Using an Event Based Priority Queue for Reliable and Opportunistic Scheduling of Bulk Data Transfers in Grid Networks

Kashif Munir[*], Somera Javed[**], Michael Welzl[*], Malik Muhammad Junaid[*]
[*] Institute of Computer Science, University of Innsbruck, Austria
{Kashif.Munir, Michael.Welzl, Muhammad.Malik}@uibk.ac.at
[**] National University of Computer and Emerging Sciences, Islamabad, Pakistan
somera11@gmail.com

Abstract-**This paper proposes an opportunistic, reliable and realistic QoS mechanism for bulk data transfers in Grids, which maximizes acceptance rate and network resource utilization by using an event based priority queue. The metrics that have been studied in the evaluation of elastic scheduling heuristics are the acceptance percentage and the mean flow time of requests. Some of the existing heuristics for the scheduling of bandwidth requests do not include the communication and computation delays and communication overheads, which are involved in the reliable transfers of such reservations. We have compared our approach with only those existing heuristics which are reliable and include all overheads. We demonstrate the performance improvement by documenting results of simulations.**

I. INTRODUCTION

Grid computing enables the virtualization of distributed computing and data resources such as processing, storage capacity and network bandwidth to provide a user with a unified view of the system.

The Grid community needs the network to be available and to provide a desired level of service at any time. In general, there are two types of network resource reservations in computer networks. One is immediate reservation, which is made in a just-in-time manner, and the other is advance reservation, which allows reserving network resources a long time before they are actually used. Advance reservations are mainly useful for Grid computing but can also be used for applications like content distribution networks that require network QoS. These reservations have properties which make them somewhat different from the classical per-flow guarantees that have been demanded for multimedia services – the service may not be used immediately after its reservation and the flows are elastic.

In order to guarantee fine-grain QoS, traffic within the protected aggregate must be controlled – but, rather than involving routers, this can be done at the end systems by communicating with a Resource Broker (a common service in Grids where one can, for instance, request a machine with a certain CPU power; our intention is to extend this element with the ability to grant Advance Network Reservation).

Our mechanism satisfy these needs in a reliable and realistic manner; as the residual network capacity is quickly and fairly shared by all existing flows, completion times are minimized and acceptance of reservation requests are maximized. Other than all the other existing work, where a loss-free network without any admission control related overhead is assumed, our mechanism works with realistic network conditions, taking possible packet loss as well as communication and computation delays into account.

Common admission control schemes assume all flows to use a certain fixed rate. It is a key feature of our mechanism that it manages to efficiently utilize network resources in a scalable manner because flows opportunistically increase their rates as bandwidth becomes available. This is attained by using high-speed congestion control for all end-to-end flows; we use a mechanism that is designed for high-speed networks (networks with a large bandwidth-delay product), where standard TCP congestion control is known not to yield satisfactory performance. Because it is designed for high speeds and particularly convenient in a Grid setting, we chose UDT [1], but stress that any max-min fair congestion control scheme could be used in its place.

The literature will be surveyed in the next section. Our reliable scheduling mechanism is described in Section 3, and we support our explanations with simulation results in Section 4. Section 5 concludes.

II. RELATED WORK

Early work on advance reservation focused on reservation protocols like RSVP [2] and routing algorithms for networks with advance reservations [3]. Grid applications need guarantees of Quality of Service (QoS) [4,5]. Targeting deadline support for bulk data transfers, the problem of network resource reservation [6] has been proposed to be studied within the grid scope. An example for a Grid toolkit that supports such mechanisms is Globus with its GARA resource allocation component [7]. The issue of bandwidth fragmentation is discussed by Burchard et al. [8]. Bandwidth fragmentation may reduce the acceptance percentage of

The work described in this paper is partially supported by the Higher Education Commission (HEC) of Pakistan under the doctoral fellowship program for Austria and by the Tyrolean Science Fund.

requests arriving later. The authors [8] propose the idea of malleable reservation to address the problem for which a start time and single rate value can be selected from a range of values.

For malleable requests in [10], the method of [8] or [9] is used to adjust the bandwidth or duration to satisfy the requester. However for a fixed request in [10], the only way to avoid being rejected is to adjust the bandwidth of admitted malleable requests. The trouble with this scheme is the extra overhead in finding and adjusting the admitted reservations which may be modified. The Multi-Interval scheme, presented in [11], avoids this trouble. The scheme is based on the concept that a request should not be rejected if there is at least one feasible solution to accept it. If there are multiple solutions, the one which yields the minimum flow time is chosen and is not changed after that.

In [12] a general view of network resources sharing in Grids and Grids traffic isolation is presented. Optimization of bandwidth sharing among Grid flows is given [13] by manipulating the transmission windows of the flexible requests between minimum and maximum rates. The formulated optimization problem is proven to be NP-complete.

Two types of strategies for scheduling bulk data transfers are possible [14]. One strategy is to immediately grant or reject admission to a reservation request on its arrival time. In the other strategy, if a reservation request can not be granted or rejected at the time of its arrival, it is put in a queue to explore its possible admission later. Our mechanism, which will be explained in section 3, is based on the latter strategy. In our mechanism, we have used an event based waiting queue. The event is termination of a request in the network.

A time-slot based approach for scheduling the elastic and streaming requests is described in [15]. However, the effect of the extra signaling overhead, which is due to the manipulation of the data transfer rates of individual flows, is not taken into account in this approach.

III. Reliable Scheduling Mechanisms

Our general method of applying fully distributed congestion control based admission control was introduced together with earlier scheduling approaches in [16,17]; for ease of understanding, we will briefly recapitulate this prior work in the next two subsections (please see [16] and [17], respectively, for in-depth explanations of these schemes).

Then, we will present our new mechanism, where we enhance this prior work with an event based priority queue.

A. ARR_CC

This heuristic [16] is based on the on-arrival decision about acceptance or rejection of a new request. The sender contacts the RB with a new reservation request by sending its data size, duration of transfer, start time and type of reservation. There are two types of reservations. One is Immediate Reservation (IR) and the other is Advance Reservation (AR). We use the terms mechanism and heuristic interchangeably.

On receiving a new request, the RB first calculates the deadline of the request based on its reservation type. As already mentioned, the RB stores the information about all existing requests and the total capacity of the bottleneck link. The RB calculates the Average Required Rate (ARR) of the new reservation request and then it adds the ARRs of all existing flows which overlap with the new request. The purpose is to check whether the ARR of the new request can be satisfied so that it can meet its deadline. An acceptance or rejection message is then sent to the sender. On acceptance the sender starts sending data at its start time according to a max-min fair congestion protocol. The max-min fairness characteristic of the congestion control protocol makes it possible for the flows to fairly share the residual network capacity. In the absence of max-min fairness the unfair sharing may prolong some of the admitted flows while quickly completing some other flows. In any case an admitted flow meets its deadline.

On completion the sender sends a termination message to the RB. The RB then removes its entry from its internal repository and relinquishes the network resources that were previously owned by the completed request.

B. ARR_Adjustable_CC

Like ARR-CC this heuristic [17] is also based on the on-arrival decision about acceptance or rejection of a new request. It takes advantage of extra signaling and tries to adjust a new reservation request even when the ARR of a request is not available. One of the ways to do this is to reduce the rates of already existed flow below their originally agreed ARRs in such a way that the existing flows still meet their deadlines. This can be done by knowing how much data and duration of each existing flow is left.

When the ARR of a new request is not being fulfilled, the RB does not immediately send rejection message. Instead the RB sends a message to all started flows, asking how much data of each flow is left to be transferred. The senders reply to the RB with their remaining data size and they reduce their rates to their ARRs. The RB already knows the start time and duration of already started flows and calculate the new ARRs of each flow by dividing the remaining data size by the remaining duration of transfer. The RB then checks the admission of the new request by seeing whether there is enough available capacity to guarantee the ARR of the new request. If the required capacity is available, request is accepted, otherwise it is rejected. On acceptance, the sender starts sending data at its specified start time according to a congestion control protocol. It informs RB when it completes in the same way as mentioned in ARR_CC. This heuristic is expected to increase the acceptance percentage of the flows in the network but this expected increase is at the cost of some extra signaling overhead.

C. ARR_CC_Event_Queue

This is our new approach for scheduling bulk data transfers in Grids. The Resource Broker (RB) is the key component of the mechanism. The admission and termination of a flow is controlled through the RB residing

on a node in the network and by having a Sender – RB Signaling Mechanism.

The RB maintains the current state of network (i.e. all information about existing flows, bottleneck links and their capacities). After the admission of a flow with its ARR, the transfer rate starts to increase according to UDT's congestion control mechanism. Upon completing the transfer of a flow, the sender sends a termination message to the RB. We use the terms "flow" and "request" interchangeably.

This new heuristic is based on the strategy that in case a request is not accepted at the time of its arrival, it is added in a waiting queue to explore its possible admission later. The waiting requests in the queue are only checked in the event of termination of an existing request in the network. The periodic checking is however applied to filter out unwanted flows (those flows which are impossible to be admitted). Requests are rejected and removed from the waiting queue if their deadline becomes less than or equal to the current time or the ARR becomes greater than the total capacity of the bottleneck link, Upon removal of a request, a rejection message is sent to the sender. The RB maintains a priority queue in which checking admission of a flow is prioritized on the basis of its arrival time. The flow in the queue which was the earliest to arrive is checked first before checking others.

On the termination of a request, all waiting requests are checked. Any request whose ARR can be met is granted admission and is removed from the waiting queue. On acceptance, the sender starts sending data at its specified start time according to a congestion control protocol. It informs the RB when it completes in the same way as mentioned in ARR_CC. There is no extra signaling overhead in this heuristic.

Fig. 1 shows the categorization of the heuristics on the basis of their acceptance and rejection policies for reservation requests.

Admission control is a central element of our mechanism. We require no prioritization and therefore no state in routers; instead, we use a high-speed, stable and fair end-to-end congestion control mechanism. Admission control in our mechanism is different from the traditional Bandwidth Broker in DiffServ as it provides per-flow end-to-end guarantees to a Grid flow keeping in view its deadline constraints. Also the RB, which is used for admission control does bandwidth monitoring besides bandwidth brokerage, which means that a flow can be dynamically admitted at any time depending on the current availability of resources as well as its own deadline and the ARR.

The Resource Broker, installed as a separate server on any node in the Grid network, is a software instance used to manage resource reservation requests and to control the access to the Grid for each flow. Note that we only assume a single node for the sake of simplicity; our architecture is scalable, as distributing the Resource Broker with a scheme as in [18] would not change anything about it.

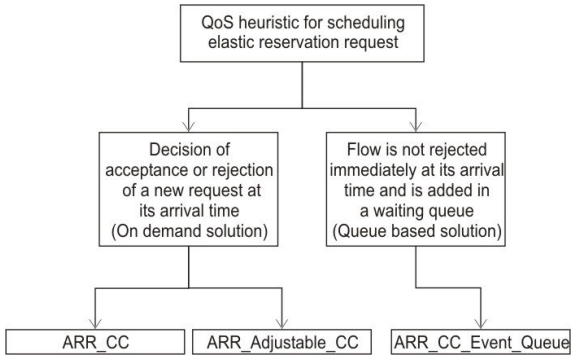

Fig. 1. Categorization of the heuristics

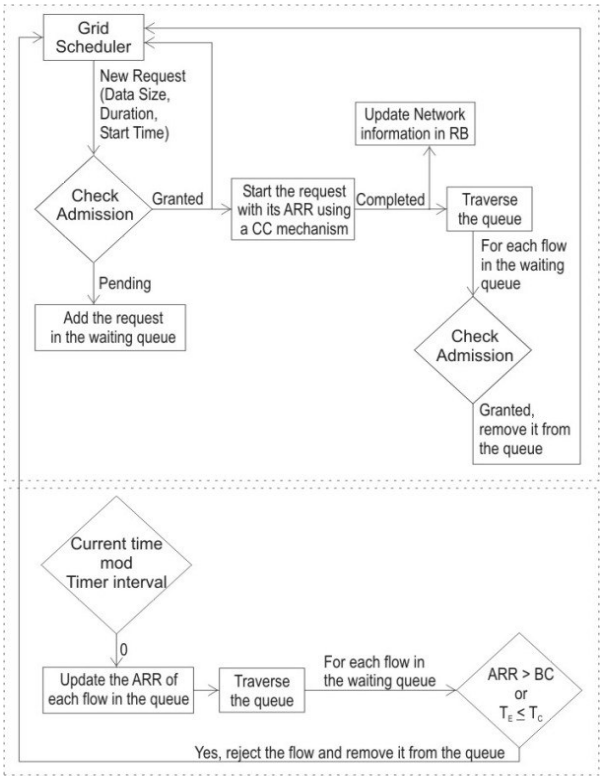

Fig. 2. Flow chart of the QoS mechanism.
In the figure, CC stands for Congestion Control, BC is the Bottleneck Capacity; T_E is the deadline of a request and T_C is the current time.

The flow chart of the QoS mechanism is shown in Fig. 2.

The Admission Control Algorithm of for the RB is given below.

D_S, D_F, T_S, ARR, ID, R_T: data size, duration, start time, average required rate, ID and reservation type of the request/flow for which a reservation is requested
$R_T \in \{IR, AR\}$: IR = immediate reservation and AR = advance reservation
Record of a request/flow: $\{R_T, T_S, T_E, D_F, D_S, ARR, ID\}$
***Φ**: Set of records of the currently accepted requests sharing the bottleneck link*

ψ: (Event based waiting queue) Set of those requests which do not get acceptance right at the time of their arrivals. The admission of requests from this set is based on a greedy acceptance policy i.e., whenever a flow terminates, any request in the waiting queue whose deadline can be met is granted admission however the checking of admission of the requests is prioritized on the basis of their arrival times.
C_T: The total capacity of the bottleneck link
T_C: The current time
T_i: The timer interval

```
Procedure ARR_CC_Event_Queue(Network_Topology_Information)

While (All requests are processed)

   If (a new reservation is requested)
         ARR = ⌈D_S/D_F⌉
         ID = {generate ID for the new request}

         If (Admission(Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C,C_T) = YES) Then
         {Accept the request by sending an acceptance message
         to the sender and start the flow with its ARR at its
         start time using a max-min fair Congestion Control
         Protocol}

         Else
               ψ = ψ + flow
               // insert the request in the waiting queue
               // flow = flow_record(R_T,T_S,T_E,D_F,D_S,ARR,ID)
         End If

   End If

   If (a served request is completed) Then
         Termination(Φ,ID)

         For each flow ∈ ψ
               ID = ψ(flow).ID

               If (ψ(flow).T_S ≤ T_C)
                  D_F = ψ(flow).T_E - T_C
                  T_S = T_C
               Else
                  D_F = ψ(flow).D_F
               End If

               ARR = ⌈ψ(flow).D_S/D_F⌉
               R_T = ψ(flow).R_T
               D_S = ψ(flow).D_S

               If Admission(Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C,C_T)= YES)
               Then
               {Accept the request by sending an acceptance
               message to the sender and start the flow with
               its ARR at its start time using a max-min fair
               Congestion Control Protocol}
                  ψ = ψ - flow
               End If

         End For

   End If

End While

End Procedure


Procedure Admission (Φ,ID,ARR,R_T,D_F,D_S,T_S,T_C,C_T)

Set C_R to 0
// C_R is the reserved capacity

If (R_T = IR) Then
         T_S = T_C
         T_E = T_C + D_F
         // T_E is the end time of a flow
Else
         // R_T = AR; this is an Advance Reservation
         T_E = T_S + D_F
End If

For each flow ∈ Φ
         If ((Φ(flow).T_S < T_E) AND (Φ(flow).T_E > T_S)) Then
                  C_R = C_R + Φ(flow).ARR
End For


If (C_T - C_R) > ARR Then
         Φ = Φ + flow     // admit the flow
         // flow = flow_record(R_T,T_S,T_E,D_F,D_S,ARR,ID)
         Return "Yes"
Else
         Return "No"
End If

End Procedure


Procedure Termination (Φ,ID)

For each flow ∈ Φ
         If (Φ(flow).ID = ID) Then
                  Φ = Φ  - flow
                  Break
End For

End Procedure


Procedure Periodic_checking_of_the_waiting_queue(ψ,Φ)

If (T_C % T_i = 0)
   For each flow ∈ ψ

         // remove flows which cannot be admitted
         If (ψ(flow).T_E ≤ T_C)
               ψ = ψ - flow
               // Reject the request by sending a
               // rejection message to the sender

         End If

         If (ψ(flow).T_S ≤ T_C)
               ψ(flow).T_S = T_C
               ψ(flow).D_F = ψ(flow).D_F - T_i
               ψ(flow).ARR = ⌈ψ(flow).D_S/ψ(flow).D_F⌉

               If (ψ(flow).ARR > C_T)
                     ψ = ψ - flow
                     // Reject the request by sending a
                     // rejection message to the sender
               End If

         End If

   End For

End If

End Procedure
```

In the simulations the FTP application protocol is used over the UDT high-speed data transfer protocol.

IV. Performance Evaluation

A single bottleneck link dumbbell network configuration is used for the simulations using ns-2. The bottleneck capacity is 1 Gbps and the bottleneck delay is set to 50ms. Drop Tail routers are used. The buffer size of the bottleneck link is set to 100% of the Bandwidth-Delay product. The packet size is set to 1500 bytes. The capacity of side links is 10 Gbps and the delay of each side link is set to 2ms.

We have compared the results of our mechanism with ARR_CC and ARR_Adjustable_CC mechanisms.

In one experiment, 5 sets of 10 simulations are performed.

In each simulation 100 flows are run. Within each set the size of all flows is the same, however the size of a flow varies from 500 MB to 1500 MB from one set to the other. The mean inter arrival time of flows is 4 seconds and the transfer duration of each flow is 50 seconds. For each simulation within a set, there are mixed types of randomly generated reservation requests, immediate and advance reservations. In each simulation the arrival time of a new reservation is also randomly chosen in each 4 seconds interval. The start time of an advance reservation request is also selected randomly in the interval [25,75] relative to the time of the arrival of the flow. The results of this experiment are shown in figures 3 and 4. Each point in these figures represents an average of the results of 10 simulations of a set. The standard deviation of the results of all sets is less than 2, which is quite small as compared to the possible range of results.

Fig. 3 shows that the acceptance percentage of ARR_CC_Event_Queue is higher than that of ARR_CC and ARR_Adjsutable_CC in all cases as we increase the load on the network by increasing the data size of the flows while keeping the mean inter arrival time between flows constant. This is due to the reason that ARR_CC_Event_Queue does not reject a request on its arrival time and explore the chances of its possible admission later by adding it in a waiting queue, which increases the admission percentage of the requests.

Fig. 4 shows that the mean flow time in ARR_CC_Event_Queue is higher than that of ARR_CC. This is due to the extra delay from queuing. The curve of the mean flow time in ARR_CC_Event_Queue fluctuates around the curve of the mean flow time in ARR_Adjustable_CC. This is due to the fact that in both mechanisms, a new request is adjusted either by decreasing the rates of the existing flows in the case of ARR_Adjustable_CC, or by admitting a request later from the queue in the case of ARR_CC_Event_Queue, due to which the existing flows take longer to complete. This consequently increases the mean flow time in both mechanisms.

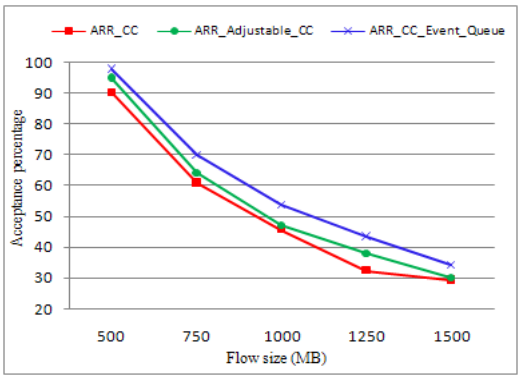

Fig. 3. Acceptance percentage of flows. The load on the network is increased by increasing the data size of each flow.

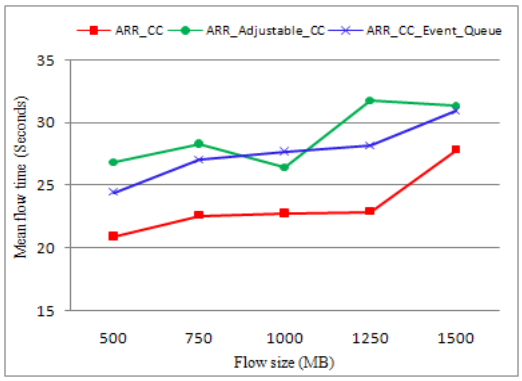

Fig. 4. Mean flow time of flows. The load on the network is increased by increasing the data size of each flow.

In another experiment, 3 sets of 10 simulations are performed with 100 flows. Other than the flow size and the mean inter arrival time of flows, all other simulation parameters are the same as in the previous experiment. In this experiment the flow size remains the same in each simulation of all sets, however the mean inter arrival time of flows varies from 2 seconds to 6 seconds. The results of the experiment are shown in figures 5 and 6. Each point in these figures represents an average of the results of 10 simulations of a set. The standard deviation of the results of all sets is again less than 2 which is quite small as compared to the possible range of results.

Fig. 5 shows that the acceptance percentage of ARR_CC_Event_Queue is higher than that of ARR_CC and ARR_Adjsutable_CC in all cases as we decrease the load on the network by increasing the mean inter arrival time and keeping the flow size constant. This has the same reason as mentioned above for the results in figure 3. Fig. 6 shows that the mean flow time in ARR_CC_Event_Queue is higher than the mean flow times of ARR_CC and ARR_Adjsutable_CC.

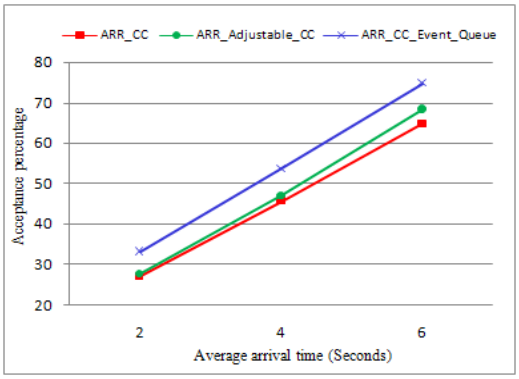

Fig. 5. Acceptance percentage of flows. The load on the network is decreased by increasing the mean inter arrival time.

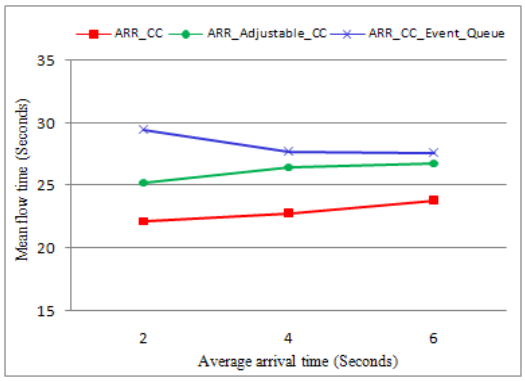

Fig. 6. Mean flow time of flows. The load on the network is decreased by increasing the mean inter arrival time.

V. Conclusion And Future Work

Our results show that, by using a queue for exploring admission of those flows which do not get admission on their arrival, the acceptance percentage of flows in the network can be increased. The heuristic, ARR_CC_Event_Queue, gives higher admission percentage of flows as compared to the earlier reliable approaches for bulk data transfers. As we have seen, this benefit is sometimes but not always obtained at the cost of slightly increased mean flow times. However, since the main goal of our system is to admit as many flows as possible while keeping all their deadlines, we consider this disadvantage to be of minor relevance. Our contribution is that we have shown the design and the implementation of an event queue based approach which is reliable and realistic, taking computation and communication overheads and delays into account.

References

[1] Y. Gu and R. Grossman, "UDT UDP-based data transfer for high-speed wide area networks," *Computer Networks, special issue on Hot topics in transport protocols for very fast and very long distance networks*, 2007.

[2] A. Schill, F. Breiter, and S. Kuhn, "Design and Evaluation of an Advance Reservation Protocol on Top of RSVP," *In IFIP 4th International Conference on Broadband Communications (BC '98)*, Stuttgart, Germany, IFIP Conference Proceedings 121, Chapman & Hall, pp. 23–40, 1998.

[3] R. Guerin and A. Orda, "Networks with Advance Reservations: The Routing Perspective," *In Proceedings of IEEE INFOCOM 2000*, pp. 118–127, 2000.

[4] H. Zhang, K. Keahey, and B. Allcock, "Providing Data Transfer with QoS as Agreement-Based Service," *International Conference on Services Computing (SCC 2004)*, Shanghai, China, September 2004.

[5] I. Foster, A. Roy, and V. Sander, "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation," *In 8th International Workshop on Quality of Service (IWQoS 2000)*, pp. 181–188, 2000.

[6] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler, "End-to-end quality of service for high-end applications," *Computer Communications*, vol. 27, no. 14, pp. 1375–1388, 2004.

[7] 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 7th International Workshop on Quality of Service (IWQoS)*, London, UK, pp. 27–36, 1999.

[8] L. Burchard, H. Heiss, and D. Rose, "Performance issues of bandwidth reservations for grid computing," *Proceedings of Computer Architecture and High Performance Computing*, pp. 82–90, 2003.

[9] J. Xing, C. Wu, M. Tao, L. Wu, and H. Zhang, "Flexible Advance Reservation for Grid Computing," *GCC 2004*, pp. 241-248, 2004.

[10] L. Wu, J. Xing, C. Wu, and J. Cui, "An Adaptive Advance Reservation Mechanism for Grid Computing," *PDCAT 2005*, pp. 400-403, 2005.

[11] B.B. Chen and P. Primet, "Supporting bulk data transfers of high-end applications with guaranteed completion time," *IEEE ICC2007 International conference on computer communication*, 2007.

[12] P. Primet and J. Zeng, "Traffic Isolation and Network Resource Sharing for Performance Control in Grids," *ACNS'05*, USA, 2005.

[13] L. Marchal, P. Primet, Y. Robert, and J. Zeng, "Optimal Bandwidth Sharing in Grid environment," *IEEE HPDC*, Paris, France, 2006.

[14] N. Kaushik, S. Figueira, and S. Chiappari, "Flexible Time-Windows for Advance Reservation in LambdaGrids," *ACM SIGMETRICS/Performance*, Saint-Malo, France, 2006.

[15] S. Naiksatam and S. Figueira, "Elastic Reservations for Efficient Bandwidth Utilization in LambdaGrids," *Elsevier's FGCS - The International Journal of Grid Computing: Theory, Methods and Applications*, vol. 23, issue 1, pp. 1-22, 2007.

[16] 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," *IEEE/IFIP International Workshop on End-to-end Virtualization and Grid Management (EVGM 2007)*, held as part of Manweek 2007, San Jose, California, USA, October 2007.

[17] 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," *ACM International Conference on Networks for Grid Applications (GridNets 2007)*, Lyon, France, October 2007.

[18] J.A. Müller, S. Hessler, and K. Irmscher, "Class of Service Concepts in Autonomous Systems," *Terena networking conference 2004*, Rhodes, Greece, 2004.