are for $t = \tau, 2\tau$ and 8τ, respectively. At-Most-σ Design. $\sigma_3 = 8$, $\sigma_2 = 2$.

B. Variant of Nested Slice Structure

When some κ_j is large, it may be unappealing that the number of level-j slices varies by κ_{j-1} (sometimes more than κ_{j-1}). To solve this problem, we next introduce another congruence slice structure related to the nested slice structure. We will called it the **Almost-σ Variant** of the nested slice structure, because it maintains at least σ_j and at most $\sigma_j + 1$ level-j slices for $j = 2, ..., l$.

The Almost-σ Variant starts the same way as the nested slice structure at $t = 0$. As time progresses from $(k-1)\tau$ to $k\tau$, for $k = 1, 2, ...$, the collection of slices anchored at $t = k\tau$, i.e., $\mathcal{G}_k$, is updated from $\mathcal{G}_{k-1}$ as in algorithm 4.

The price to pay is that the Almost-σ Variant introduces new slice types different from the pre-defined level-i slices, for $i = 1, ..., l$. Fig. 6 shows a three-level Almost-σ Variant.

Algorithm 4 Almost-σ-Variant

for $j = l$ down to 2 **do**
  if $z_j < \sigma_j$ **then**
    Bring in (and remove) the next available slice of a larger size and create additional $\sigma_j - z_j$ level-j slices.
    $z_j \leftarrow \sigma_j$.
    The remaining portion of the removed level-$(j-1)$ slice forms another slice.
  end if
end for

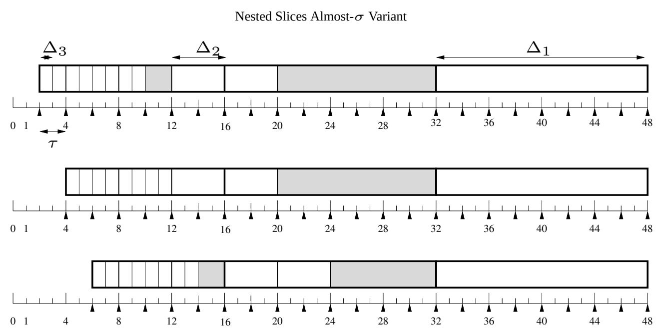


Fig. 6. Three-level nested slice structure Almost-σ Variant. $\tau = 2$, $\Delta_1 = 16$, $\Delta_2 = 4$ and $\Delta_3 = 1$. The anchored slice sets shown are for $t = \tau, 2\tau$ and 3τ, respectively. $\sigma_3 = 8$, $\sigma_2 = 2$. The shaded areas are also slices, but are different in size from any level-j slice, $j = 1$, 2 or 3.

IV. Evaluation

This section describes the performance results of different variations of our AC/scheduling algorithms. We also evaluate the required computation time to determine the scalability of our algorithms.

Most of the experiments are conducted on the Abilene network, which consists of 11 backbone nodes connected by 10 Gbps links. Each backbone node is connected to a randomly generated stub network. The link speed between each stub network and the backbone node is 1 Gbps. The entire network has 121 nodes and 490 links. For the scalability study of the algorithms, we use random networks with the number of nodes ranging from 100 to 1000. The random network generator takes the number of nodes and the average node degree as arguments, from which it computes the total number of links in the network. Then, it repeatedly picks a node pair uniformly at random from those unconnected node pairs, and connects them with a pair of links in both directions. This process is repeated until all links are assigned. We use the commercial CPLEX package for solving linear programs on Intel-based workstations. Each workstation has a dual-core processor and 4 GB of memory.

Unless mentioned otherwise, we use the following experimental models and parameters. Job requests arrive following a Poisson process. In order to simulate the file size (i.e., the demand size D_i) distribution, we resort to the heavy-tailed Pareto distribution, with the distribution function $F(x) = 1 - (x/b)^{-\alpha}$, where $x \geq b$ and $\alpha > 1$. It is known that the Internet traffic has the heavy-tail distribution. The heavy-tail Pareto distribution is widely used for generating simulated Internet traffic. It has the property that large files occur with a non-negligible probability.

As argued in Section I, files in e-science are often much larger than those in the ordinary Internet environment. That is, the file size distribution in e-science is more heavy-tailed than the Internet traffic. It appears appropriate to use the Pareto distribution to generate very large files in our simulation. The closer α is to 1, the more heavy-tailed is the distribution, and the more likely it is to have jobs with very large sizes. In most of our experiments, the average file size is 50 GB and $\alpha = 1.3$. By default, each job uses 8 shortest paths. We adopt this approach because our experiments on multi-path scheduling revealed the following result (See also [16].): For a typical network with several hundred nodes, 8 shortest paths are sufficient for achieving near optimal performance while keeping the algorithm execution time within the range of practicality[6]. We evaluate our algorithms under three traffic loads, namely, light, medium and heavy. By light, medium and heavy traffic loads, we mean that the average inter-arrival time between jobs is 5 minutes, 2 minutes and 30 seconds, respectively. In order to obtain stable results, we generated jobs under these different traffic loads for a period of 3 days. For the heavy traffic load, roughly 10,000 file transfer requests were generated.

We will compare the uniform slice (US) and the nested slice structures (NS) of the Almost-σ Variant type. For US, the time slice and AC/scheduling interval (τ) is 21.17 minutes. This corresponds to 68 slices in every 24-hour period. For NS, we use a two-level NS structure with 48 fine (level-2) slices and 20 coarse (level-1) slices. The fine slice size is $\Delta_2 = 5$ minutes, and the coarse slice size is $\Delta_1 = 60$ minutes. These parameters are chosen so that the first 24-hour period is divided into 68 fine and coarse slices, the same number as the US case. The AC/scheduling interval τ is 5 minutes, which is finer than the US case.

The plots and tables use acronyms to denote the algorithms used in the experiments. Recall that SR stands for Subtract-Resource and RR stands for Reassign-Resource in admission control; LB stands for Load-Balancing as the scheduling objective and QF stands for Quick-Finish.

The performance measures are:

- Rejection ratio: This is the ratio between the number of jobs rejected and total number of job requests. From the network's perspective, it is desirable to admit as many jobs as possible.
- Response time: This is the difference between the completion time of a job and the time when it is first being transmitted. From an individual job's perspective, it is desirable to have shorter response time.

A. *Comparison of Algorithm Execution Time*

Before comparing the performance of the algorithms, we first compare their execution time. Short execution time is important for the practicality of our centralized network control strategy. The results on the execution time put the performance comparison (Section IV-B) in perspective: Better performance often comes with longer execution time. Table I shows the execution time of different schemes under two representative traffic conditions.

[6]We ignore the connection setup (path setup) time because, due to the small network size, we can pre-compute and store the allowed paths for every possible source-destination pair.

TABLE I

AVERAGE AC/SCHEDULING ALGORITHM EXECUTION TIME (S)

Algorithm	Heavy Load		Light Load	
	AC	Scheduling	AC	Scheduling
US+SR+LB	13.13	5.70	0.40	0.61
US+SR+QF	12.03	1.86	0.32	0.23
US+RR+LB	80.89	5.89	1.05	0.65
US+RR+QF	34.36	4.74	0.36	0.21
NS+SR+LB	1.54	4.50	0.14	0.60
NS+SR+QF	1.57	1.60	0.13	0.07
NS+RR+LB	25.16	4.30	1.07	0.61
NS+RR+QF	17.43	3.54	0.17	0.06

1) SR vs. RR and LB vs. QF: The results show that, for admission control, SR can have much shorter average execution time than RR. This is because, in SR, AC works only on the new jobs, whereas in RR, AC works on all the jobs currently in the system. Hence, for SR, the **AC(**k**,** J**)** feasibility problem usually has much fewer variables.

When the AC algorithm is fixed, the choice of the scheduling algorithm, LB or QF, also affects the execution time for AC. For instance, the RR+LB combination has much longer execution time for AC than the RR+QF combination. This is because, in LB, each job tends to be stretched over time in an effort to reduce the network load on each time slice. This results in more jobs and more active slices (slices in $\mathcal{L}_k$) in the system at any moment, which means more variables for the linear program.

For scheduling, since LB and QF are very different linear programs, it is difficult to explain the differences in their execution time. But,we do observe that LB has longer execution time, again, possibly due to more variables for the reason stated in the previous paragraph.

2) US vs. NS: Depending on the number of levels for the NS, the number of slices at each level and the slice sizes, the NS can be configured to achieve different objectives: improving the algorithm performance, reducing the execution time, or doing both simultaneously. Our experimental results in Table I correspond to the third case. Since the two-level NS structure has $\Delta_1 = 60$ minutes and the US has the uniform slice size $\Delta = 21.17$ minutes, the NS typically has fewer slices than the US. For instance, under heavy load, US+RR+QF uses 150.5 active slices on average for AC, while NS+RR+QF uses 129.6 active slices on average. The number of variables, which directly affect the computation time of the linear programs, is generally proportional to the number of slices.

Part of the performance advantage of the NS (to be shown in Section IV-B) is attributed to the smaller scheduling interval τ. To reduce the scheduling interval for the US, we must reduce the slice size Δ, since $\Delta = \tau$ in the US. In the next experiment, we set the US slice size to be 5 minutes, which is equal to the size of the finer slice in the NS. Table II shows the performance and execution time comparison between the US and NS. Here, we use RR for admission control and QF for Scheduling. The US and NS have nearly identical performance in terms of the response time and job rejection ratio. But, the NS is far superior in execution times for both AC and scheduling.

Upon closer inspection (Table III), the NS requires far fewer active time slices than the US on average.

TABLE II

COMPARISON OF US AND NS ($\tau = 5$ MINUTES)

	Response Time (min)	Rejection Ratio	Execution Time (s) AC	Execution Time (s) Scheduling
LIGHT LOAD				
US	6.064	0	0.469	0.309
NS	5.821	0	0.162	0.062
MEDIUM LOAD				
US	9.767	0.006	3.177	2.694
NS	9.354	0.006	0.587	0.387
HEAVY LOAD				
US	16.486	0.183	131.958	26.453
NS	17.107	0.173	17.428	3.539

TABLE III

AVERAGE NUMBER OF SLICES OF US AND NS ($\tau = 5$ MINUTES)

		Average Number of Slices AC	Average Number of Slices Scheduling
Light Load	US	299.0	299.9
	NS	68.9	69.0
Medium Load	US	421.6	462.9
	NS	79.1	82.1
Heavy Load	US	975.1	799.8
	NS	129.6	113.4

In summary,

- SR is much faster than RR for admission control.
- LB tends to be slower than QF for both AC and scheduling.
- The NS requires much shorter execution time than the US, or achieves better performance, or has both properties.

The advantage of the NS can be extended by increasing the number of slice levels. In practice, it is likely that the US is too time consuming and the NS is a must.

B. Performance Comparison of the Algorithms

In this subsection, the experimental parameters are as stated in the introduction for Section IV. In particular, we fix the number of paths per job (K) to be 8. Table IV shows the response time and rejection ratio of different algorithms.

TABLE IV

PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS

Algorithm	Light Load		Medium Load		Heavy Load	
	Response Time (s)	Rejection Ratio	Response Time (s)	Rejection Ratio	Response Time (s)	Rejection Ratio
US+SR+LB	46.55	0	42.35	0.056	35.56	0.423
US+SR+QF	21.51	0.014	22.21	0.100	23.56	0.477
US+RR+LB	46.55	0	40.73	0.026	35.73	0.313
US+RR+QF	21.55	0	23.36	0.021	25.16	0.312
NS+SR+LB	49.60	0	43.83	0.021	28.74	0.237
NS+SR+QF	5.73	0.006	7.56	0.052	11.06	0.403
NS+RR+LB	49.60	0	43.88	0.011	30.16	0.168
NS+RR+QF	5.82	0	9.35	0.006	17.11	0.173

1) US vs. NS: In Table IV, the algorithms with the NS have a comparable to much better performance than those with the US. Furthermore, it has already been established in Section IV-A that the NS has much shorter algorithm execution time.

2) Best Performance: The best performance in terms of both response time and the rejection ratio is achieved by the RR+QF combination.

Suppose we fix the slice structure and the scheduling algorithm. Then, SR has worse rejection ratio than RR because SR does not allow flow reassignment for the old jobs during the admission control. Since response time increases with the admitted traffic load, an algorithm that leads to lower rejection ratio can have higher response time. This explains why RR often has higher response time than the corresponding SR algorithm. Note that a lower rejection ratio does not *always* lead to higher traffic load since some algorithms, such as RR, use the network capacity more efficiently.

Suppose we fix the slice structure and the AC algorithm. Then, LB does much worse than QF in terms of response time, because LB tends to stretch a job until its requested end time while QF tries to complete a job early if possible. If RR is used for admission control, then under high load, the different scheduling algorithms have a similar effect on the rejection ratio of the next admission control operation. However, for medium load we notice that the work conserving nature of QF contributes to a lower rejection ratio than LB, which tends to waste some bandwidth.

3) Merits of SR and LB: Given the above discussion, one may quickly dismiss SR and LB. But, as we have noted in Section IV-A, SR can have considerably shorter execution time than RR. Furthermore, it is a candidate for conducting realtime admission control at the instance when a request is made, which is not possible with RR.

If SR is used, then LB often has a lower rejection ratio than QF. The reason is that QF tends to highly utilize the network on earlier time slices, making it more likely to reject small jobs requested for the near future. This is a legitimate concern because, in practice, it is more likely that small jobs are requested to be completed in the near future rather than the more distant future.

There is indication that the more heavy-tailed is the file size distribution, the larger is the difference in rejection ratio between LB and QF. The evidence is shown in Fig. 7 for the light traffic load. As the Pareto parameter α approaches 1 while the average job size is held constant, the chance of having very large files increases. Even if they are transmitted at the full network capacity as in QF, such large files can still congest the network for a long time, causing more future jobs to be rejected. The correct thing to do, if SR is used, is to spread out the transmission of a large file over its requested time interval.

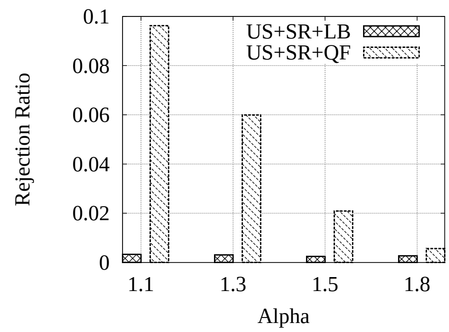


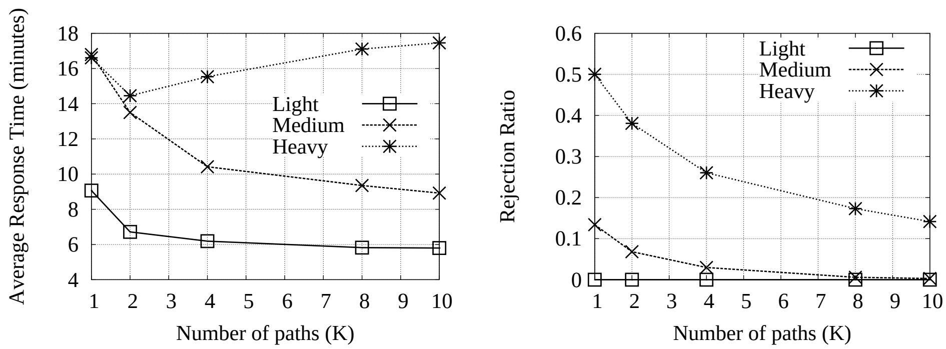
Fig. 7. Rejection ratio for different α's under SR.

To summarize the key points, between the admission control methods, RR is much more efficient in utilizing the network capacity, which leads to fewer jobs being rejected, while SR is suitable for fast or realtime admission control; if SR is used for admission control, then the scheduling method LB is superior to QF in terms of the rejection ratio.

C. Single vs Multi-path Scheme

The effect of using multiple paths is shown in Fig. 8 for the light, medium and heavy traffic loads. Here, the NS is used along with the admission control scheme RR, and the scheduling objective QF. For every source-destination node pair, the K shortest paths between them are selected and used by every job between the node pair. We vary K from 1 to 10 and find that multiple paths often produce better response time and always produce a lower rejection ratio. The amount of improvement depends on many factors such as the traffic load, the version of the algorithm, and the network parameters. For the light load, no job is rejected. As the number of paths per job increases from 1 to 8, we get 35% reduction in the response time. No further improvement is gained with more than 8 paths. For the medium load, the response time is almost halved as the number of paths varies from 1 to 10. The improvement in the rejection ratio is even more impressive, from 13.3% down to 0.3%. For the heavy load, there is no improvement in the response time due to the significant reduction in the rejection ratio: With multiple paths, many more jobs are admitted, resulting in a large increase of the actual network load.

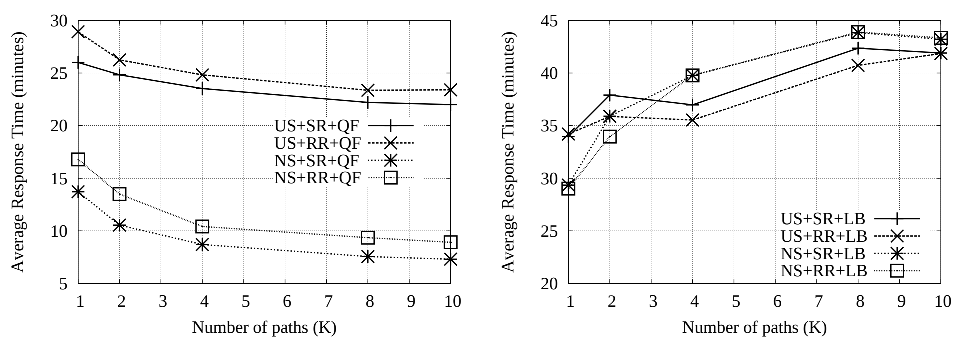
Fig. 9 and Fig. 10 show the response time and the rejection ratio, respectively, under the medium traffic load for all algorithms. It is observed that the rejection ratio decreases significantly for all algorithms as K increases. All the algorithms that use LB for scheduling experience an increase in the response time due to the reduction in the rejection ratio. But, this is not a disappointing result because it is not a goal of LB to reduce the response time. All



(a) (b)

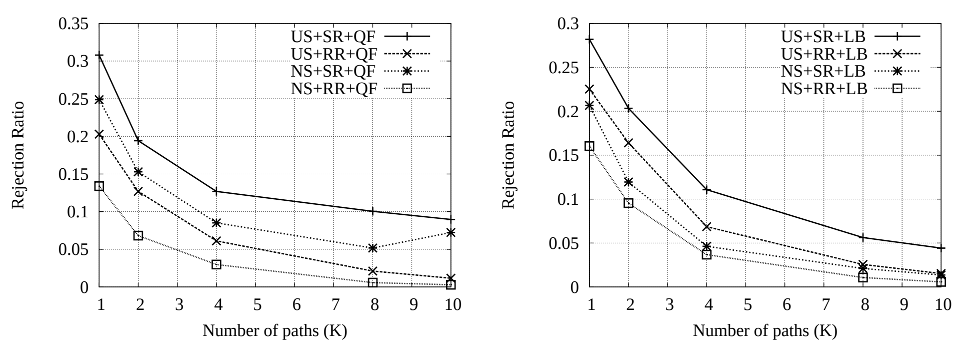
Fig. 8. Single vs. multiple paths under different traffic load. (a) Response time; (b) Rejection ratio.

the algorithms using QF for scheduling experience a decrease in the response time. Inspite of the increased load, QF is able to pack more jobs in earlier slices by utilizing the additional paths.



(a) (b)

Fig. 9. Single vs. multiple paths under medium traffic load for different algorithms. (a) Response time for QF; (b) Response time for LB.



(a) (b)

Fig. 10. Single vs. multiple paths under medium traffic load for different algorithms. (a) Rejection ratio for QF; (b) Rejection ratio for LB.

D. Comparison against Typical AC/Scheduling Algorithm

The next experiment compares our AC/scheduling algorithms with a simple, incremental AC/scheduling algorithm, which will be called the *simple scheme*. The simple scheme decouples AC from routing, and assumes a single default path given by the routing protocol. AC is conducted in realtime upon the arrival of a request. The requested resource is compared with the remaining resource in the network on the default path. If the latter is sufficient, then the job is admitted. The remaining resource is updated by subtracting from it what is allocated to the new request by the scheduling step (See next.).

Compared to our AC/scheduling algorithms, the simple scheme resembles our SR admission control algorithm but allows only one path for each job. For bulk transfer with start and end time constraints, the simple scheme still requires a scheduling stage, because bandwidth needs to be allocated to the newly admitted job over the time slices on its default path. We can apply the time slice structure and the scheduling objective of LB or QF to the newly admitted job. However, unlike our scheduling algorithm, the scheduling algorithm in the simple scheme does not reschedule the *old* jobs; that is, it does not change the bandwidth allocation for the old jobs.

The reason we use the simple scheme as the baseline for comparison with our algorithms is that it is fairly general: The basic part of the simple scheme is really what most other systems or proposals use. If we remove the advance reservation part, the AC in the simple scheme resembles a typical AC algorithm proposed for most traditional QoS architectures for large networks [23], [24], [25], [26]. With advance reservation, it is similar to most proposals for AC in research networks [13], [22]. But, compared with most other schemes, the simple scheme has something extra: The bandwidth for a job can be different from slice to slice. Hence, the performance of the simple scheme is at least as good as, and nearly always better than, that of other exiting schemes.

Table V shows the rejection ratio of the simple scheme with different slice structures and scheduling algorithms for different traffic loads. This should be compared with Table IV. The simple scheme leads to considerably higher rejection ratios than all of our algorithms involving SR, which in turn have higher rejection ratios than the corresponding algorithms involving RR.

TABLE V

REJECTION RATIO OF THE SIMPLE SCHEME

	Light Load	Medium Load	Heavy Load
US+SR+LB	0.010	0.345	0.781
US+SR+QF	0.031	0.308	0.792
NS+SR+LB	0	0.225	0.596
NS+SR+QF	0.026	0.249	0.642

E. Scalability of AC/Scheduling Algorithms

For this experiment, we assume that all job requests arrive at the same time and have the same start and end time requirement. Hence, the AC/scheduling algorithms run only once. The objective is to determine how the execution time of the algorithms scales with the number of simultaneous jobs in the system, the number of time slices used, or the network size. In this case, RR and SR are indistinguishable. In the following results, we use the US+SR+QF scheme.

Fig. 11 shows the execution time of AC and scheduling as a function of the number of jobs. The interval between the start and end times is partitioned into 24 uniform time slices. It is observed that the increase in the execution time is linear or slightly faster than linear. Scaling up to thousands of simultaneous jobs appears to be possible.

Fig. 12 shows the execution time against the number of time slices for 100 requests. The increase is linear. With respect to the execution time, the practical limit is several hundred slices. This is sufficient if the NS is used. But with the US, the slice size may be too coarse if one wishes to cover several months of advance reservations.

Fig. 13 shows the scalability of the algorithms against the network size. For this, we generate random networks with 100 to 1000 nodes in 100-node increments. The average node degree is $5, 5, 7, 9, 9, 10, 10, 11, 11$, and 11, respectively, so that the number of edges also increases. The network link capacity ranges from 0.1 Gbps to 10 Gbps. There are 100 jobs to be admitted and scheduled. It is observed that the execution time increases slightly faster than linear, indicating acceptable scaling behavior.

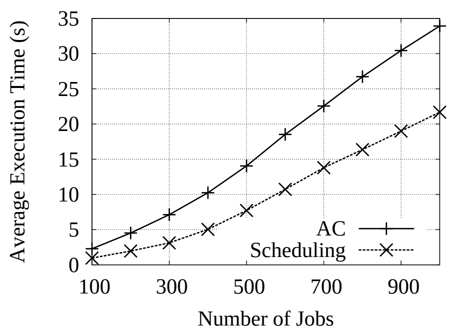


Fig. 11. Scalability of the execution times with the number of jobs.

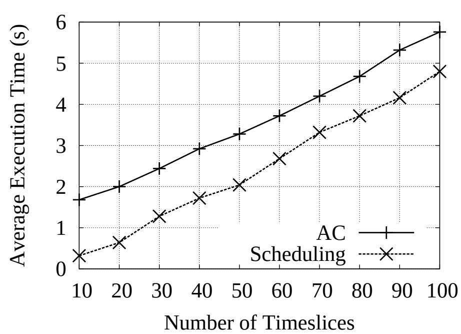


Fig. 12. Scalability of the execution times with the number of time slices.

V. Related Work

Compared with the traditional QoS frameworks, such as InterServ [27], DiffServ [28], the ATM network [23], or MPLS [24], admission control and scheduling for research networks are recent concerns with much fewer published studies.

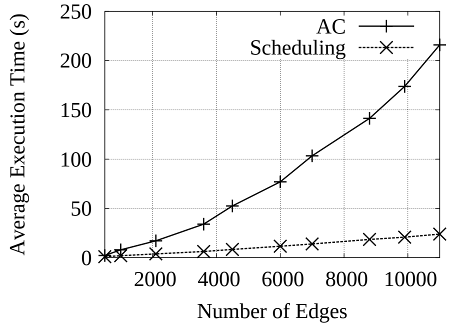


Fig. 13. Scalability of the execution times with the network size.

A. Bulk Transfer

Recent papers on AC and scheduling algorithms for bulk transfer with advance reservations include [14], [15], [16], [17], [18], [19], [13], [20], [21]. In [13], the AC and scheduling problem is considered only for the single link case. Network-level AC and scheduling are considered to be outside the scope of [13]. As a result, multi-path routing and network-level bandwidth allocation and re-allocation have no counter-part in [13]. Moreover, the solution is a heuristic one instead of an optimal one. Finally, once a job is admitted permanently, it won't be reconsidered in the future. In contrast, we periodically re-optimize the bandwidth assignment for all the new and old jobs.

In one of our earlier papers [16], we focus on a one-time scheduling subproblem, as apposed to periodic scheduling, and conduct a detailed performance comparison between single-slice scheduling and multi-slice scheduling under various slice sizes, and between single-path routing, multi-path routing and an arc-flow formulation, which is equivalent to allowing all possible paths for every job. We conclude that having a small number of paths per job is usually sufficient to yield near-optimal throughput, which is defined as what is achievable when all possible paths are allowed. Multi-slice scheduling is justified for its significant performance (e.g., throughput) improvement. Another research team has also considered the similar problem but with different emphasis [14].

In [17], [18], [19], the authors consider single-link admission control or link-by-link admission control under single-path routing. The admission control uses heuristic algorithms instead of solutions to optimization problems. Based on its size and the deadline, the average required bandwidth of a bulk transfer job is computed. The admission control is based on the job's average bandwidth requirement. The bandwidth of existing jobs may be re-allocated in the single link case but not in the network case.

The authors of [21] propose a malleable reservation scheme for bulk transfer, which checks every possible interval between the requested start time and end time for the job and tries to find a path that can accommodate the entire job on that interval. The scheme favors intervals with earlier deadlines. In [20], the computational complexity of a related path-finding problem is studied and an approximation algorithm is suggested. [15] starts with an advance reservation problem for bulk transfer. Then, the problem is converted into a constant bandwidth allocation problem at a single time instance to maximize the job acceptance rate. This is shown to be an NP-hard problem. Heuristic algorithms are then proposed. In [15], all the requests are known at the time of the admission control and no additional requests come later. The AC/scheduling is carried out only once. Our focus is quite different. We assume that the requests continue to arrive and the AC/scheduling must be done repeatedly. The concern for us is how to optimize and re-optimize the bandwidth assignment to the jobs as new job requests arrive, so that the early commitments are not violated and the network resource is used efficiently. In [15], the bandwidth constraints are at the ingress and egress links only. As a result, there is no routing issue. In our case, we have a full network and we use multiple paths for each job. We may alter the bandwidth assignment on the paths for each existing job in the system in order to accommodate later jobs.

B. MBG Traffic

Several earlier studies [29], [30], [31], [32] have considered admission control at an individual link for the MBG (minimum bandwidth guarantee) traffic class with start and end times. The concern is typically about designing efficient data structures, such as the segment tree [30], for keeping track of and querying bandwidth usage at the link on different time intervals. The admission of a new job is based on the availability of the requested bandwidth between the job's start time and end time. [20], [33], [21], [34] and [31] go beyond single-link advance reservation and tackle the more general path-finding problem for the MBG class, but typically only for new requests, one at a time. The routes and bandwidth of existing jobs are unchanged. [35] considers a network with known routing in which each admitted job derives a profit. It gives approximation algorithms for admitting a subset of the jobs so as to maximize the total profit.

C. Other Related Work

The authors of [36] also advocate periodic re-optimization to determine new bandwidth allocation in optical networks. However, they do not assume that users make advance reservations with requested end times. As a result, [36] does not have the admission control step. In the scheduling step, it uses a multi-commodity flow formulation for bandwidth assignment, similar to our formulation but without the time dimension. That is, the scheduling problem in [36] is for a single (large) time slice, rather than over multiple time slices. Many papers study advance reservation, re-routing, or re-optimization of lightpaths, at the granularity of a wavelength, in WDM optical networks [37], [38]. But, they do not consider the start and end time constraint.

D. Control Plane Protocols, Architectures and Tools

This paper focuses on the AC and scheduling *algorithms*. A complete solution for the intended e-science application will also need the control plane protocols, architectures and middleware toolkits, which are considered outside the scope of the paper. In the control plane, [22] presents an architecture for advance reservation of intra and interdomain lightpaths. The DRAGON project [11] develops control plane protocols for multi-domain traffic engineering and resource allocation on GMPLS-capable [39] optical networks. GARA [9], the reservation and allocation architecture for the grid computing toolkit Globus [10], supports advance reservation of network and computing resources. [40] adapts GARA to support advance reservation of lightpaths, MPLS paths and DiffServ paths. GridJIT [41] is another signaling protocol for setting up and managing lightpaths in optical networks for grid computing applications. ODIN [42] is a toolkit for optical network control and management for supporting grid computing. Another such toolkit is reported in [43]. [44] discusses the architectural and signaling-protocol issues for advance reservation of network resources.

VI. Conclusion

This study aims at contributing to the management and resource allocation of research networks for data-intensive e-science collaborations. The need for large file transfer and high-bandwidth, low-latency network paths is among the main requirements posed by such applications. The opportunities lie in the fact that research networks are generally much smaller in size than the public Internet, and hence afford a centralized resource management platform. This paper combines the following novel elements into a cohesive framework of admission control and flow scheduling: advance reservation for bulk transfer and minimum bandwidth guaranteed traffic, multi-path routing, and bandwidth reassignment via periodic re-optimization.

To handle the start and end time requirement of advance reservation, as well as the advancement of time, we identify a suitable family of discrete time-slice structures, namely, the congruent slice structures. With such a structure, we avoid the combinatorial nature of the problem and are able to formulate several linear programs as the core of our AC and scheduling algorithms. Moreover, we can develop simple algorithms that can retain the performance guarantee for the existing jobs in the system while admitting new jobs. Our main algorithms apply to all congruent slice structures, which are fairly rich. In particular, we describe the design of the nested slice structure and its variants. They allow the coverage of a long segment of time for advance reservation with a small number of slices without compromising performance. They lead to reduced execution time of the AC/scheduling algorithms, thereby making it practical. The following inferences were drawn from our experiments.

- The algorithms can handle up to several hundred time slices within the time limit imposed by practicality concern. If the NS is used, this number can cover months, even years, of advance reservation with sufficient time slice resolution. If the US is used, either the duration of coverage must be significantly shortened or the time slice be kept very coarse. Either approach tends to degrade the algorithms' utility or performance.
- We have argued that between the admission control methods, RR is much more efficient than SR in utilizing the network capacity, thereby, leading to fewer jobs being rejected. On the other hand, SR is suitable for fast or realtime admission control. If SR is used for admission control, then the scheduling method LB is superior to QF in terms of the rejection ratio. We have also observed that using multiple paths improves the network utilization dramatically.
- The execution time of our AC/scheduling algorithms exhibits acceptable scaling behavior, i.e., linear or slightly faster than linear scaling, with respect to the network size, the number of simultaneous jobs, and the number of slices. We have high confidence that they can be practical. The execution time can be further shortened by using fast approximation algorithms, more powerful computers, and better decomposition of the algorithms for parallel implementation.

Even in the limited application context of e-science, admission control and scheduling are large and complex problems. In this paper, we have limited our attention to a set of issues that we think are unique and important. This work can be extended in many directions. To name just a few, one can develop and evaluate faster approximation algorithms as in [45], [46]; address additional policy constraints for the network usage; incorporate the discrete lightpath scheduling problem; develop a price-based bidding system for making admission request; or address more carefully the needs of the MBG traffic class, such as minimizing the end-to-end delay.

The AC/scheduling algorithms presented in this paper are only part of a complete solution for the intended e-science applications. Control plane protocols and middleware are needed for setting up the network paths, controlling the bandwidth allocation, and for the end systems to take advantage of the new networking capabilities. The software tools should also automate the user-network interaction, such as the request submission and re-negotiation process. There are several projects in protocol, architecture and toolkit development, mainly in the grid computing community, as discussed in Section V. Developing similar protocols and adding new components to the existing toolkits in support of our algorithms are among the future tasks.

APPENDIX I
FREQUENTLY USED NOTATIONS

Frequently used notations and definitions are summarized in Table VI.

REFERENCES

[1] The U.K. Research Councils, http://www.research-councils.ac.uk/escience/, site last visited on Feb. 18, 2008.

[2] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[3] H. B. Newman, M. H. Ellisman, and J. A. Orcutt, "Data-intensive e-science frontier research," *Communications of the ACM*, vol. 46, no. 11, pp. 68–77, Nov. 2003.

[4] J. Bunn and H. Newman, "Data-intensive grids for high-energy physics," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. John Wiley & Sons, Inc, 2003.

[5] T. DeFanti, C. d. Laat, J. Mambretti, K. Neggers, and B. Arnaud, "TransLight: A global-scale LambdaGrid for e-science," *Communications of the ACM*, vol. 46, no. 11, pp. 34–41, Nov. 2003.

[6] National Lambda Rail, http://www.nlr.net, site last visited on Feb. 18, 2008.

[7] Abilene Network, http://www.internet2.edu/network/, site last visited on Feb. 18, 2008.

[8] CA*net4, http://www.canarie.ca/canet4/index.html, site last visited on Feb. 18, 2008.

[9] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *Proceedings of the International Workshop on Quality of Service (IWQoS '99)*, 1999.

[10] The Globus Aliance, http://www.globus.org/, site last visited on Feb. 18, 2008.

[11] T. Lehman, J. Sobieski, and B. Jabbari, "DRAGON: A framework for service provisioning in heterogeneous grid networks," *IEEE Communications Magazine*, March 2006.

[12] T. Ferrari, *Grid Network Services Use Cases from the e-Science Community*, The Open Grid Forum, Dec. 2007, http://www.ogf.org/, site last visited on Feb. 18, 2008.

[13] S. Naiksatam and S. Figueira, "Elastic reservations for efficient bandwidth utilization in LambdaGrids," *The International Journal of Grid Computing*, vol. 23, no. 1, pp. 1–22, January 2007.

[14] B. B. Chen and P. V.-B. Primet, "Scheduling deadline-constrained bulk data transfers to minimize network congestion," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, May 2007.

TABLE VI

FREQUENTLY USED NOTATIONS AND DEFINITIONS

C_e	Capacity of link e
D_i	Demand size of job i
S_i, $\hat{S}_i$	Start time and rounded start time of job i
E_i, $\hat{E}_i$	End time and rounded end time of job i
τ	Interval between consecutive AC/scheduling runs
Δ_i	Duration of a level-i slice in the NS
σ_i	Number of level-i slices in the NS
In the following, assume $t = k\tau$.	
$\mathcal{G}_k$	Slice set anchored at time $k\tau$
M_k	Index of the last slice in which some rounded end time falls
$\mathcal{L}_k \subset \mathcal{G}_k$	Finite slice set $1, ..., M_k$
$ST_k(i)$, $ET_k(i)$	Start and end times of slice i
$LEN_k(i)$	Length of slice i
$I_k(t)$	Index of the slice that time t falls in
$\mathcal{J}_k^o$	Set of the old jobs
$\mathcal{J}_k^n$	Set of the new jobs
$\mathcal{J}_k^a$	Set of the admitted jobs
$P_k(s, d)$	Allowable paths from node s to d
$R_k(i)$	Remaining demand of job i
$f_i(p, j)$	Total flow allocated to job i on path p on slice j
$C_e(j)$	Remaining capacity of link e on slice j

[15] L. Marchal, P. Primet, Y. Robert, and J. Zeng, "Optimal bandwidth sharing in grid environment," in *Proceedings of IEEE High Performance Distributed Computing (HPDC)*, June 2006.

[16] K. Rajah, S. Ranka, and Y. Xia, "Scheduling bulk file transfers with start and end times," *Computer Networks*, to Appear. Manuscript avaiable at http://dx.doi.org/10.1016/j.comnet.2007.12.005. A short version is published at NCA 2007.

[17] K. Munir, S. Javed, M. Welzl, and M. Junaid, "Using an event based priority queue for reliable and opportunistic scheduling of bulk data transfers in grid networks," in *Proceedings of the 11th IEEE International Multitopic Conference (INMIC 2007)*, December 2007.

[18] K. Munir, S. Javed, M. Welzl, H. Ehsan, and T. Javed, "An end-to-end QoS mechanism for grid bulk data transfer for supporting virtualization," in *Proceedings of IEEE/IFIP International Workshop on End-to-end Virtualization and Grid Management (EVGM 2007)*, San Jose, California, October 2007.

[19] K. Munir, S. Javed, and M. Welzl, "A reliable and realistic approach of advance network reservations with guaranteed completion time for bulk data transfers in grids," in *Proceedings of ACM International Conference on Networks for Grid Applications (GridNets 2007)*, San Jose, California, October 2007.

[20] R. Guerin and A. Orda, "Networks with advance reservations: The routing perspective," in *Proceedings of IEEE INFOCOM 99*, 1999.

[21] L.-O. Burchard and H.-U. Heiss, "Performance issues of bandwidth reservation for grid computing," in *Proceedings of the 15th Symposium on Computer Archetecture and High Performance Computing (SBAC-PAD'03)*, 2003.

[22] E. He, X. Wang, V. Vishwanath, and J. Leigh, "AR-PIN/PDC: Flexible advance reservation of intradomain and interdomain lightpaths," in *Proceedings of the IEEE GLOBECOM 2006*, 2006.

[23] D. E. McDysan and D. L. Spohn, *ATM Theory and Applications*. McGraw-Hill, 1998.

[24] E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol label switching architecture*, RFC 3031, IETF, Jan. 2001.

[25] F. P. Kelly, P. B. Key, and S. Zachary, "Distributed admission control," *IEEE Journal On Selected Areas In Communications*, vol. 18, no. 12, Dec. 2000.

[26] G. de Veciana, G. Kesidis, and J. Walrand, "Resource management in wide-area ATM networks using effective bandwidths," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1081–1090, Aug. 1995.

[27] R. Braden, D. Clark, and S. Shenker, *Integrated services in the internet architecture: An overview*, RFC 1633, IETF, June 1994.

[28] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An architecture for differentiated services*, RFC 2475, IETF, Dec. 1998.

[29] O. Schelén, A. Nilsson, J. Norrgård, and S. Pink, "Performance of QoS agents for provisioning network resources," in *Proceedings of IFIP Seventh International Workshop on Quality of Service (IWQoS'99)*, London, UK, June 1999.

[30] A. Brodnik and A. Nilsson, "A static data structure for discrete advance bandwidth reservations on the Internet," Department of Computer Science and Electrical Engineering, Luleå University of Technology, Sweden, Tech. Rep. Tech report cs.DS/0308041, 2003.

[31] L.-O. Burchard, J. Schneider, and B. Linnert, "Rerouting strategies for networks with advance reservations," in *Proceedings of the First IEEE International Conference on e-Science and Grid Computing (e-Science 2005)*, Melbourne, Australia, Dec. 2005.

[32] Q. Xiong, C. Wu, J. Xing, L. Wu, and H. Zhang, "A linked-list data structure for advance reservation admission control," in *ICCNMC 2005*, 2005, lecture Notes in Computer Science, Volume 3619/2005.

[33] T. Wang and J. Chen, "Bandwidth tree - A data structure for routing in networks with advanced reservations," in *Proceedings of the IEEE International Performance, Computing and Communications Conference (IPCCC 2002)*, April 2002.

[34] T. Erlebach, "Call admission control for advance reservation requests with alternatives," Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH) Zurich, Tech. Rep. TIK-Report Nr. 142, 2002.

[35] L. Lewin-Eytan, J. Naor, and A. Orda, "Routing and admission control in networks with advance reservatione," in *Proceedings of the Fifth International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 02)*, 2002.

[36] R. Bhatia, M. Kodialam, and T. V. Lakshman, "Fast network re-optimization schemes for MPLS and optical networks," *Computer Networks*, vol. 50, no. 3, Feb. 2006.

[37] D. Banerjee and B. Mukherjee, "Wavelength-routed optical networks: linear formulation, resource budgeting tradeoffs, and a reconfiguration study," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 598–607, Oct. 2000.

[38] E. Bouillet, J.-F. Labourdette, R. Ramamurthy, and S. Chaudhuri, "Lightpath re-optimization in mesh optical networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 437–447, 2005.

[39] E. Mannie, *Generalized multi-protocol label switching (GMPLS) architecture*, RFC 3945, IETF, Oct. 2004.

[40] C. Curti, T. Ferrari, L. Gommans, B. van Oudenaarde, E. Ronchieri, F. Giacomini, and C. Vistoli, "On advance reservation of heterogeneous network paths," *Future Generation Computer Systems*, vol. 21, no. 4, pp. 525–538, Apr. 2005.

[41] S. R. Thorpe, D. Stevenson, and G. K. Edwards, "Using just-in-time to enable optical networking for grids," in *First ICST/IEEE International Workshop on Networks for Grid Applications (GridNets 2004)*, 2004.

[42] J. Mambretti, et al., "The photonic TeraStream: enabling next generation applications through intelligent optical networking at iGRID2002," *Future Generation Computer Systems*, vol. 19, no. 6, p. 897908, August 2003.

[43] R. Boutaba, W. Golab, Y. Iraqi, T. Li, and B. Arnaud, "Grid-controlled lightpaths for high performance grid applications," *Journal of Grid Computing*, vol. 1, no. 4, pp. 387–394, December 2003.

[44] L.-O. Burchard, "Networks with advance reservations: applications, architecture, and performance," *Journal of Network and Systems Management*, vol. 13, no. 4, pp. 429–449, Dec. 2005.

[45] N. Garg and J. Köenemann, "Faster and simpler algorithms for multi-commodity flow and other fractional packing problems," in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, November 1998, pp. 300–309.

[46] B. Awerbuch and F. T. Leighton, "Improved approximation algorithms for multi-commodity flow problem and local competitive routing in dynamic networks," in *Proceedings of the ACM Symposium on Theory of Computing*, 1994, pp. 487–496.