A flexible bandwidth reservation framework for bulk data transfers in grid networks

Bin Chen Bin, Pascale Primet

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

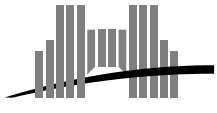
A flexible bandwidth reservation framework for bulk data transfers in grid networks

Bin Bin Chen ,
Pascale Primet

May 2006

Research Report N° 2006-20

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`



A flexible bandwidth reservation framework for bulk data transfers in grid networks

Bin Bin Chen , Pascale Primet

May 2006

Abstract

In grid networks, distributed resources are interconnected by wide area network to support compute and data-intensive applications, which require reliable and efficient transfer of gigabits (even terabits) of data. Different from best-effort traffic in Internet, bulk data transfer in grid requires bandwidth reservation as a fundamental service. Existing reservation schemes such as RSVP are designed for real-time traffic specified by reservation rate, transfer start time but with unknown lifetime. In comparison, bulk data transfer requests are defined in terms of volume and deadline, which provide more information, and allow more flexibility in reservation schemes, i.e., transfer start time can be flexibly chosen, and reservation for a single request can be divided into multiple intervals with different reservation rates. We define a flexible reservation framework using time-rate function algebra, and identify a series of practical reservation scheme families with increasing generality and potential performance, namely, FixTime-FixRate, FixTime-FlexRate, FlexTime-FlexRate, and Multi-Interval. Simple heuristics are used to select representative scheme from each family for performance comparison. Simulation results show that the increasing flexibility can potentially improve system performance, minimizing both blocking probability and mean flow time. We also discuss the distributed implementation of proposed framework.

Keywords: Reservation, grid, bulk data transfer, flexibility

Résumé

Dans les réseaux de grilles, les ressources distribuées sont interconnectées par des réseaux longues distance pour exécuter des applications intensives de calcul ou de traitement de données, qui nécessitent des transferts fiables et efficaces de volumes de données de l'ordre de plusieurs giga-octets ou teroctets. Le transferts massifs dans les grilles, contrairement au trafic "best effort" de l'Internet, requierent un service de réservation de bande-passante. Les schémas de réservation existants, tels RSVP, ont été conçus pour du trafic temps-réel et pour lequel on spécifie un débit réservé, une date de début de transfert mais on ne précise pas la durée. En comparaison, les transferts massifs de grilles sont définis en termes de volumes et de date limite, ce qui offre plus d'informations et autorise des schémas de réservation plus flexibles. Le début effectif du transfert peut être choisi, une réservation pour une même requête peut être divisés en plusieurs intervals avec des débits réservés différents. Nous définissons un cadre flexible de réservation de bande passante à l'aide d'une algèbre de fonctions temps-débit et identifions une série de familles de schémas de réservation, que nous nommons FixTime-FixRate, FixTime-FlexRate, FlexTime FlexRate, et Multi-Interval, présentant une généralité et un potentiel de performance croissants. Des heuristiques simples sont utilisées pour sélectionner un schéma représentatif dans chaque famille pour comparer les performances. Les résultats de simulation montrent que l'augmentation de la flexibilité peut potentiellement augmenter les performances du système, minimiser la probabilité de blocage et la durée moyenne des flux. Nous discutons aussi de l'implantation distribuée du cadre proposé.

Mots-clés: Réservation, grille, transferts massifs, flexibilité

1 Introduction

Grid computing is a promising technology that brings together large collection of geographically distributed resources (e.g., computing, storage, visualization, etc.) to build a very high performance computing environment for compute and data-intensive applications [7]. Grid networks connect multiple sites, each comprising a number of processors, storage systems, databases, scientific instruments, and etc. In grid applications, like experimental analysis and simulations in high-energy physics, climate modeling, earthquake engineering, drug design, and astronomy, massive datasets must be shared by a community of researchers distributed in different sites. These researchers transfer large subsets of data across network for processing. The *volume* of dataset can usually be determined from task specification, and a strict *deadline* is often specified to guarantee in-time completion of the whole task, also to enforce efficient use of expensive grid resources, not only network bandwidth, but also the co-allocated CPUs, disks, and etc.

While Internet bulk data transfer works well with best-effort service, high-performance grid applications require bandwidth reservation for bulk data transfer as a fundamental service. Besides strict deadline requirement and expensive co-allocated resources as we discussed above, the *smaller multiplexing level* of grid networks compared to Internet also serves as a main driving force for bandwidth reservation. In Internet, the source access rates are generally much smaller ($2Mbps$ for DSL lines) than the backbone link capacity (hundreds to thousands of Mbps, say). Coexistence of many active flows in a single link smoothes the variation of arrival demands due to the law of large number, and the link is not a bottleneck until demand attains above 90% of its capacity [13]. Thus no proactive admission control is used in Internet for bulk data transfer. Instead, distributed transport protocols, such as TCP, are used to statistically share available bandwidth among flows in a "fair" way. Contrarily, in grid context, the capacity of a single source ($c = 1Gbps$) is comparable to the capacity of bottleneck link. For a system with small multiplexing level, if no pro-active admission control is applied, burst of load greatly deteriorates the system performance.

A concrete example is given in Section 2 to demonstrate the importance of resource reservation for grid networks. Through the example, we also show that existing RSVP-type framework is not flexible enough for bulk data transfer reservation. In Section 3, we define a flexible reservation framework using time-rate function algebra. Section 4 identifies a series of practical reservation scheme families with increasing generality, and we use simple heuristics to select representative scheme from each family. In Section 5, simulation result of chosen schemes are presented and the impact of flexibility is analyzed. A distributed architecture is proposed in Section 6. In Section 7, we briefly review related works on bandwidth reservation. Finally, we conclude in Section 8.

2 Motivation

In Figure 1, we simulate a single link with capacity C. Bulk data transfer requests arrive according to a Poisson process with parameter λ. Request volume is independent of arrival time, and follows an exponential distribution with parameter μ. Simulations with other arrival processes and traffic volume distributiones reveal similar trend, which are not presented here for brevity. Load $\rho = \lambda/(C * \mu)$. Requests have maximal transfer rate R_{max}. In Internet setting $R_{max}^{Internet} = C/100$, and in grid setting $R_{max}^{grid} = C/10$. *Ideal transport protocol* is

assumed, so that if there are no more than C/R_{max} active flows, all of them transfer at full rate R_{max}. If there are $n > C/R_{max}$ active flows, they all transfer at rate C/n. A request with volume v "fails" and immediately terminates, if it does not complete transfer within v/R_{min} time, where $R_{min} \leq R_{max}$ is the expected average throughput of the request (in this example $R_{min} = R_{max}/2$ for all requests). In *Internet-NoAC* setting (AC stands for "Admission Control"), the fail probability is low until load ρ attains above 95%. In *grid-NoAC* setting, however, the fail probability is nonneglectable even under a medium load, and it deteriorates rapidly as load increases. Thus we consider using a simple reservation scheme, which enforces requests to reserve R_{min} bandwidth when they arrive, so that all accepted requests are guaranteed to complete before deadline (fail probability is 0). Requests are blocked if the number of active reservations reaches C/R_{exp}. This kind of reservation can be supported by existing reservation schemes, for example, RSVP [3]. In *grid-AC* setting, we still assume *ideal transport protocol*, i.e., accepted requests are able to fairly share unreserved capacity in addition to their reserved bandwidth. Block probability of *grid-AC* setting is much lower than fail probability of *grid-NoAC* setting.

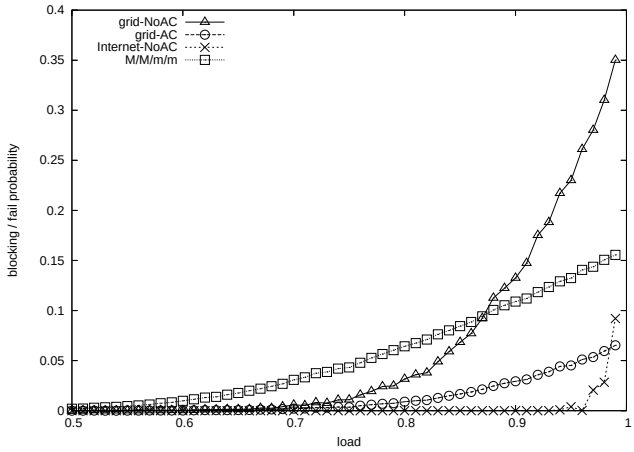


Figure 1: Fail/block probability under different multiplexing level

In Figure 1, we also plot a variation of *grid-AC* setting in which flows can only use reserved bandwidth. With this *dull transport protocol* assumption, the link can be modeled as a standard $M/M/m/m$ queuing system with $m = C/R_{min} = 10$. Comparing this $M/M/m/m$ setting against *grid-NoAC* setting, simple reservation scheme with dull transport protocol can still outperform no admission control setting with ideal transport protocol when load is relatively high. This again demonstrates the benefit of reservation. Meanwhile, the big performance gap between $M/M/m/m$ setting and *grid-AC* setting shows that when transport protocol is dull, a RSVP-type reservation does not fully exploit the system's capacity. The transport protocol design for high speed network is still an ongoing research. Complementarily, we consider how to improve system's performance by using more flexible reservation schemes in this paper.

RSVP is designed for real-time traffic which normally requests for a specified value of bandwidth from a fixed start time. Their lifetime is unknown, thus reservation remains in effect for an indefinite duration until explicit "Teardown" signal is issued or soft state expires. In stead, bulk data transfer requests are specified by *volume* and *deadline*. This allows more flexibility in the design of reservation schemes. As volume is known, the completion time

can be calculated by scheduler and kept in time-indexed reservation states. If there is not enough bandwidth at the moment a request arrives, transfer can be scheduled to start at some future time point as long as it can complete before deadline. Bandwidth reservation can also comprise sub-intervals with different reserved rates.

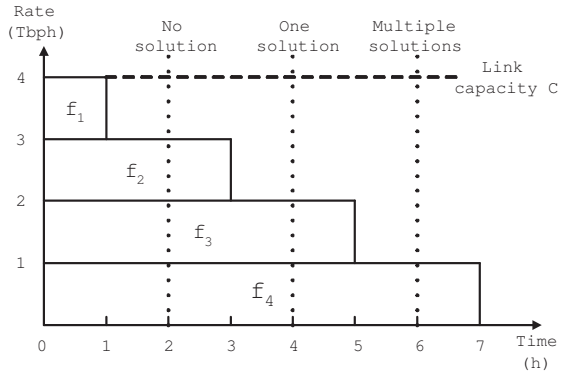


Figure 2: Flexible reservation schemes example

Limitation of RSVP-type reservation for bulk data transfer is illustrated in Figure 2. In this example, we consider a link with capacity $C = 4Tbph$. Requests arrive online with varying volume, their maximal transfer rate is $R_{max} = 2Tbph$ and their minimum average transfer rate is $R_{min} = 1Tbph$. A request arrives at time t with volume v has a deadline $t + v/R_{min}$. Assume at current time $0h$, there are four active reservations each reserving $1Tbph$ bandwidth. Their termination times are known and marked in the figure. A new request arrives at $0h$ with volume $v = 4Tb$, and its deadline is $0h + 4Tb/R_{min} = 4h$. Since there is no bandwidth left at time $0h$, this request will be rejected by RSVP-type reservation scheme. This unnecessary rejection can be avoided, if we use more flexible reservation scheme and exploit the time-indexed reservation state information. A feasible reservation solution is to reserving $1Tbph$ for the request from time $1h$ (other than from $0h$) until $3h$, followed by a different reservation rate of $2Tbph$ until $4h$.

In the case of $v = 4Tb$, this is the only solution to accept the request and guarantee its successful completion without preempting any existing reservations. However, if the request has volume $v = 2Tb$ and thus deadline $2h$, no feasible solution exists to accept the new request unless *preemption* is allowed. The concept of *preemption* is borrowed from job scheduling literature, which means the modification (including teardown) of the reservation state of an already-accepted request by system. Compared to *non-preemptive* schemes, *preemptive schedulers* enjoy higher decision flexibility which implies potential performance gain. But they have some drawbacks including:

- Dropping accepted request causes more dissatisfaction than blocking new one;
- Dynamic change (QoS degration) of reservation state hurts service predictability, which is important because bandwidth is co-allocated with other resources.

Also, it is challenging to design a distributed preemptive reservation architecture. In this paper, we focus on a *non-preemptive reservation framework*.

There may be multiple feasible solutions to accept a request, for example if the request here is with volume $v = 6Tb$ and deadline $6h$. The algorithm to select a solution out of all feasible

solutions depends on the objective functions of reservation schemes. Besides increasing accept probability, there are other important performance criteria. Borrowing concept again from job scheduling, *flow time* is defined as the time between a request's arrival and its completion. For bulk data transfers, especially in grid applications, it is desirable to minimize flow time. Smaller flow time not only improves users' satisfaction, but also releases all co-allocated resources earlier back to sharing pool. Fairness among flows is also an important performance criteria. For example, bulk data transfer may define fairness over their average throughput. These criteria may be conflicting with each other. For example, the solution to minimize flow time here is to reserve $1Tbph$ from $1h$ to $3h$, and $2Tbph$ from $3h$ to $5h$ so that the request can be finished at $5h$. While the solution to minimize peak reservation rate is to reserve $1Tbph$ from $1h$ to $3h$, and $4/3Tbph$ from $3h$ to $6h$. Yet another reasonable solution is to reserve $0.5Tbph$ from $1h$ to $3h$, $1.5Tbph$ from $3h$ to $5h$, followed by $2Tbph$ (R_{max}) from $5h$ to $6h$, so that the remained bandwidth variation along time axis is minimized.

It is very difficult (if not totally impossible) to identify the optimal solution in both off-line and on-line setting. Sometimes it is preferable to reject a request even when feasible solution exists. In this paper, we don't emphasis the choice of objective functions and optimal solutions. Instead, we focus on formalizing a flexible yet practical solution space, so that a potential candidate solution will not be missed because of the limitation in reservation framework flexibility.

3 Flexible reservation framework

3.1 System model

We model grid networks as a set of resources interconnected by wide area network. The underlying communication infrastructure of grid networks is a complex interconnection of enterprize domains and public networks that exhibit potential bottlenecks and varying performance characteristics. For simplicity, we assume a centralized scheduler manages reservation state vector L for all links in the system. We will discuss the distributed implementation in Section 6.

We define a request as a 6-tuple:

$$r = (s_r, d_r, v_r, a_r, d_r, R_r^{max}) \tag{1}$$

As suggested by name, source s_r requests to transfer bulk data of volume v_r to destination d_r. Request arrives at time a_r and transfer is ready to begin immediately. Transfer should complete before deadline d_r, and R_r^{max} is the maximum rate that request r can support, constrainted by either link capacity of end nodes, application or transport protocol.

A bandwidth scheduler makes decision for request based on system state $L(t)$ and request specification r. As shown in Figure 3, a scheduler first calculates constraint function $C_r(t)$ for the reservation, considering both request specification and current system state $L(t)$. Calculation of constraint is a *min* operation over *time-rate function* which will be defined below. Constraint function $C_r(t)$ then is used to make reservation decision $D_r(t)$. $D_r(t)$ is the output of scheduler, and is also used internally to update link state $L(t)$.

3.2 Time-rate function algebra

We denote the set of all time-rate functions as $\mathcal{F}$, and we define *Min-Plus* algebra over $\mathcal{F}$:

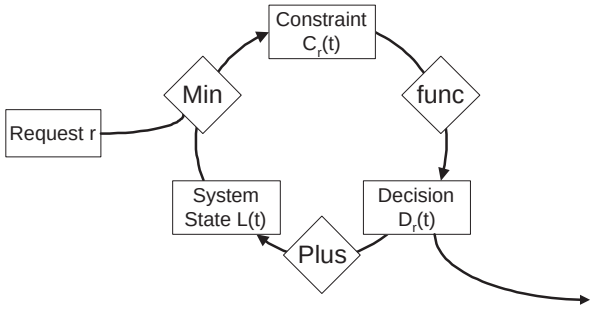


Figure 3: Reservation schemes algorithm framework

$$(f_1 \; min \; f_2)(t) = min(f_1(t), f_2(t)) \tag{2}$$

$$(f_1 + f_2)(t) = f_1(t) + f_2(t) \tag{3}$$

While Min-algebra is a semigroup, Plus-algebra is a group with identity element $f^0(t) = 0, \forall t \in (-\infty, \infty)$. We define $\leq$ relation over $\mathcal{F}$ as:

$$f_1 \leq f_2, \text{ iff } f_1(t) \leq f_2(t), \forall t \in (-\infty, \infty) \tag{4}$$

Note that $\mathcal{F}$ with $\leq$ is a partial order set not satisfying comparability condition.

a_r, d_r, R_r^{max} in request specification determines a time-rate function, which can be viewed as the original constraint function imposed by request specification:

$$C_r^{request}(t) = R_r^{max} h(t - a_r) - R_r^{max} h(t - d_r) \tag{5}$$

where:

$$h(t) = \begin{cases} 1 & t \in [0, \infty) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

is the Heaviside step function (unistep function). Translation of $h(t)$ is indicator function for half-open interval.

The constraint calculation stage shown in Figure 3 is to consider both $C_r^{request}(t)$ and system reservation state $L(t)$, so that the resulted $C_r(t)$ returns the maximum bandwidth that can be allocated to request r at time t:

$$C_r(t) = (C_r^{request} \; min \; L_1 \; min \; L_2 \; min \; \ldots \; min \; L_k)(t) \tag{7}$$

where we assume links $L_1, L_2, \ldots L_k$ form path from $source[r]$ to $dest[r]$, and $L_i(t)$ is the time-(remained bandwidth) function for link L_i. The *min* operation is illustrated in Figure 4 with two links L_1 and L_2 in request r's path:

Reservation decision function $D_r(t)$ returns the reserved data rate for r at time t. If scheduler rejects the request, no bandwidth will be reserved for the request in the whole time axis. Thus rejection decision can be represented by f^0. $D_r(t)$ satisfies:

$$D_r(t) \leq C_r(t) \tag{8}$$

$$\int_{t=a_r}^{d_r} D_r(t)dt = v_r, \text{ if } D_r \neq f^0 \tag{9}$$

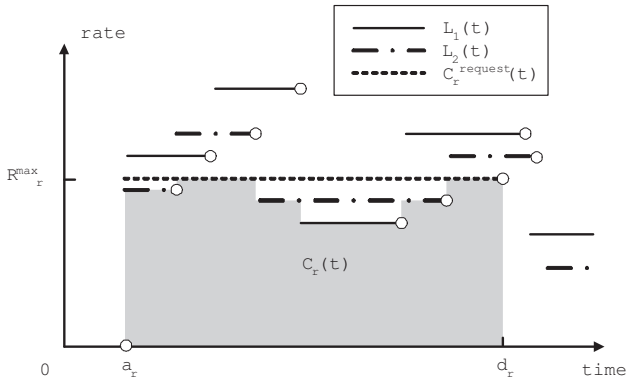


Figure 4: Calculate request's constraint function $C_r(t)$

In the system state update stage shown in Figure 3:

$$L_i(t) = (L_i - D_r)(t), \forall L_i \in \text{path of } r \tag{10}$$

At time τ, an empty link L_i without any reservation has $L_i(t) = B[L_i]h(t-\tau)$, where $B[L_i]$ is the total capacity of link L_i.

3.3 Step time-rate functions

General time-rate functions are not suitable for implementation, thus we restrict our discussion to a special class of time-rate functions, i.e., the *step time-rate functions*, which are easy to be stored and processed.

Formally, a function is called *step function* if it can be written as a finite linear combination of indicator functions of half-open intervals. Informally speaking, a step function is a piecewise constant function having only finitely many pieces. A time step function $f(t)$ can be represented as:

$$f(t) = a_1 h(t-b_1) + a_2 h(t-b_2) + \cdots + a_n h(t-b_n) \tag{11}$$

We denote the set of all step functions as $\mathcal{F}_s \subset \mathcal{F}$. A step function with n non-continuous points can be uniquely represented by a $2 \times n$ matrix $\begin{bmatrix} a_1 & \dots & a_n \\ b_1 & \dots & b_n \end{bmatrix}$ with elements in first row non-zero, and elements in second row strictly increasing. All step functions with n non-continuous points form *n-step function set* $\mathcal{F}_s^n \subset \mathcal{F}_s$. $\mathcal{F}_s^0 = \{f^0\}$. $\mathcal{F}_s^1 = \{\text{all translations of } h(t)\}$. All non-regressive linear combination of two different elements in $\mathcal{F}_s^1$ form $\mathcal{F}_s^2$. For $f^2 \in \mathcal{F}_s^2$, if $a_1 + a_2 = 0$, f^2 and f^0 encompass a rectangular in time-rate coordinate. All such f^2 form the *rectangular function set* $\mathcal{F}_{rec}$. We also define *general n-step function set* $\mathcal{G}^n = \mathcal{F}^0 \cup \mathcal{F}^1 \dots \mathcal{F}^n$.

Following discussions restrict reservation schemes to make decision in step function form, i.e., $D_r \in \mathcal{F}_s$. For $f^n(t) \in \mathcal{F}_s^n$ and $f^m(t) \in \mathcal{F}_s^m$, it is easy to show that $(f^n \; min \; f^m)(t) \in \mathcal{F}_s^{n+m}$, and $(f^n + f^m)(t) \in \mathcal{F}_s^{n+m}$, i.e., both *min* and *plus* operations are closed in $\mathcal{F}_s$, thus constraint function $C_r(t)$ and time-(remained bandwidth) function $L_i(t)$ are also step functions. The computation and space complexity for *min*, *plus* and *order* operations over function $f^n(t)$ and $f^m(t)$ are $O(n+m)$. We discuss calculation of $D_r(t)$ based on $C_r(t)$ and v_r in next section.

Schemes	accept decision	flexibility
FixTime-FixRate	$D_r(t) = C_r(t)$	0
FixTime-FlexRate	$D_r(t) \in \mathcal{F}_{rec}$ with term $h(t - a_r)$	1
FlexTime-FlexRate	$D_r(t) \in \mathcal{F}_{rec}$	2
Multi-Interval	$D_r(t) \in \mathcal{G}_n$	2n-2

Table 1: Reservation schemes

4 Reservation schemes

4.1 Schemes taxonomy and heuristics

Existing RSVP-type reservation schemes only supports reservation of a fixed bandwidth from a fixed start time, which we name as *FixTime-FixRate* schemes. Slightly more general are *FixTime-FlexRate* schemes, which still enforces a fixed start time, but allow scheduler to flexibly determine the reservation bandwidth. To further generalize the idea, we have *FlexTime-FlexRate* schemes, which allows reservation starts from any time in $[a_r, d_r]$ and reserves any rate (but need to be constant) continuously until transfer completes. Finally, by allowing reservation comprise of multiple ($n \leq 1$) sub-intervals with different reservation bandwidths, we have *Multi-Interval* schemes. Regarding their solution space, *FixTime-FixRate* $\subset$ *FixTime-FlexRate* $\subset$ *FlexTime-FlexRate* $\subset$ *MultiRate*. Their different flexibilities are summarized in Table 1.

The flexibility makes it hard to choose a suitable decision $D_r(t)$ if multiple candidates are available. As mentioned in Section 2, there are multiple performance criteria, increasing accept probability, minimizing flow time, and ensuring fairness among flows, just name a few. In fact, even for RSVP-type reservation scheme with only two choices (reject, or accept the request with fixed rate at fixed start time), it is hard to make an optimal selection as proved in [12]. Instead, we use simple heuristics to select representative scheme from each family for performance comparison. A *threshold-based rate-tuning* heuristic is used to choose candidate from *FixTime-FlexRate* schemes which will be detailed in Section 5. Simple Greedy-Accept and Minimize-FlowTime heuristics are used to choose candidate from *FlexTime-FlexRate* family and *Multi-Interval* family.

Greedy-Accept means: If there is at least one feasible solution to accept a coming request, the request should not be rejected. Greedily accept new request is not optimal in an offline sense, because sometimes it maybe better to *Early-Reject* a request even when feasible solution exists, so that capacity can be kept for more rewarded-requests which arrive later. Despite this, it is an interesting heuristic to study, because:

- *Greedy-Accept* heuristic can be used orthogonally with trunk reservation to mimic the behavior of *Early-Reject*;
- *Greedy-Accept* introduces a strict priority based on arriving order, which by itself is a reasonable assignment philosophy.

Minimize-FlowTime means: If there are multiple feasible solutions in the solution space, the one with minimal completion time will be chosen. Besides the straightforward benefit on minimizing flow time, this philosophy also helps maximize the utilization of resource in near

future, which otherwise is more likely to be wasted if no new request comes soon. However, since the near future is more densely packed with reservation, assuming all requests have identical R_{exp}, then a small volume request with short life span is easier to get rejected than a large volume request with long life span. This unfairness can also be addressed by volume-based trunk reservation.

4.2 *FixTime-FixRate* schemes

In *FixTime-FixRate* schemes, request specifies its desired reservation rate. Scheduler can only decide to accept or reject. As shown in [1], reducing reservation rate increases system's Erlang capacity. Thus a candidate *FixTime-FixRate* scheme to maximize accept rate is to enforce:

$$D_r = \begin{cases} R_r^{min}(h(t-a_r) - h(t-d_r)) & \text{if } R_r^{min} \leq C_r(a_r) \\ f^0 & \text{otherwise} \end{cases} \tag{12}$$

Here $R_r^{min} = \frac{v_r}{d_r - a_r}$ satisfy Equation (9). In this scheme, every accepted request completes transfer exactly at its deadline, if a dull transfer protocol is used. This is the reservation scheme used in Figure 1. Notice that for *FixTime* schemes without advance reservation, Equation (8) is simplified to consider constraint function $C_r(t)$'s value at a_r only, because:

- *FixTime* schemes' reservation is enforced to begin from a_r;
- Under *FixTime* schemes without advance researvation, time-(remained capacity) function $L_i(t)$ for any link L_i is non-decreasing along time axis.

4.3 *FixTime-FlexRate* schemes

FixTime-FlexRate schemes still enforce transfer start at a_r, thus $D_r(t) \in \mathcal{F}_{rec}$ must have term $h(t-a_r)$. Compared to *FixTime-FixRate* schemes, *FixTime-FlexRate* schemes can flexibly choose the rate parameter R_r in $D_r(t)$. *FixTime-FlexRate* schemes allocate a single rate R_r for accepted request r from its arrival time a_r to its completion time $a_r + \frac{v_r}{R_r}$:

$$D_r(t) = R_r(h(t-a_r) - h(t-a_r-\frac{v_r}{R_r})) \tag{13}$$

The second term above is calculated using Equation (9). While Equation (8) is simplified as: $a_r + \frac{v_r}{R_r} \leq d_r$ thus $R_r \geq \frac{v_r}{d_r - a_r}$, and $R_r \leq C_r(a_r)$ similar to *FixTime-FixRate* schemes.

4.4 *FlexTime-FlexRate* schemes

FlexTime-FlexRate schemes relax the fix start time constraint. Thus, *Decision Function* $D_r(t)$ of *FlexTime-FlexRate* schemes can be any rectangular function satisfying Equation (8) and (9). *FlexTime-FlexRate* schemes allocate a single rate R_r in interval $[t_r^{start}, t_r^{start} + \frac{v_r}{R_r}] \subseteq [a_r, d_r]$. The D_r can be fully characterizes by a pair (t_r^{start}, R_r). Completion time is calculated using Equation (9).

To simplify Equation (8), we define *constraint rectangular function set* $\mathcal{F}_{rec}^{constraint}$ and *Pareto optimal rectangular function set* $\mathcal{F}_{rec}^{Pareto}$ for constraint function $C_r(t)$:

$$\mathcal{F}_{rec}^{constraint} = \{f(t) | f(t) \in F_{rec} \text{ and } f(t) \leq C_r(t)\} \tag{14}$$

$$\mathcal{F}_{rec}^{Pareto} = \{f(t)|f(t) \in F_{rec}^{constraint} \text{and} \\ g(t) \in \mathcal{F}_{rec}^{constraint}, g(t) > f(t)\} \quad (15)$$

Pareto optimal rectangular function set of a n-step constraint function $C_r(t)$ can be calculated in $O(n^2)$ as illustrated in Figure 5, $\mathcal{F}_{rec}^{Pareto}$ contains $O(n^2)$ elements.

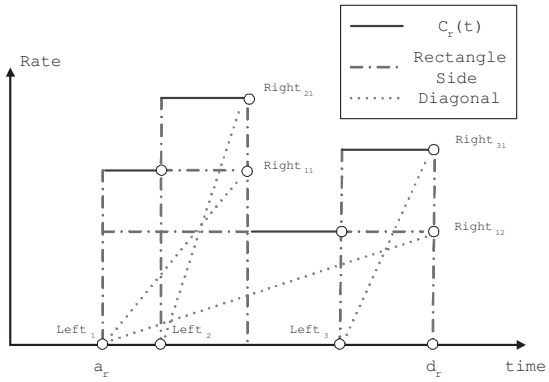


Figure 5: Pareto Optimal Rectangular function set

Apply *Greedy-Accept* and *Minimize-FlowTime* heuristics here: a request r is rejected, if and only if there is no $f(t) \in \mathcal{F}_{rec}^{Pareto}$ with integration no less than v_r; otherwise, all Pareto optimal rectanglar functions with large enough integration are checked to identify the one providing minimum flow time. Given a Pareto optimal rectangular function $f(t) = a_1(h-t_1) - a_1(h-t_2)$, the minimum flow time it can provide is $t_1 + \frac{v_r}{a_1}$. The implementation of this scheme is detailed in Table 2.

4.5 *Multi-Interval* schemes

Compared to all above schemes, reservation decision in *Multi-Interval* schemes can be composed of multiple intervals with different reservation rates. Note that *Multi-Interval* schemes are different from preemptive schemes. Although multiple rates can be used in *Multi-Interval* schemes, and flows are probably scheduled to transfer in two discontinuous intervals, this decision is determined at the moment the request arrives, and is not changed (preempted) after that.

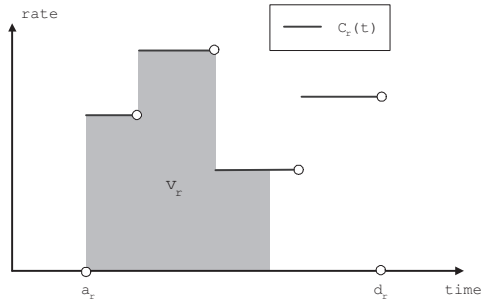


Figure 6: *Multi-Interval* schemes

Apply *Greedy-Accept* and *Minimize-FlowTime* heuristics here, if integration of $C_r(t)$ over

```
struct time-rate{
      double time;
      double rate;
      boolean unVisited = true;
};

Input: 6-tuple representation of request r and its constraint function C_r(t), which is
a n-step function represented by a time-rate vector v. For i∈[0,...,n−1]:
      v[i].time is the (i+1)^th noncontinuous points of C_r(t),
      v[i].rate = C_r(v[i].time).

Output: decision d in a time-rate structure.

int nextIncrease(int i){
      for(i++; i <= n; i++)
            if(v[i-1] < v[i])
                  break;
      return i;
}

int nextDecrease(int i){
      if(v[i].unVisited){
            v[i].unVisited = false;
            double r = v[i].rate;
            for(i++; i < n; i++)
                  if(r > v[i].rate)
                        break;
                  return i;
      }
      else
            return n;
}

struct time-rate reservation(request r, struct time-rate v[]){
      struct time-rate d;
      d.time = r.deadline;
      d.rate = 0;
      for(int left = 0; left < n-1 && v[left].rate > 0 && v[left].time < d.time; left =
nextIncrease(left)){
            double resv-rate = v[left].rate;
            for(int right = nextDecrease(left); right < n; right = nextDecrease(right)){
                  if(v[left].time + r.volume / resv-rate < d.time){
                        d.time = v[left].time + r.volume / resv-rate;
                        d.rate = resv-rate;
                        break;
                  }
                  resv-rate = v[right].rate;
            }
      }
      if(d.rate > 0) d.time − = r.volume / d.rate;
      return d;
}
```

Table 2: *Greedy-Accept Minimize-FlowTime FlexTime-FlexRate* schemes

time axis is larger than v_r:

$$D_r(t) = \begin{cases} C_r(t) & t \leq \tau \\ 0 & t > \tau \end{cases} \tag{16}$$

where time τ satisfies: $\int_{t=a_r}^{\tau} C_r(t)dt = v_r$. $D_r(t) = f^0$ if no such τ exists. As shown in Figure 6, when $C_r(t) \in \mathcal{F}_s^n$ is a n-step function, computational complexity of MR-MaxPack-MinDelay scheme is $O(n)$, and $D_r(t) \in \mathcal{G}_s^n$.

Sometimes it is useful to enforce $D_r(t) \in \mathcal{G}_s^n$ for a constant n. For example, *FlexTime-FlexRate* schemes are subset of *Multi-Interval* schemes enforcing $D_r(t) \in \mathcal{G}_s^2$. If reservation decision is allowed to be composed of at most two adjecent subintervals with different rates, it can be modeled as subset of *Multi-Interval* schemes enforcing $D_r(t) \in \mathcal{G}_s^3$.

5 Performance evaluation

5.1 Simulation setup

We use simulation to demonstrate the potential performance gain from the increasing flexibility. We consider the performance of both *blocking probability* and *mean flow time* for following schemes:

- *FixTime-FixRate-*R_{max} scheme is a *FixTime-FixRate* scheme with reservation rate of R_{max};
- *FixTime-FixRate-*R_{min} scheme is a *FixTime-FixRate* scheme with reservation rate of R_{min};
- *Threshold-FixTime-FlexRate* scheme is a simple *FixTime-FlexRate* scheme which reserves R_{max} when the minimum unreserved bandwidth among all links along the path is above a threshold (set as 20% of link capacity in this simulation), and reservates R_{min} otherwise;
- *Greedy-Accept* and *Minimize-FlowTime* heuristic in the *FlexTime-FlexRate* family;
- *Greedy-Accept* and *Minimize-FlowTime* heuristic in the *Multi-Interval* family.

For all above settings, *dull transport protocol* is assumed, which uses and only uses reserved bandwidth.

To simplify the discussion on the potential gain of increasing flexibility, we ideally assume that bulk data transfer requests arrive online according to a Poisson process with parameter λ, all requests have the same volume v, $R_{max} = C/10$ and $R_{min} = C/20$, where C is the link capacity. Observation in this simple setting also helps explain the system behavior in more general settings, which may have different arrival process, volume distribution, R_{max} and R_{min}.

5.2 Single Link setting

We first consider the case of single bottleneck link. Performance of above schemes is plotted under increasing load.

Figure 7 shows that in terms of blocking probability, *FixTime-FixRate-*R_{min} scheme performs better than *FixTime-FixRate-*R_{max} scheme. When reservation rate decreases, two

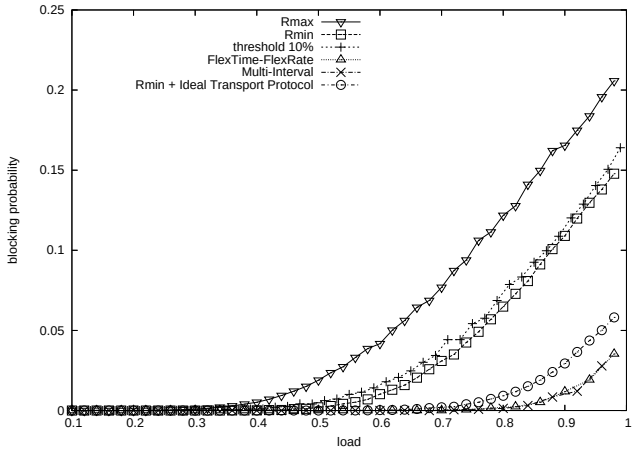


Figure 7: Blocking probability of reservation schemes

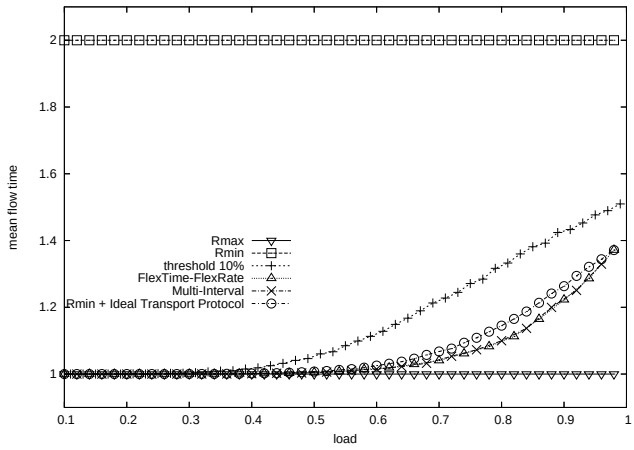


Figure 8: Mean flow time of reservation schemes

conflicting effects happen: On one hand, more requests can be accepted simultaneously; on the other hand, each request takes a longer time to finish. [1] shows that decreasing reservation rates increase system's Erlang capacity, which is verified in this Figure. However, as *FixTime-FixRate-*R_{min} always conservatively reserve R_{min}, its request flow time is always v_r/R_{min}. Contrarily, flow time of *FixTime-FixRate-*R_{min} scheme is aways v_r/R_{max}, which is only half of v_r/R_{min} under our simulation setting, as shown in Figure 8.

Exploiting the flexibility of selecting reservation rates, *Threshold-FixTime-FlexRate* scheme strikes a good balance between reducing blocking probability and minimizing mean flow time. When load is low, a new request reserves full rate R_{max}, so that its flow time is minimized. Although the new request agressively seizes bandwidth, the threshold statistically ensures that there are still abundant bandwidth left. Thus the probability is low that in a near future coming flows are blocked due to this aggressive request. Instead, the new request exploits the resource which will otherwise be wasted, and also it is able to release network resource more quickly, which benefits the system at a middle-range time scale. In the lightly-loaded region *Threshold-FixTime-FlexRate* scheme performs similar to *FixTime-*

FixRate-R_{max} scheme. However when load increases, links are often run in saturated state, a new request has higher probability to find remained capacity below threshold. Thus in this region, *Threshold-FixTime-FlexRate* scheme automatically adapts its behavior to perform similar to *FixTime-FixRate-R_{min}*. From the two figures, it is observed that *Threshold-FixTime-FlexRate* scheme has a much lower blocking probability than *FixTime-FixRate-R_{max}* scheme, while has a much lower mean flow time than *FixTime-FixRate-R_{min}* scheme.

In this single link setting, behavior of selected *FlexTime-FlexRate* and *Multi-Interval* schemes are identical. This is an artificial result of the uniform volume and R_{max} setting, as well as the integer value of C/R_{max}. We also conduct extensive simulations over more general volume, R_{max} and R_{min} distribution over a single link, and results also show that the performance of *FlexTime-FlexRate* and *Multi-Interval* remains close. Both *FlexTime-FlexRate* and *Multi-Interval* schemes perform much better than above three schemes in both blocking rate and flow time.

A remarkable observation is that, *FlexTime-FlexRate* and *Multi-Interval* schemes with dull transport protocol even outperform the *FixTime-FixRate-R_{min}* scheme equipped with ideal transport protocol, in terms of both blocking rate and flow time (see the *Rmin + Ideal Transport Protocol* curve in both Figure. In addition, the small flow time of *Rmin + Ideal Transport Protocol* is achieved opportunistically by ideal transport protocol, which can not be guaranteed at the moment when the reservation is made (in contrast, *FixTime-FixRate-R_{min}* scheme can only guarantee that accepted requests are completed before deadline). Thus other co-allocated resources can not exploit the small flow time to increase their scheduling efficiency. On the other hand, the request flow time is known and guaranteed in reservation schemes at the moment when request is processed. This predictability can benefit other co-allocated resources. This result strongly motivates the study of advanced reservation schemes.

5.3 Grid network setting

We also evaluate different schemes' performance in a network setting. We use the topology as shown in Figure 9. n ingress sites and n egress sites are interconnected by over-provisioned core networks. Each site composed of a cluster of grid nodes, and is connected to core network with a link of capacity C. The maximal aggregate bandwidth demands from the culster may exceed C, making these links potential bottlenecks. For simplicity, we assume that the core network is over-provisioned, like the visioned Grid5000 networks in France [5]. Core network can be provisioned, for example, using hose model [6]. When generating request, its source is randomly selected from ingress sites, then a random destination is selected independently among egress sites. All sites have the same probability to be chosen.

Figure 10 and Figure 11 plot the performance when there are 10 ingress nodes and 10 egress nodes in the network. Compared to Figure 7 and Figure 8, three phenomenons are observed:

- Overall, performance of schemes degrades slightly;
- *FlexTime-FlexRate* scheme's blocking probability shows a big increase, and its performance is no longer close to Multi-Interval scheme;
- *Multi-Interval* scheme's mean flow time performance deteriorates obviously.

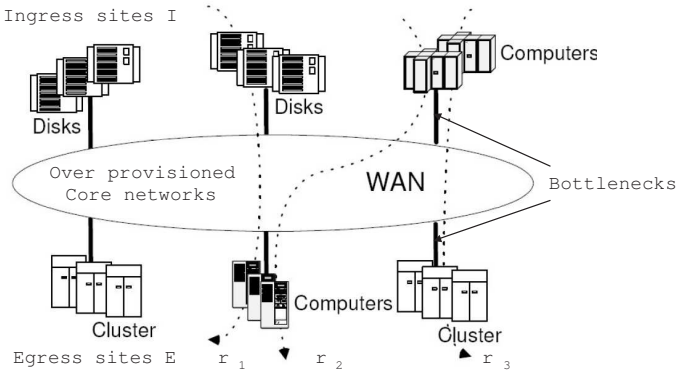


Figure 9: Topology

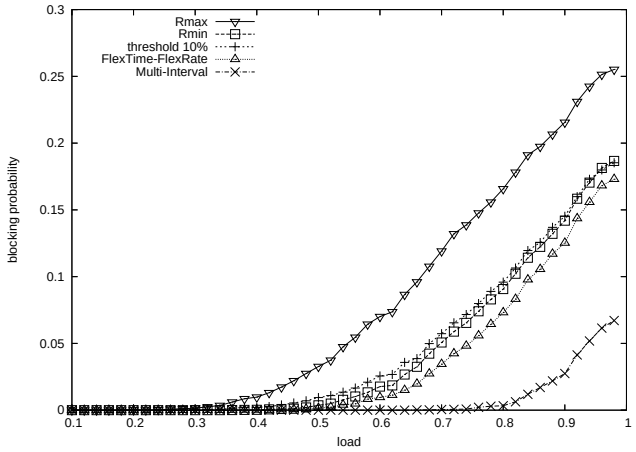


Figure 10: Blocking probability of reservation schemes

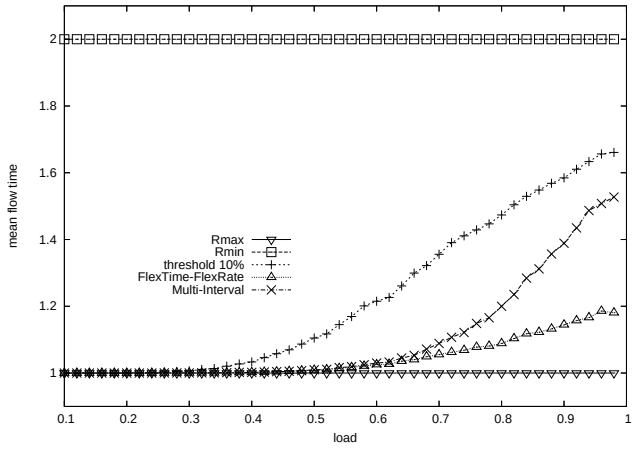


Figure 11: Mean flow time of reservation schemes

The overall performance degration can be traced to the fact that reservation in a network need to consider multiple links (both ingress and egress link in this topology). A reservation

request is blocked or its flow time becomes longer when any one of them is congested. If we assume that congestion states in two links are independently and identically distributed, with mean congestion probability p, the probability that there is at least one of them being congested is $2p - p^2 > p$. This intuitively explains the overall degration of performance.

The performance degration of *FlexTime-FlexRate* scheme's blocking probability and *Multi-Interval* scheme's mean flow time can be explained using a simple example in Figure 12.

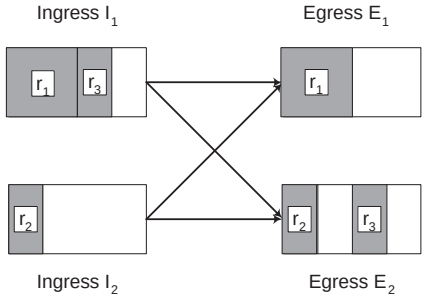


Figure 12: A fragmentation example

In this example, there are two ingress links and two egress links interconnected by over-provisioned core networks. Existing request r_1 reserves bandwidth in I_1 and E_1, while existing request r_2 reserves bandwidth in I_2 and E_2 as shown in the Figure. At current system time, a new request r_3 arrives at I_1 with destination E_2. For the three *FixTime* schemes (*FixTime-FixRate-*R_{max} scheme, *FixTime-FixRate-*R_{min} scheme and *Threshold-FixTime-FlexRate* scheme), they are not allowed to accept r_3 since bandwidth is fully reserved for the current time. This prevents fragmentation as shown in the Figure when both *FlexTime-FlexRate* scheme and *Multi-Interval* scheme exploit their flexibility to accept r_3. This time-axis framentation increases *FlexTime-FlexRate* scheme's blocking probability, since *FlexTime-FlexRate* scheme can only allocate a continuous time interval. On the other hand, blocking rate of *Multi-Interval* scheme is not affected as much as *FlexTime-FlexRate* scheme because *Multi-Interval* scheme can make use of multiple (discontinuous) intervals. However *Multi-Interval* scheme's mean flow time is affected.

In above examples, *Multi-Interval* schemes often give the best perfromance. However, using multiple intervals comes at a cost. Figure 13 shows the increase trend of sub-interval number when network size is increased. It is shown that this number becomes quite stable around a small level, when the number of nodes grows larger than the multiplexing level of a single link, which is C/R_{max}. This result holds for different load levels. This observation shows the feasibility of exploiting *Multi-Interval* scheme.

6 System architecture

The logic framework shown in Figure 3 corresponds to a centralized scheduler, which may not be desirable because:

- links may be under control of different authorities;
- when network size grows, the centralized scheduler itself may become a bottleneck;
- Centralized scheduler presents an one-failure-point.

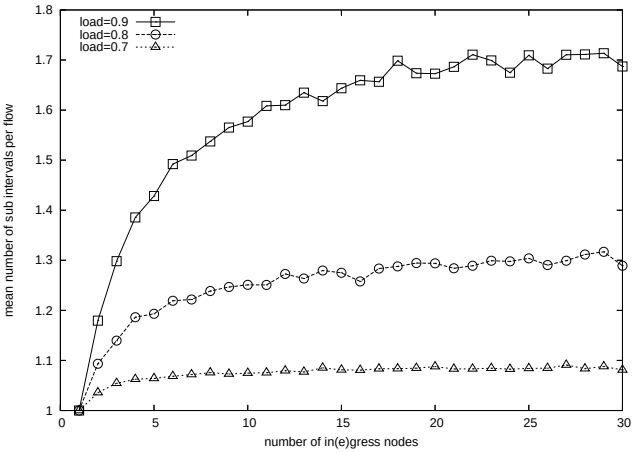


Figure 13: Mean number of intervals per flow in *Multi-Interval* scheme

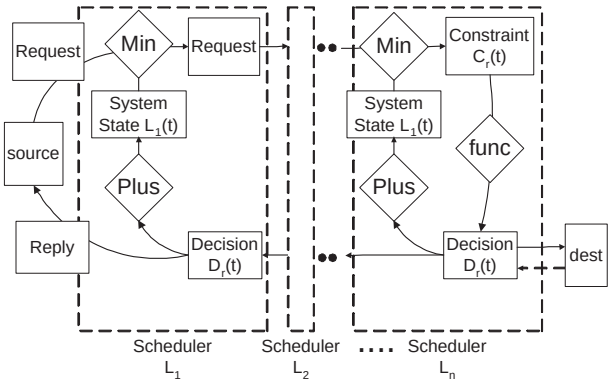


Figure 14: Distributed architecture

Thus we present a simple distributed architecutre as shown in Figure 14. In this architecture, every bottleneck link is associated with a local bandwidth scheduler, which maintains the local reservation state. Request generated from the source first arrives at link L_1, whose scheduler uses *min* operation to combine its local link state constraint into the request specification. The updated request specification is forwards to the nexthop. In this way, the constraint function is updated hop by hop: $C_r^i(t) = (L_i \; min \; C_r^{i-1})(t)$. When request reaches the last hop L_n, the constraint function $C_r(t)$ is completely constructed, and the scheduler in L_n makes decision $D_r(t)$ based on $C_r(t)$. $D_r(t)$ is sent to destination, which may issue a confirmation. $D_r(t)$ is then sent through the same path back to source. $D_r(t)$ is kept unchanged along the path, and each hop uses $D_r(t)$ to update its local reservation state L_n.

Single out a local scheduler, its logic can still be interpreted using the logic framework of Figure 3. The only difference is that for schedulers not in the last hop, their *"func"* operation is not a local operation but depends recursively on the next hop.

7 Related works

Admission control and bandwidth reservation have been studied extensively in multimedia networking. A real-time flow normally requests a specified value of bandwidth. Existing reservation schemes such as RSVP [3] attempt to reserve the specified bandwidth immediately when request arrives. Reservation remains in effect for an indefinite duration until explicit "Teardown" signal is issued or soft state expires. No time-indexed reservation state is kept.

Time-indexed reservation is needed when considering advance reservation of bandwidth [15], which allows requesting bandwidth before actual transfer is ready to happen. For example, a scheduled tele-conference may reserve bandwidth for a specified future time interval. [4] shows that advance reservation causes bandwidth fragmentation in time axis, which may significantly reduce accept probability of requests arriving later. To address the problem, they propose the concept of *malleable reservation*, which defines advance reservation request with flexible start time and rate.

Optimal control and their complexity is studied for different levels of flexibility. [2] studies call admission control in a resource-sharing system, i.e. how to use the reject flexibility regarding different classes of traffic. Optimal policy structure is identified for some special case. [12] proved that in a network with multiple ingress and egress sites, *off-line* optimization of accept rate for uniform-volume uniform-rate requests with randomly specified life span is NP-complete. They also consider flexible tuning of reservation rate. [1] studies the increase of Erlang capacity of a system by decreasing the service rate. In its essential, such service rate scaling is identical to the capacity scaling, which is studied by [10] and [9] to approximate large loss networks.

There is also a large literature of online job scheduling with deadline, for example, [8], [11], [14]. A job monopolizes processor for the time it's being scheduled, which maps exactly to packet level scheduling, while in flow level, we must consider multiple flows share bandwidth concurrently, as represented by R_{max}.

8 Conclusion

In this paper, we study the bandwidth reservation problem for bulk data transfers in grid networks. We model grid networks as multiple sites interconnected by wide area networks with potential bottlenecks. Data transfer requests arrive online with specified *volumes* and *deadlines*, which allow more flexibility in reservation schemes design. We formalize a general non-preemptive reservation framework, and use simulation to examine the impact of feasibility over performance. We also propose a simple distributed architecture for the given framework. The increased flexibility can potentially improve system performance, but the enlarged design flexibility also raises new challenges to identify appropriate reservation schemes inside the solution space.

References

[1] E. Altman. Capacity of multi-service cellular networks with transmission-rate control: A queueing analysis. In *MOBICOM*, September 2002.

[2] E. Altman, T. Jimenez, and G. Koole. On optimal call admission control in a resource-sharing system. *IEEE Transactions on Communications*, 49(9):1659–1668, September 2001.

[3] R. Braden, L. Zhang, S. Berson, S. Herzog, and S.Jamin. Resource reservation protocol (rsvp), September 1997.

[4] L. Burchard, H. Heiss, and C. De Rose. Performance issues of bandwidth reservations for grid computing. In *CAHPC*, pages 82–90, November 2003.

[5] F. Cappello, F. Desprez, M. Dayde, E. Jeannot, Y. Jegou, S. Lanteri, N. Melab, R. namyst, P. Primet, O. Richard, E. Caron, J. Leduc, and G. Mornet. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In *the 6th IEEE/ACM International Workshop on Grid Computing, Grid'2005*, November 2005.

[6] N. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. Ramakrishnan, and J. van der Merwe. A flexible model for resource management in virtual private networks. In *SIGCOMM*, pages 95–108, 1999.

[7] I. Foster. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[8] M. Goldwasser. Patience is a virtue: the effect of slack on competitiveness for admission control. In *SODA*, pages 396–405, January 1999.

[9] P. Hunt and T. Kurtz. Large loss systems. *Stochastic Processes and their Applications*, 53:363–378, October 1994.

[10] F. Kelly. Loss networks. *The Annals of Applied Probability*, 1(3):319–378, 1991.

[11] F. Li, J. Sethuraman, and C. Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *SODA*, pages 801–802, January 2005.

[12] L. Marchal, P Vicat-Blanc Primet, Y. Robert, and J. Zeng. Optimizing network resource sharing in grids. In *Globecom*, volume 2, pages 835–840. IEEE Computer Society Press, November 2005.

[13] J. Roberts. A survey on statistical bandwidth sharing. *Computer Networks*, 45(3):319–332, June 2004.

[14] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Jnl on Computing*, 12:294–299, May 1983.

[15] D. Wischik and A. Greenberg. Admission control for booking ahead shared resources. In *INFOCOM*, pages 873–882, April 1998.