A QoS Framework for Clusters to Support Applications with Resource Adaptivity and Predictable Performance

Sandhya Senapathi Dhabaleswar K. Panda Don Stredney† Han-Wei Shen

Department of Computer and Information Sciences
The Ohio State University, Columbus, OH 43210
{senapath, panda, hwshen}@cis.ohio-state.edu

† Ohio Supercomputer Center
Columbus, OH 43212
don@osc.edu

Abstract—**Simultaneous advances in processor and network technologies have made clusters of workstations attractive vehicles for high-performance computing. Emerging applications targeted for clusters are inherently interactive and collaborative in nature. These applications demand end-to-end Quality of Service (QoS) in addition to performance. Achieving predictable performance and ability to exploit resource adaptivity are also common requirements of these next generation applications. Providing QoS mechanisms for clusters to satisfy the demands of next generation applications is a challenging task. In this paper, we propose a QoS framework that provides bandwidth guarantees for communication within a cluster. The framework consists of a novel Network Interface Card (NIC)-based rate control mechanism and a coordinated admission control/scheduling mechanism. An interface is developed so that applications using the common Message Passing Interface (MPI) standard can specify bandwidth requirements of their flows to the underlying network. The framework is developed and evaluated on a Myrinet cluster testbed for a range of scientific and visualization applications. The experimental evaluations demonstrate the various advantages such as predictability and resource adaptability associated with the framework. The proposed framework is quite unique and is the first of its kind in the literature to support next generation interactive and collaborative applications on clusters.**

I. Introduction

Clusters of workstations have emerged as powerful computing vehicles as a result of recent advances in high-speed networking technology and increasing processor speeds [1]. Such clusters are becoming increasingly popular for providing cost-effective and affordable computing environments for day-to-day computational needs of a wide-range of applications [2]. Traditionally, applications targeted for clusters have primarily included compute-intensive jobs such as scientific and engineering simulations. However, with the growth of modern networking and the Web technologies, a new generation of applications is being targeted for clusters. These applications include large databases, data mining, imaging, collaborative interactions, virtual reality, multimedia server, web server, distributed visualization, collaborative computing, and telemedicine [3].

The above applications are primarily *interactive* and *collaborative* in nature. Thus, in addition to demanding *high performance* , many of these applications demand *end-to-end QoS* from the underlying system. For example, a visualization application integrated with volume rendering on each node of a cluster might need to have some dedicated bandwidth from each node to a front-end node in the cluster so that the rendered data can be composed at the front-end node with a desired frames/sec and sent to the client over the LAN/WAN network. Similarly, for a cluster-based file server, a telemedicine application involving medical video transmission might require guarantee on bandwidth allocation from a back-end node (containing the disk with the file) to the front-end node so that it can be sent to the client over LAN/WAN at a certain rate. To provide such end-to-end QoS, one needs to have QoS associated with both computation and communication inside the cluster and support *predictable* execution time for the application. Many of these applications also demonstrate *resource adaptive* property so that they can adapt depending on the available resources (such as CPU cycles and network speed) and desired QoS level for the applications [4].

Current generation clusters using the popular interconnects like Myrinet [5], Fast Ethernet/Gigabit Ethernet [6], and GigaNet [7] do not provide any QoS support at switches or at NICs. Clusters typically use irregular topologies and associated deadlock-free routing schemes [8]. Thus, disjoint partitioning of computing nodes in a cluster for different jobs does not necessarily lead to disjoint paths for inter-processor communication and I/O for these jobs. Even though a space-partitioned node allocation scheme is used for mapping different jobs to the nodes of a cluster, some communication links may still be used in a time-shared manner. This is a serious impediment for harnessing the computational power of clusters for next generation applications. In this paper, we take on this challenge and present a QoS framework that provides bandwidth guarantees for communication within a cluster. The framework consists of a novel Network Interface Card (NIC)-based rate control mechanism and a coordinated admission control/scheduling mechanism. The rate control mechanism works by controlling the rate at which packets are injected at the source NIC into the network. This NIC-based scheduling scheme provides advantages of finer granularity, and lessens the processing load on the host CPU. This scheme is used in conjunction with the call admission control/scheduling unit which maintains global knowledge of cluster traffic patterns to provide bandwidth allocation to competing flows and that

0-7803-7426-6/02/$17.00 (c) 2002 IEEE.

monitors incoming requests for allocation and deallocation. An interface is developed so that applications using the common Message Passing Interface (MPI) standard [9], [10] can specify bandwidth requirements of their flows to the underlying system. The framework is developed and evaluated on a Myrinet cluster testbed for a range of scientific and visualization applications. The experimental evaluations demonstrate the following advantages associated with the proposed QoS framework:

1) Resource requirements of applications can be studied by performing experimental evaluations with a range of bandwidth allocations. These evaluations show the minimum and maximum network bandwidth required by the application as well as its resource adaptive property.
2) An application can be executed in a cluster in a predictive manner (with respect to overall execution time) with no network interference from competing flows of other applications or best-effort traffic.
3) Multiple applications can execute in a cluster transparently from each other leading to the concept of network partitioning whereby every application has the illusion of having an exclusive virtual network that has no interference from flows from other jobs also running on the same cluster.

The proposed framework is quite unique and is the first one to be proposed in the literature to support next generation applications with predictability, resource adaptability, and end-to-end QoS. This framework has been developed and evaluated on Myrinet clusters. However, it is very general and can be implemented on clusters with programmable network interfaces such as Gigabit Ethernet [6] and InfiniBand [11].

The rest of this paper is organized in the following manner. Section II explains in detail the problems faced by current applications and the need for a QoS framework in clusters. Section III introduces the basic concepts behind the QoS mechanism. Section IV explains in detail the basic rate-control algorithm and also gives some details of the implementation. Section V moves one layer higher to the interface developed for Message-Passing applications and also explains in detail the working of the bandwidth policing manager. Section VI provides information about experiments performed and results obtained. Section VII looks at related work on QoS. Section VIII gives some insight into ongoing and future work from our research group in the same area and gives a brief conclusion summarizing the paper.

II. The Problem

As indicated in the previous section, clusters of workstations are increasingly being used by a range of applications. However, clusters are not being operated in the traditional single application mode but are used by multiple applications simultaneously for more efficient resource utilization. Running any such application on a cluster guarantees that each application will get its own disjoint set of processors. However no such guarantee can be made for the underlying network. As shown in Figures 1 and 2, applications running on clusters of uni-processor or SMP workstations may affect each other due to contention in the network. This contention is dependent on the underlying topology and routing scheme [8] and is unavoidable for large clusters.

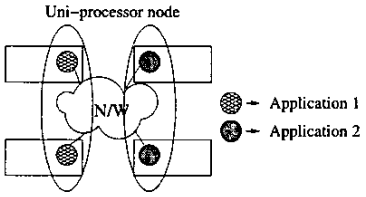

Fig. 1. Applications running on a cluster of uni-processor machines

Such contention may adversely affect the performance of the applications, even though the network capacity may be very high. Although each application will run on an exclusive set of processors there might be interference in the network from other applications that are also using the cluster. The behavior of the application is thus no longer predictive. Therefore, a guarantee on network resources to a given application is needed. A resource adaptive application can execute with any fraction of the underlying network resources. Such applications can therefore reserve any available network bandwidth and can be considered to have an exclusive partition of the network. The overall view of the system would be a set of resource adaptive applications each running on its own exclusive virtual network.

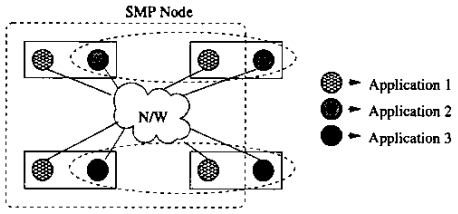

Fig. 2. Applications running on a cluster of shared memory multi-processor(SMP) machines

Bandwidth guarantees are also required for client-server applications. Consider a scenario shown in Figure 3.

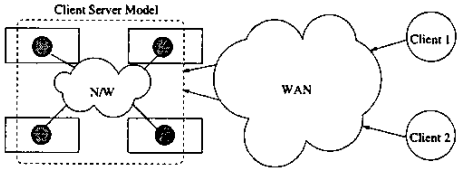

Fig. 3. Client applications accessing a cluster-based server

A client situated on a remote node over a wide-area network (WAN) requires to transfer a large amount of data from a

group of server-workstations on a cluster, for example in a file server. The client would require some kind of guarantee on network resources not only in the WAN context but also to eliminate contention from other clients accessing the cluster. QoS reservations within the cluster can help provide a deterministic behavior of the response time and the data transfer time between the client and the server. Such a cluster-level QoS support together with WAN-level QoS support can lead to an end-to-end QoS mechanism which can provide bandwidth guarantees on all components of the path between a client and the server.

III. Basic Idea behind the proposed QoS Framework

The proposed QoS framework is implemented with a rate-control mechanism at the NIC level together with a Call Admission Control (CAC) manager. Communication flows are regulated by controlling the rate at which data is transferred into the network interface and sent into the network.

Such a rate-control scheme can be implemented at the host, but a NIC-based scheme is preferred since it allows a finer granularity of control, because the NIC deals with frames whereas the host will deal with messages. Since the QoS mechanism is implemented by the firmware on the NIC, which is loaded by the operating system, it can be trusted, and no other policing mechanism is required. Thus, this solution is particularly attractive in that it requires no additional hardware components, or changes in commodity components. An example of the rate-control mechanism is shown in Figure 4. Every NIC connecting to the network has the QoS features uploaded on it. Applications that are executing at the hosts can reserve certain amounts of network resources in terms of network bandwidth, and be guaranteed that the reserved bandwidth is available only to them. This scenario is similar to the case in which every executing application has its own independent virtual network that is unaffected by interference due to the communication flows of other processes on the cluster. As shown in Figure 4, though the flows of application 1 running on Hosts A and B, and application 2 running on Hosts C and D share links between the switches, there will not be any interference between them if a certain amount of bandwidth is reserved by each communication flow (not exceeding the total capacity of the link), and the rate of injection of each flow into the network is controlled.

The proposed rate control mechanism is only a local scheme and every NIC can be expected to know only information about flows coming out of/going into that host node. However, reservation of network links requires a more global knowledge of the traffic patterns in the cluster. A trusted manager called the Call Admission Control (CAC) manager maintains such information.

Figure 5 shows the steps taken when a new reservation or a change of reservation has to be made. A CAC manager approves reservation requests of every application before they can be granted. The CAC keeps information about the

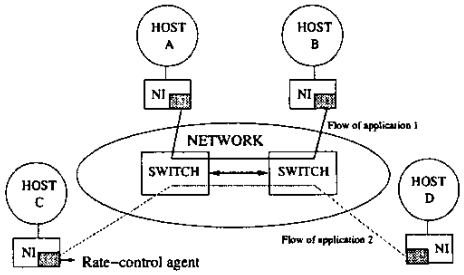

Fig. 4. Applications sharing the network are guaranteed a certain pre-specified share of the network resources

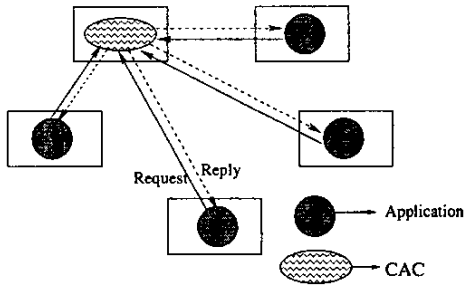

Fig. 5. The Call Admission Control (CAC) Manager and it's interaction with an application

bandwidth available on every link in the network. When a request comes in, either for a new bandwidth allocation or for a change of reservation, the CAC uses the stored data to verify that granting the request will not over-allocate the links to be traversed by the flow in the network. If this is the case, it grants the request, and updates the information maintained in its data structures. The responsibility of the CAC manager is therefore to make sure that no link in the cluster is being over-allocated and to inform the rate control agent in the NIC of changes in bandwidth reservations arising from allocations and deallocations. Section V-B gives a detailed description of the working of the CAC manager. For advanced cluster interconnects such as Myrinet, source-based routing is used [8]. Thus, the entire path (in terms of links in the network) is known. In Section IV we will take a closer look at the actual mechanism at the NIC that performs rate-control on packet injection into the network.

IV. Overview of NIC-based rate control

A message stream between applications is denoted by the term *communication flow*. Every communication flow has a well-defined source and a well-defined sink. The end-points of the flow are logical and there may be several communication endpoints on the same physical network node. The flows are then multiplexed over the physical link. When requesting bandwidth guarantees, an application is required to make QoS reservations for each of the communication flow originating from it. Under the rate-control algorithm, the assumption is that the bandwidth requested by the application is the

TABLE I

WORKING EXAMPLE OF THE RATE-CONTROL ALGORITHM

t	0	1	2	3	4	5
NDT(A)	0	2	2	4	4	6
NDT(B)	0	0	3	3	6	6
Flow serviced	A	B	A	B	A	-

maximum amount that it can obtain. The CAC agent running on that node, after checking to make sure that the request can be satisfied, maps the bandwidth value to a parameter known as the *Inter-Dispatch Time* (IDT) and passes this value to the rate-control agent at the NIC. Therefore the IDT value associated with every flow can be defined as the minimum interval that can elapse between the consecutive injections of the packets from that flow into the network interface. At any given time, let f_1 to f_n be n communication flows sourced at a particular node. Let their corresponding IDT values be IDT_1 to IDT_n. The actual rate control algorithm uses another set of parameters called *Next Dispatch Time* (NDT) NDT_1 to NDT_n which specifies the absolute time before which a packet from a given flow should not be dispatched. The NDT value of a flow is initialized to the current time when a message send is posted for that flow.

Let t be the time and j be the communication flow.

The rate control algorithm works like this:

NDT_j = min (NDT_1 to NDT_n)

If $NDT_j <=$ t

Dispatch packet from flow j

Update NDT_j to NDT_j + IDT_j

Thus for every flow we have to maintain IDT and NDT information at the NIC and these values will be used in deciding the next packet to be transferred into the network. Assume that we can send out packets at most once every T time units. This T then corresponds to the peak achievable bandwidth B_{max}. Table 1 shows a working example of the rate control algorithm in the presence of two flows from sources A and B, each requesting IDT values of 2 and 3 respectively. At time t = 0, the NDT values of both A and B are equal to the current time and both can be sent. We assume, without loss of generality that A is selected as the next flow to be serviced. Once a packet from $flow_A$ is dispatched, its NDT value is incremented by $IDT_A = 2$. At time t = 1, $flow_B$ has the smallest NDT value, and the value of NDT_B is also less than the current value of t. Thus at time t = 1, $flow_B$ is serviced and NDT_B is updated to NDT_B + IDT_B. Again at time t = 2, A is ready to be serviced again. The algorithm proceeds in similar steps. It should be noted that at time t = 5, the NDT values of both A and B are greater than the current time, and no packet is dispatched.

The rate control mechanism was implemented in the GM messaging layer over Myrinet networks [12], [13].The Myrinet NIC is controlled by firmware called the Myrinet Control Program (MCP) [14]. The MCP consists of four state machines: SDMA, SEND, RDMA and RECV, that take care of message sends and receives. Message sends and receives therefore do not involve the operating system, but take place by direct interaction between the client and the MCP through DMA of data. Additional QoS information for a flow was added to the MCP so that the NIC maintains the IDT values for various flows. IDTs are specified in terms of clock ticks of the *Real Time Clock* (RTC) available on the LANai processor on the Myrinet NIC. The modified SDMA state machine maintains information about the validity of the flow and its NDT and IDT values. If a send has been posted for a flow, then the flow is declared to be valid. Whenever a send token is inserted as the first sendable token for a flow, the flow is validated and its NDT value is updated to the larger of RTC and the existing NDT value. Packet scheduling in the NIC is now based on NDT values. To search for the next packet to be sent, the SDMA state machine finds the flow with the minimum NDT among all valid flows. If the RTC is larger than the NDT value for this flow then the flow is serviced using its first sendable token and its NDT is incremented by its IDT value. This procedure is followed every time a packet has to be sent. To reduce the overhead of the QoS mechanism, the next smallest NDT value is searched in parallel with the DMA of the current packet being sent. This ensures that the search for the smallest NDT value among all the flows that have requested QoS does not fall in the critical path of sends. If no flow is ready to be sent at a given time step, then data from non-premium or best effort flows is transmitted.

V. QoS FRAMEWORK FOR SUPPORTING MPI-LEVEL APPLICATIONS ON CLUSTERS

So far we have described the incorporation of the QoS framework at the GM level for providing rate control. But GM is only a low-level communication layer for the network. This scheme requires a user-level interface and some kind of policing so that applications cannot arbitrarily request and obtain any bandwidth value. In this section we will first look at the changes needed at the Message Passing Interface (MPI) layer [9] for providing an interface for resource reservations and deallocations. Then we will have a detailed look at the structure of the Call Admission Control (CAC) manager, a user level program that monitors incoming requests for bandwidth and makes the decision as to whether the request can be allowed or rejected based on the information maintained about existing reservations in all the links of the cluster.

A. Support for MPI applications

The basic QoS feature at the GM level is sufficient for guaranteeing network resources and for actually administering QoS policy. However, for ease of programming and portability, support has to be added at the application level. We have modified a familiar programming model, the Message Passing Interface (MPI) [9], since it is in widespread use and freely available. MPI is a message-passing library that offers a range

of point-to-point and collective interprocess communication functions to a set of single threaded processors executing in parallel. All communication is performed within the definition of a communicator. A communicator is a group of processes that are communicating with each other, in which each process has a unique id between 0 and N-1, N being the number of processes in the communicator. MPICH [15] is a freely available, portable implementation of MPI. The mechanism for achieving portability is a specification called the Abstract Device Interface (ADI). All MPI functions are defined in terms of a set of basic MPI communication primitives, which are implemented in the ADI layer. This layer uses message-passing functions native to the underlying system. MPICH-GM is therefore an implementation for Myrinet clusters that uses GM as the underlying message-passing system. As Figure 6 shows, the ADI layer in MPICH-GM uses GM primitives to implement MPI primitive functions in terms of which more complex MPI operations are defined.

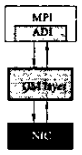

Fig. 6. The Message Passing Interface

To use the QoS features that GM provides in a standard-compliant fashion, we used the *Attribute mechanism* provided by the MPI layer [10]. MPI provides functions to set and get the values of attributes for a communicator. An attribute is identified by an integer, a *keyval*, and its value may be of any arbitrary type (void * in C). When used with special *keyval* values that signify that the value is related to QoS parameters, the *set attribute* function call in MPI is mapped to the GM function for requesting bandwidth allocation or deallocation. Process ids are converted into appropriate GM node and port identifiers. Also, since the MPI layer incurs additional overhead, bandwidths achieved at the MPI level are lower than raw GM bandwidths. The *get attribute* call can be used to determine if a request made using the *set attribute* function call is accepted or denied. The bandwidth value requested can also be changed dynamically during the execution of the application by using the same attribute function calls.

B. Implementation of Call Admission Control (CAC)

The CAC is the reservation entity at the cluster-level that handles requests for bandwidth allocation and deallocation on network links. The criteria for admitting a new communication flow with a given bandwidth requirement is as follows:

1) If the flow is admitted, then the cumulative bandwidth requirement for all flows at the source network node of the new flow should not exceed its peak capacity.
2) If the flow is admitted, then the cumulative bandwidth requirement for all flows using the switch output ports traversed by the new flow should not exceed the corresponding outgoing link capacity.
3) If the flow is admitted, then the cumulative bandwidth requirement for all flows at the destination network node of the new flow should not exceed its peak capacity.

If the addition of a reservation request satisfies all of the above conditions then the request is permitted. Now, it is the duty of the CAC to convert this reservation into an IDT value and inform the rate-control agent at the NIC level of the new premium reservation. The IDT value is defined as follows: $IDT = T * (B_{max}/B_{req})$ where B_{req} is the bandwidth requirement of the new flow.

The implementation of the CAC is designed in such a way that a copy of the CAC should run on every node in the cluster. The local copies of the CACs are called *slave CACs* . The following scenario is illustrated in Figure 7. An application on a node will send its reservation request to the local slave CAC running on that node. One of these CACs is also designated as the *master CAC*, which maintains information about all live requests currently operating on all the links in the cluster. To reserve a request, each slave CAC sends it to the master CAC, which then uses the information it maintains to study if the three conditions specified above are satisfied if the new request is to be allowed. Once a decision is taken, the master informs the slave, which then either makes the reservation if allowed, or informs the requester that the reservation cannot be satisfied if disallowed. The actions taken when an MPI application makes a QoS request are as follows:

1) The 'set attribute' implementation studies the value of the given keyval and since it is a request for QoS, sends the request to the locally running slave CAC.
2) The slave CAC passes the request to the master CAC, which then decides whether the request can be allowed based on the maintained information. The master CAC then informs the slave of its decision. If the local CAC is the master CAC, then it makes the decision itself.
3) In either case, if the reservation has been accepted, the slave CAC calculates the corresponding IDT value, passes the request to the rate-control manager at the NIC, and informs the client application that its request has

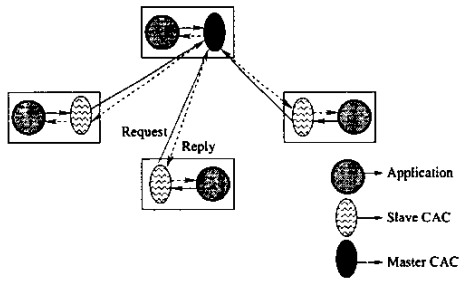

Fig. 7. Interaction between the master and the slave CACs

been accepted. At the NIC, flow information for the new flow is set up and values are initialized.

4) If the reservation is rejected, the slave CAC informs the client application that its request has been rejected.

Since the slave CAC makes the reservation on behalf of the user, a rogue application cannot misuse the bandwidth feature by assigning any desired bandwidth value to itself. The CAC also maintains information about all best-effort flows in the system, those that have not requested any specific bandwidth value but are willing to use whatever bandwidth is remaining after allocations have been done. As premium requests are allocated and deallocated as requested by the applications, the CAC also calculates the amount of bandwidth remaining for best-effort traffic and updates the applications about the changed values.

VI. Performance Evaluation

In this section, we present detailed, application-level evaluation to validate our framework as well as demonstrate its benefits. We carried out the following three kinds of experiments.

1) Applications were run with varying bandwidth reservations to show the impact of rate control on their execution time. This test also serves to highlight the resource adaptive nature of the application by which the application can be executed with any available network bandwidth allocation.
2) Applications were run in the presence of background bandwidth-hungry flows to show the effectiveness of the QoS framework in reserving network resources and delivering predicted execution time.
3) Multiple applications were executed simultaneously on the same set of nodes, on different sets of processors, but sharing the same underlying network. Since each application reserves a certain percentage of network bandwidth, the network gets essentially partitioned into disjoint virtual networks, where each virtual network is available for the exclusive use of the application that has reserved it. This test also highlights the predictable execution time for an application in the presence of background applications.

A. Experimental testbed

The implementation was evaluated on a cluster of workstations with eight 700 Mhz Quad Pentium III processors, running Red Hat Linux kernel version 2.4.7-10 smp. These machines were connected by an 8-port Myrinet switch and LANai 7.2 NICs with 66 MHz processors. The communication layer running on the Myrinet cards was GM 1.5.1, and the MPI version was MPICH 1.2.1.7. To determine the maximum bandwidth that can be supported by the network, MPI bandwidth and latency tests were performed on the testbed. It was determined that the maximum bi-directional bandwidth that can be supported per link is 210 MBps (Mega Bytes per second).

B. Overview of Applications

The tests were performed using the NAS Benchmark Suite [16], and two visualization applications. The NAS Benchmarks used were Integer Sort (IS), Block tridiagonal solver (BT), LU solver (LU), Conjugate Gradient (CG), Multigrid (MG), and Pentadiagonal solver (SP). All the NAS benchmarks with the exception of MG were executed with size class A and some of the benchmarks were executed with number of iterations and size differing from the class A default. The visualization applications tested were an iso-surface extraction application and a ray-tracing application that can be executed with different sizes of input data.

C. Impact of rate control on application execution time

The applications were run with different bandwidth reservations. Uniform allocations were made to every communication flows in the application. The tests were performed for four and eight nodes. The results are shown in Figure 8 for the NAS benchmarks and the visualization applications for four nodes, and in Figure 9 for eight nodes. Since reservations were made for all possible communication links, each application on N nodes will have N-1 outgoing flows, and N-1 incoming flows, making the total number of flows on the link from a node to the switch as 2N-2. By this calculation, on four nodes, the maximum reservation that can be made per flow is 210/6 = 35 MBps, and for eight nodes, it is 210/14 = 15 MBps. The results show that as the allocated bandwidth for each flow is increased, the execution time decreases. Since these are not simple bandwidth or latency tests, but complex applications with various computation to communication ratios and different communication patterns, the rate at which the execution time decreases varies according to the application. These tests also show the resource-adaptive nature of the application, in that the application can be run with any assigned QoS value. It also shows the predictive property that the QoS mechanism provides i.e., a reservation of a certain amount of network resources for an application provides a guarantee that the application will complete within a certain amount of time. By detailed study of the graph curves, one can also determine the exact minimum bandwidth required for each application for a certain data size. For example, 15 MBps is sufficient for the iso-surface extraction application on 4 nodes. This indicates that an application need not reserve all available network bandwidth and the remaining available bandwidth can be shared with other applications.

D. Guaranteeing QoS in the presence of background flows

The same applications were tested on four nodes again, but now in the presence of 2 and 3 background flows that consists of a sender running continuously and pumping out data to a receiver. The experimental setup is shown in Figure 10. The first test consists of 2 background flows, whereas

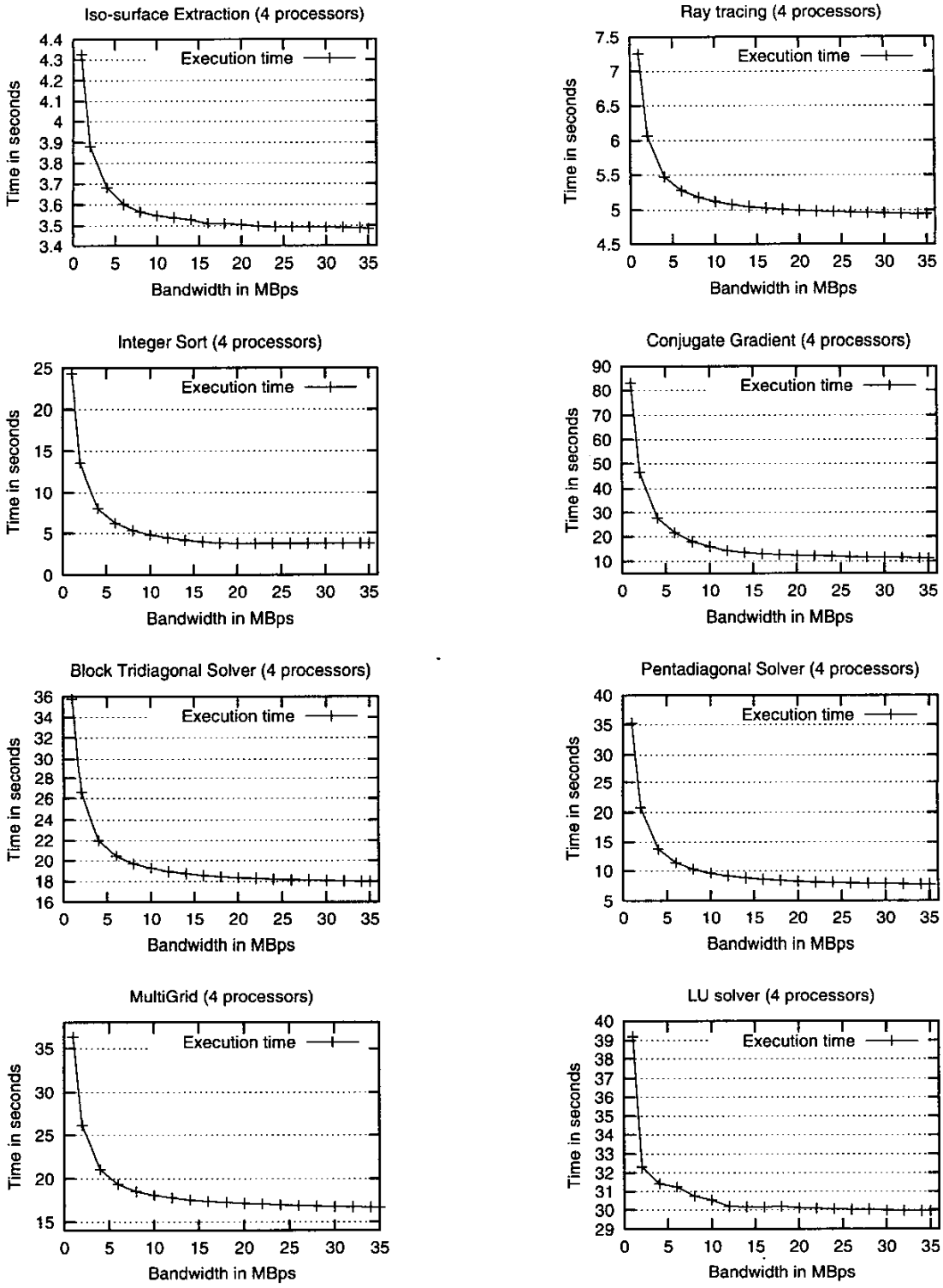

Fig. 8. Impact of rate-control on execution time of applications on 4 nodes

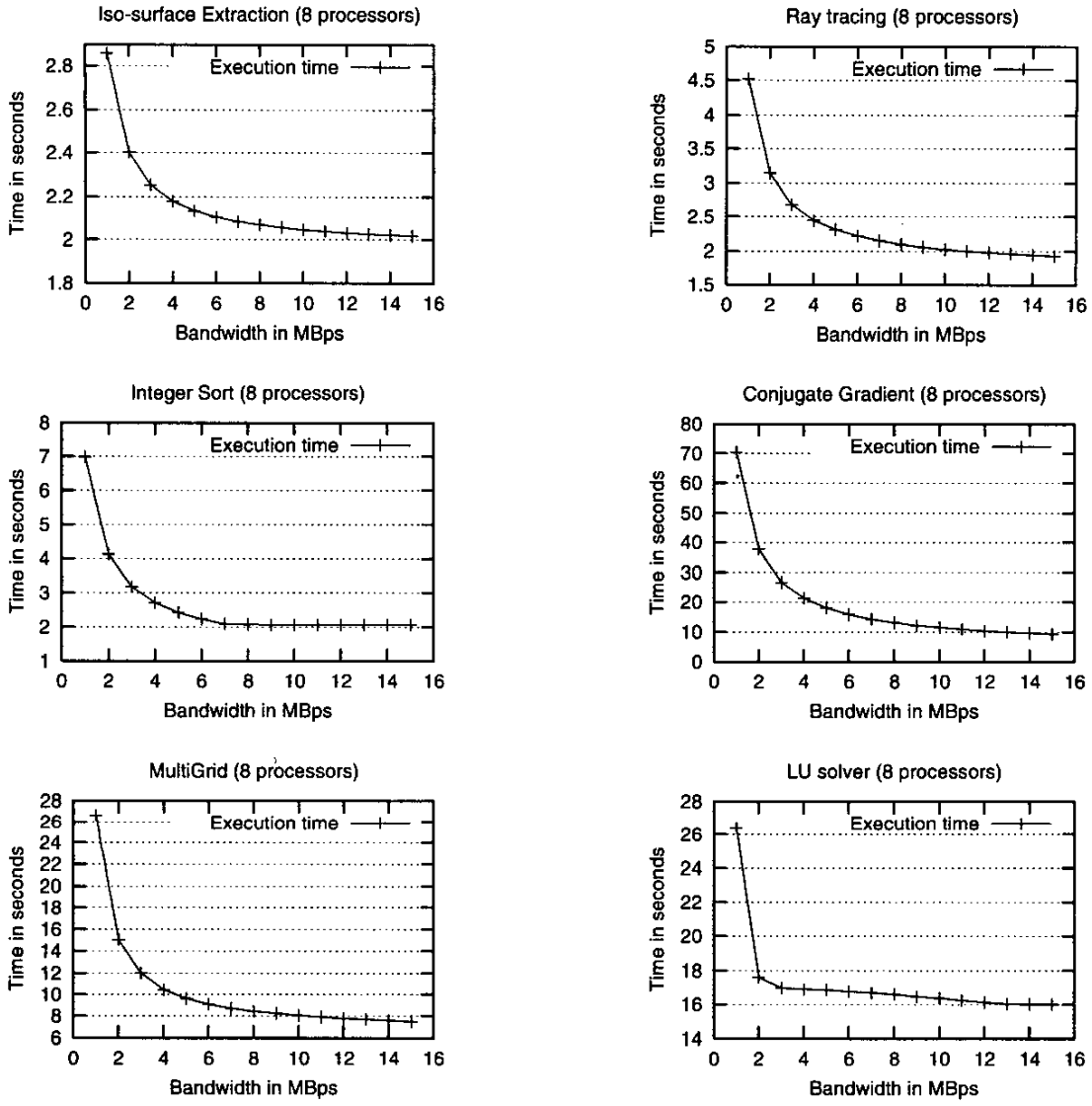

Fig. 9. Impact of rate-control on execution time of applications on 8 nodes

the second test consists of 3 background flows. Since these are parallel applications that interact frequently, it is sufficient that we run the background flows on one of the nodes on which the application is executing so as to put the maximum load on the NIC at that node. In the communication system without QoS features, these flows are treated equal to the communication flows of the application and will therefore cause more interference. Whereas in the network layer with QoS features, these background flows, not having made any reservations will be treated as best-effort traffic, and will take second priority to the premium traffic. The results are shown in Figure 12 for four nodes. Figure 12a shows the results obtained for the NAS benchmarks in the presence of three background flows. Figure 12b shows the results obtained for the ray-tracing application in the presence of 2 and 3 background flows. From the graphs, it is clear that the QoS framework guarantees a predictable response time for the application even in the presence of heavy-duty flows running on the same nodes. Since the nodes in the testbed are quad SMPs, we can reasonably suppose that the flows are assigned to different processors from those on which the application is running, and the interference due to the flows is solely at the network level.

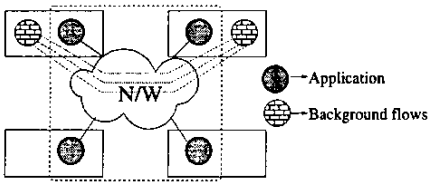

Fig. 10. Performance evaluation framework in the presence of background flows

E. Executing multiple jobs simultaneously with individual QoS requirements

In this final evaluation, we ran 2 applications simultaneously with different QoS values. Since each application has allocated bandwidth, it has exclusive use of the portion of network resources assigned to it, and behaves as if it is running on an exclusive virtual network with resources equal to the value of bandwidth allocated to that application. This mechanism thus paves the way for efficient and reliable use of clusters wherein network resources can be partitioned for client applications with no interference between users. The performance evaluation framework for this experiment is shown in Figure 11. The results of these tests are shown in Figure13 for four and eight nodes.

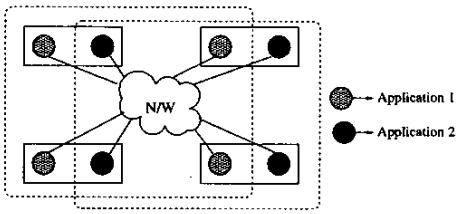

Fig. 11. Performance evaluation framework for executing multiple applications simultaneously with individual QoS requirements

The x-axis plots the value of assigned bandwidth to the application pairs and triples in MBps. The y-axis plots the total execution time of the applications in seconds. It can be seen that even in the presence of a background application, the execution times of the application do not change, and are the same as in the experiment where only a single application is run. For example, from Figure 13a, the execution times for the bandwidth pair (5,8) is (0.1404, 0.09175). From the graph for Integer Sort (IS) in Figure 8, we can see that the execution time is 0.1404 when the bandwidth reserved is 5 MBps, and it is 0.0925 when the bandwidth reserved is 8 MBps. Thus the presence of another application does not affect the performance of an application as long as it has reserved a fraction of the network bandwidth.

VII. Related Work

In recent years, a few other studies have examined service differentiation and the related issue of QoS in clusters. Low-cost packet scheduling and queuing algorithms for high performance networks are described in [17]. In [18], Gerla et al describe three schemes for providing QoS in wormhole-routed networks. However, these require modifications to switches or extra NICs on hosts. FM-QoS [19] provides predictable performance by devising conflict-free communication schedules. In [20] and [21], the authors propose new hardware or hardware/software infrastructures to support QoS. The Globus project's GARA framework [22], [23] provides end-to-end QoS for different types of resources. The framework presented in this paper is quite different. It uses a NIC-based approach to provide QoS in current and next generation clusters with programmable NICs.

VIII. Conclusions and Future Work

We have presented a QoS framework for supporting applications in clusters with resource adaptivity and predictable execution performance. The framework uses a NIC-based rate control mechanism. Experimental results have shown that the framework can be used to execute applications in a resource-adaptive manner. It can also handle interference between applications and provide a scheme for partitioning clusters without any significant overhead or hardware modifications. Applications running on clusters can therefore execute independently, without interference from other applications using the cluster, at the same time providing good resource utilization. We are currently exploring the addition of a monitoring mechanism to the QoS framework at the NIC level that monitors the behavior of packets belonging to individual flows, to study if the bandwidth allocated to an application is suitable for the data it is actually sending, whether the flow has been over-allocated or whether it is being choked. This feature, combined with a dynamic re-allocation scheme will allow us to determine the exact bandwidth resource requirements of applications and re-assign the bandwidth value on the fly. Another extension to the mechanism is support end-to-end QoS reservations, so that a client application accessing a service on a cluster-based server over a WAN can be assured of bandwidth guarantees over both network components of the total path, over the WAN and the LAN/SAN in the cluster. This work will entail merging the existing cluster-level QoS mechanism with a wide-area QoS mechanism. We plan to integrate our work with Globus' GARA and MPICH-GQ frameworks to achieve this.

Acknowledgment: The authors would like to thank Abhishek Gulati for his invaluable contribution during the earlier period of this project. The authors would also like to thank Darius Buntinas and Dr. Pete Wyckoff for their help on GM and MPICH-GM.

References

[1] T. Anderson, D. Culler, and Dave Patterson. *A Case for Networks of Workstations (NOW)*, IEEE Micro, pages 54-64, Feb 1995.

[2] D. K. Panda and C. B. Stinkers, editors. *Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC)*, Lecture Notes in Computer Science, Volume 1362, Springer-Verlag, 1998.

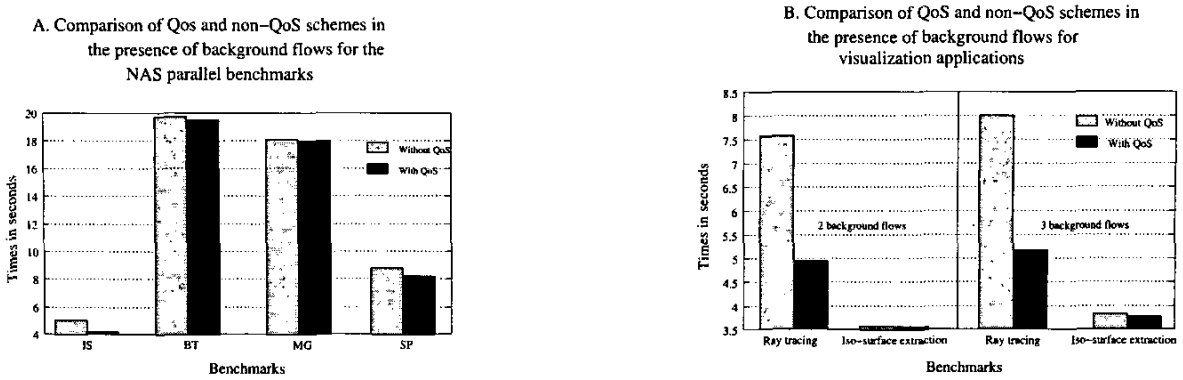

Fig. 12. Performance comparison between QoS-based and non-QoS-based frameworks

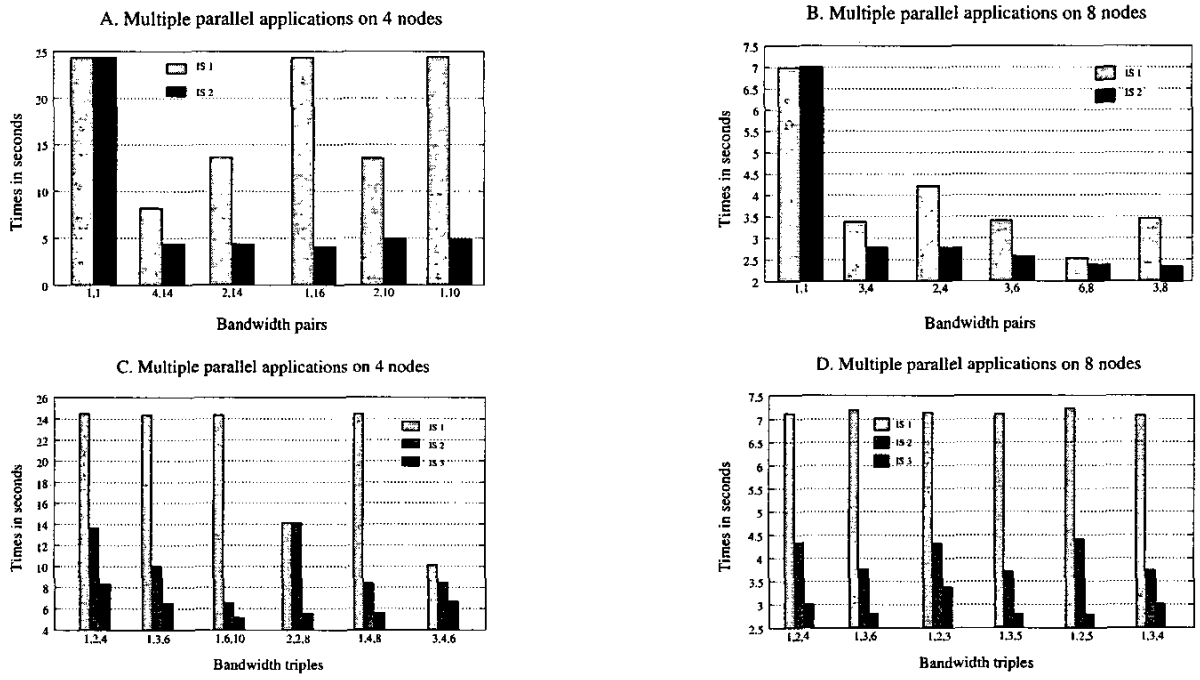

Fig. 13. Performance evaluation of multiple applications executing on 4 and 8 nodes with individual QoS requirements

[3] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[4] I. Foster, A. Roy, and V. Sander. *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*, Int'l Workshop on Quality of Service (IWQOS '00), June 2000.

[5] N. J. Boden, D. Cohen, et al. *Myrinet: A Gigabit-per-Second Local Area Network*, IEEE Micro, pages 29–35, Feb 1995.

[6] R. Sheifert, *Gigabit Ethernet*, Addison-Wesley, 1998.

[7] *GigaNet Corporations*, http://www.giganet.com.

[8] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. The IEEE Computer Society Press, 1997.

[9] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, March 1994.

[10] A. Roy, I. Foster, W. Gropp, N. Karonis, V. Sander, and B. Toonen. *MPICH-GQ: Quality of Service for Message Passing Programs*, Int'l Conference on Supercomputing (SC '00), November 2000.

[11] *InfiniBand Trade Association*, http://www.infinibandta.com .

[12] A. Gulati, D.K. Panda, P. Sadayappan, P. Wyckoff. *NIC-Based Rate Control for Proportional Bandwidth Allocation in Myrinet Clusters*. In Int'l Conf On Parallel Processing, September 2001.

[13] A. Gulati, *A proportional bandwidth allocation scheme for Myrinet clusters*. M.S. thesis, Dept. of Comp. & Info. Science, The Ohio State University, June 2001.

[14] *The GM Message Passing System*. Documentation available at hhtp://www.myri.com/scs/index.html.

[15] W. Gropp, E. Lusk, N. Doss, N. Skjellum. *A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard*. Tech. Report, Argonne National Laboratory and Mississippi State University.

[16] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, M. Yarrow. *The NAS Parallel Benchmarks 2.0*. Tech Report NAS-95-020, December 1995.

[17] J. H. Kim, A. Chien. *Rotating Combined Queuing (RCQ): Bandwidth and latency guarantees in low-cost, high-performance networks*, In Proceedings of the Int'l Symposium on Computer Architecture: 226-

236, May 1996.

[18] M. Gerla, B. Kannan, B. Kwan, P. Palnati, S. Walton, E. Leonardi, F. Neri. *Quality of Service support in high-speed, wormhole-routing networks.* In Int'l Conf. On Network Protocols, Oct. 1996.

[19] K. Connelly, A. Chien. *FM-QoS: Real-time communication using self-synchronization schedules.* In Proc. Of Supercomputing conference, San Jose, CA, November 1997.

[20] H. Eberle, E. Oertli. *Switcherland. A QoS communication architecture for workstation clusters.* In Proceedings of ACM ISCA '98, Barcelona, Spain, June 1998.

[21] R. West, R. Krishnamoorthy, W. Norton, K. Schwan, S. Yalamanchili, M. Rosu, S. Chandra. *Quic: A quality of service network interface layer for communication in NOWs.* In Proc. Of the Heterogeneous Computing Workshop, in conjunction with IPPS/SPDP, San Juan, Puerto Rico, April 1999.

[22] I. Foster, C. Kesselmann, C. Lee, B. Lindell, K. Nahrstedt, A. Roy. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation.* In 7th IEEE/IFIP International Workshop on Quality of Service, 1999.

[23] I. Foster, A. Roy, V. Sander. L. Winkler. *End-to-end Quality of Service for High-End Applications.* IEEE Journal on Selected Areas in Communications, Special Issue on QoS in the Internet, 1999.