[7] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing", *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 3-6, 1995, Copper Mountain, Colorado, pp. 40-53.

[8] R. Felderman, A. DeSchon, D. Cohen, and G. Finn, "ATOMIC: A High-Speed Local Communication Architecture", *Journal of High Speed Network*, Vol. 1, 1994, pp. 1-28.

[9] C. Huang and P. K. Mckinley, "Communication Issues in Parallel Computing across ATM Networks", *IEEE Parallel and Distributed Technology*, Vol. 2, No. 4, Winter 1994, pp. 73-86.

[10] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistics Multiplexing", *Proceedings of ACM SIGCOMM'94 Symposium on Communication Architectures, Protocols and Applications*, August 31 - September 2, 1994, London, United Kingdom, pp. 101-114.

[11] M. Lin, J. Hsieh, D. Du, J. Thomas, and J. MacDonald, "Distributed Network Computing over Local ATM networks", *IEEE Journal of Selected Areas in Communications*, Vol. 13, No. 4, May 1995.

[12] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet", *Proceedings of Supercomputing 95*, San Diego, California, 1995, (CD-ROM).

[13] J. P. Singh, W. D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared Memory", *Computer Architecture News*, Vol 20, No 1, 1992, pp. 5-44.

[14] W. Stallings, *Handbook of Computer-Communication Standards, Volume 2: Local Network Standards*, MacMillan, 1987.

[15] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994.

[16] V. S. Sunderam, "PVM: A framework for parallel distributed computing", *Concurrency: Practice and Experience*, Vol. 2, No. 4, 1992, pp. 315-339.

[17] C. A. Thekkath, "System Support for Efficient Network Communication", Department of Computer Science, University of Washington, PhD Dissertation, 1994.

[18] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, 1995.

- Modeling and simulation of a network interface handling two priority levels. We propose a scheme called Double Queue Scheme to reduce the effects of local jobs' communication on the performance of parallel jobs. It divides the only one input and output queues in traditional network interfaces into two ones. One is for the communication of parallel jobs, and the other is for the communication of local jobs. Two queues are given different priorities to adjust the allocation of communication bandwidth among them. By giving the parallel jobs' queue a higher priority, through simulation, we find that the scheme will improve the performance of parallel jobs considerablely with a slight effect on local jobs communication especially when the sizes of local jobs' communication is small and moderate.

How to schedule parallel jobs and local jobs is another issue that will affect the performance of both jobs. We have proposed a power preservation approach [6] to schedule the jobs in a NOW. But the communication overhead is not considered there. Currently, we are integrating them and evaluating the performance when both processors and networks are taken into account.

Acknowledgement: We are grateful to the anonymous referees for their rigorous and constructive comments.

References

[1] ANSI X3T9.3/90-043, "High Performance Parallel Interface", American Standard Institute, 1990.

[2] ANSI X3.269-199x, "Fibre Channel Protocol for SCSI", American National Standard Institute, June 1, 1995.

[3] D. Bailey, et al., "The NAS parallel benchmarks", *International Journal of Supercomputer Applications*, Vol. 5, No. 3, 1991, pp. 63-73.

[4] P. W. Dowd, S. M. Srinidhi, and R. Claus, "Issues in ATM Support of High Performance Geographically Distributed Computing", *Proceedings of IPPS'95 Workshop on High Speed Networks*, April 1995, Santa Barbara, California, pp. 352-358.

[5] X. Du, Y. Dong, and X. Zhang, "Characterizing Communication Interactions of Parallel and Sequential Jobs on Networks of Workstations", *Proceedings of the IEEE International Conference on Communications*, June 8-12, 1997, Montreal, Canada, pp. 1133-1137.

[6] X. Du and X. Zhang, "Coordinating Parallel Processes on Network of Workstations", To appear in *Journal of Parallel and Distributed Computing.*

queue scheme in the message filter under the same condition as 2), and T_3 was measured.

Again, the curves of T_2/T_1 in the environment without using the double queue scheme and T_3/T_1 using the double queue scheme are plotted in Figure 10 to observe the slowdowns of both the testing streams (streams A and B) and the local stream when the message size of the local stream increased from 2 Kbytes to 16 Kbytes.

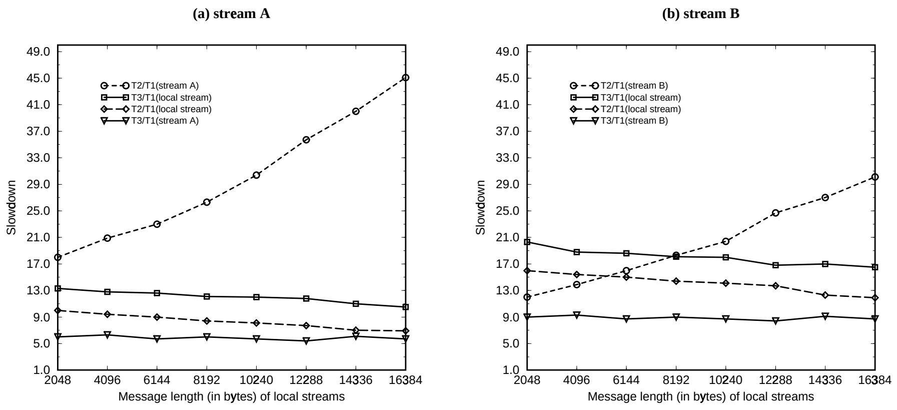


Figure 10: Experiments with the one-to-all and all-to-one combinations.

Figure 10(a) and (b) show that the message transmission times of both testing streams A and B were delayed up to 45 times and 30 times, respectively, when message types are not distinguished. The local stream itself were delayed by streams A and B up to 7 and 12 times, respectively. The double queue scheme in the message filter reduced the delay for stream A to 6 times (about 8 times reduction), and for stream B to 9 times (3 times reduction). The message filter is more effective in reducing the message transmission delay on a small message stream like stream A than a medium one like stream B. The local stream was only slightly affected by streams A and B when the message filter was used.

7 Conclusions

Communication networks are critical resources in NOW which may affect the performance of parallel jobs. However, a NOW is usually not dedicated to parallel jobs. Local users of workstations may run some jobs which require also the use of communication links. This paper addresses the communication interaction between parallel and local jobs. The contributions include:

- Modeling of communication interaction of parallel and local jobs in a NOW. Three representative communication patterns of parallel jobs are studied. Their performance affected by the local jobs' communication is modeled and analyzed. Experiments are conducted to verify the accuracy of the model. Analytical and experimental study shows that local jobs may degrade moderately or significantly the performance of parallel jobs.

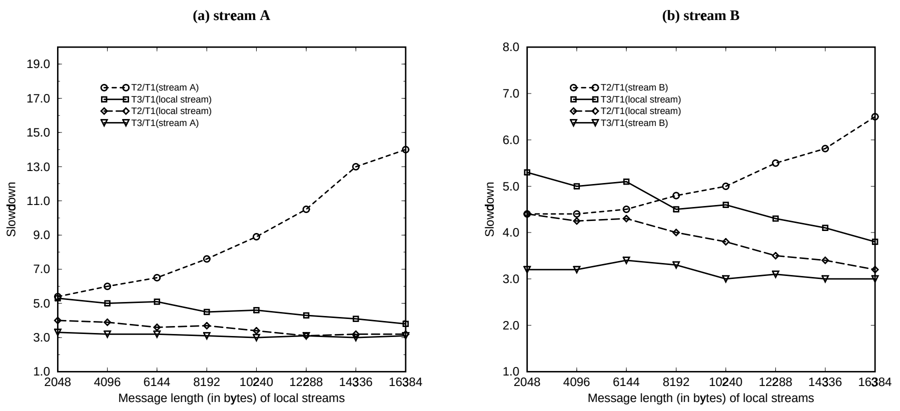


Figure 9: Experiments on the receiving side.

example, the slowdown factors were reduced to 3 for both streams A and B when the local stream increased to 16 Kbytes. The message filter is more effective for a small message stream (stream A) than for a medium message stream (stream B).

Our experiments indicate that the message filter in both sending and receiving workstations shows its effectiveness in reducing the slowdown of message passing caused by interactions in the simulated network interface.

6.4 Performance of other communication patterns

Some communication patterns are representative in parallel applications. We choose some collective communication patterns to evaluate the performance of the message filter further.

One-to-all and all-to-one are two popular collective communication patterns in parallel computing. In the one-to-all pattern, $(n-1)$ parallel messages are sent from the same node to $(n-1)$ different destination nodes. A communication interaction with a local user job in the source (sending) node would delay the one-to-all communication. In the all-to-one pattern, $(n-1)$ parallel messages from $(n-1)$ nodes are sent to the same destination node, such as a synchronization barrier operation. A communication interaction with a local user job in the destination (receiving) node would delay the receiving time of the all-to-one communication. In our experiments, we combined the one-to-all and the all-to-one to be a single operation for measurements. The source workstation first sends messages to the rest of the workstations in the system. As soon as the workstations receive the messages from the source, each sends a message back to the source workstation. The local stream was sent from the source workstation and another workstation in the system.

We measured the average time to finish this one-to-all and all-to-one combination in three ways: 1) it was performed in a dedicated environment, and T_1 was measured; 2) it was performed in a nondedicated environment when the local message stream was transmitted between the source workstation and another workstation in the system, and T_2 was measured; 3) using the double to observe its slowdown effect caused by the local stream. There are two T_2/T_1 slowdown curves (dashed lines) in Figure 8(a): the slowdown curve of stream A caused by the local stream, denoted as "stream A" and the slowdown curve of the local stream caused by stream A, denoted as "local stream". Figure 8(a) shows that, without separating the two types of messages, stream A was affected significantly by the local stream, while the local stream was moderately affected by stream A. In addition, there are two T_3/T_1 slowdown curves (solid lines) in Figure 8(a): the slowdown curve of stream A caused by the local stream, denoted as "stream A", and the slowdown curve of the local stream caused by stream A, denoted as "local stream". Our experiments show that the message filter in a sending workstation effectively reduced the slowdowns of stream A caused by the local stream. The slowdown of the local stream caused by stream A was only slightly higher.

Then, Stream B was used as the testing stream to observe the slowdown effects on medium messages caused by the local stream. Figure 8 (b) shows that stream B, which has a medium message size, has less slowdown effect than that of stream A, which has a small message size, when message types are not distinguished. Again, the double queue scheme in the message filter effectively reduced the slowdowns of stream B caused by the local stream. The slowdown of the local stream caused by stream B was moderately higher.

In summary, our experiments show that relatively small messages simulated by stream A and stream B, are easily delayed by the interaction of another message stream. The proposed message filter would significantly reduce the delay of small messages without considerably affecting the local stream on the sending side.

6.3 Performance of the message filter on receiving workstations

Similar to the above section, there are also two message streams: the testing stream (A or B) and the local stream from two workstations merging to a same receiving workstation. In our simulation, times T_1, T_2 and T_3 are the receiving times of the messages on the receiving workstation. Again, we measured T_2/T_1 to evaluate the slowdown effects without using the double queue scheme, and T_3/T_1 using the double queue scheme in the message filter in the receiving workstation. We measured the receiving times to receive 100 messages on the same receiving workstation to evaluate and compare the performance.

Figure 9 (a) and (b) show the slowdown factors of stream A and stream B affected by the local stream, respectively. The goal of this group of experiments is to investigate the effect of the message filter in a receiving workstation. Our experiments show that when message types were not distinguished, stream A or stream B sent from one workstation could be easily delayed by the local stream from another workstation. For example, the slowdown factors were up to 14, and 6 for streams A and B, respectively, when the messages size of the local stream increased to 16 Kbytes. The message filter in the network interface in the receiving workstation could effectively reduce the slowdown by distinguishing the types of the messages and separating them into two queues. For communication interactions. The interarrival times are independent and exponentially distributed.

6.2 Performance of the message filter on sending workstations

We report the simulated performance of interactions between two message streams. Both streams are from the same source workstation to two different destination workstations. First, both streams were equally treated without the message type distinction. Then the streams were treated differently by allocating 80% of the bandwidth to parallel jobs' frames when both types of frames interact. We used the receiving time on the destination workstation to measure the difference between the two methods. The receiving time was measured after the destination workstation received 100 messages from the same sending workstation. We conducted these experiments many times to get the average time.

The receiving time of a stream in a dedicated environment is denoted as T_1. The receiving time of a stream which interacts with another stream in the environment, without distinguishing the messages types in a sending workstation, is denoted as T_2. The receiving time of a stream which interacts with another stream in the environment using the double queue scheme in a sending workstation, is denoted as T_3. The ratio T_2/T_1 presents the slowdown factor caused by the interaction between the two streams without separating the two queues, which is relative to the receiving time of a single stream in a dedicated environment. The ratio T_3/T_1 presents the slowdown factor caused by the interaction between the two streams where the message filter is used in a sending workstation. This ratio is relative to the receiving time of a single stream in a dedicated environment. Figure 8 gives the ratio curves of T_2/T_1 and T_3/T_1 when the message length of the local stream varies.

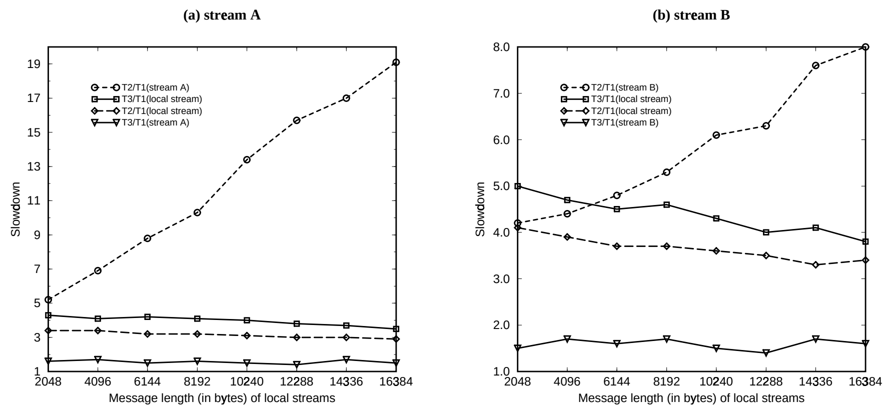


Figure 8: Experiments on the sending side.

We used stream A of smaller message size (64 bytes) and stream B of medium message size (2 Kbytes) to simulate parallel jobs' messages, the a local stream of size varying from 2 Kbytes to 16 Kbytes to simulate local users' messages. Stream A was first used as the testing stream

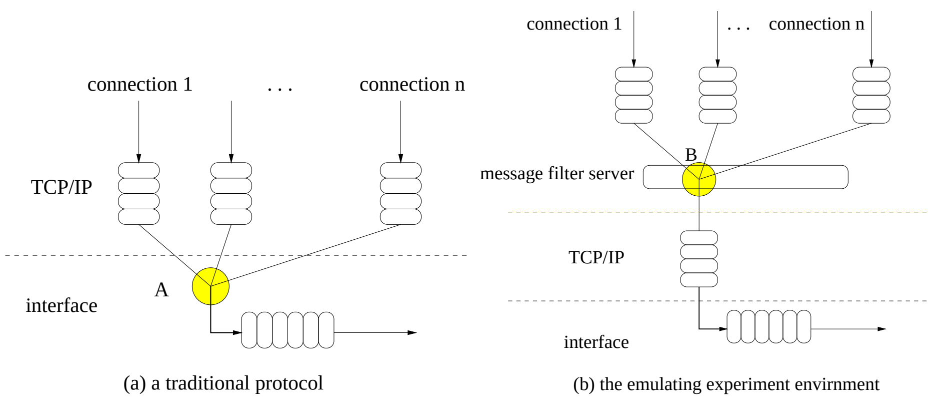


Figure 7: The traditional interface (a) and the simulator (b).

workstations connected by a FORE ASX-200BX ATM network. The structure of the simulator is shown in Figure 7. In the TCP/IP, the interaction point (Bottleneck A in Figure 7(a)) in the interface is the place where all the messages from the protocol layer will interact. As shown in Figure 7(b), all communication operations in our environment will use the message filter. Thus, all the messages will interact at the point of B in the simulated environment. In the message filter, the double queue scheme was implemented to distinguish and to prioritize the two types of messages.

To fairly compare the performance between that using the scheme and that without using the scheme, and to eliminate the software simulation overhead, in the experiments, we also developed a layer (server) in traditional protocol, which works the same as that in message filter except the fact that it does not distinguish the two kinds of messages. When the scheme is implemented in the network interface in hardware, it will deliver much better performance compared with software simulation.

We set up a message filter in each workstation of the system. The message filters set up the multiple and independent input/output queues for each of its connections. The filters also support an acknowledgement mechanism which includes the receiving window size (the credit) in every acknowledgement. It is used to control the upcoming local user frames when a receiving workstation is processing parallel jobs' frames. We used two policies in the control of the input queue and the output queue. The first policy treats all frames equally, just as in the traditional interface layer. The second policy distinguishes the frames from parallel and local jobs.

In our simulator, the frame size was set 1 Kbytes. Two kinds of message streams were studied. The first one, denoted as stream A, has an average message length of 64 bytes, which is smaller than the frame size. The second one, denoted as stream B, has an average message length of 2 Kbytes, which is larger than the frame size. A local job's stream (we call it local stream for short in the following discussions) was sent simultaneously with a testing stream. We varied the average message length of the local stream from 2 Kbytes to 16 Kbytes to show the effects of different

```
        if (all parallel frames have been received)
           Wake_up limited local users' sending with current_buf_s;
        else {
           current_buf_p = current_buf_p - 1;
           ACK(current_buf_p);
           /* send a credit to the parallel sender */
      }
      else /* the frame is from a local user */
        if (frames from a parallel job are still coming)
           ACK(THRESHOLD_recv);
           /* to limit sending rate of local jobs */
        else
           ACK(current_buf_s);
           /* normal message passing of local jobs */
}
```

The network interface should support the Acknowledgement mechanism in order to implement the above scheme. It can also be done by minor revision of the ACK mechanism of the TCP protocol to embed it.

In the BSP model, the communication of a parallel job is generally concentrated in the same phase periodically. After this communication phase, the parallel job will enter the computation phase. Thus, local jobs' messages will be affected regularly during the execution of a parallel job. Sometimes, the BSP is implemented to overlap the computation and communication. It may result in that almost all communication resources are occupied by the parallel job. In this situation, the power preservation method [6] can be used to guarantee that local jobs can go ahead in a given period.

The hardware cost of the scheme is relatively low. The overhead involved in distinguishing frame types in sending is very small. It only needs to look up a tag in the fixed position of a frame header. A counter is needed to find the fixed position, and a comparator is used to distinguish the tag. For receiving a message, the frames are distinguished by tags in their headers. The receiving network interface needs to check the tag of a frame in order to assign the frame to the right queue, and to check if all the frames of parallel jobs have arrived.

6 Performance Evaluation

6.1 Message filter

We developed a simulator using TCP/IP facilities to simulate the proposed scheme. The simulator is called a *message filter*, and was implemented in a system which consisted of 4 SUN SPARE 20 time unit , we may assign 4 to THRESHOLD_send, which means that 4 frames of a parallel job's message are sent and 1 frame of local jobs' message is sent in a unit time if both types of messages exist. Because all the frames have the same size, so at least 80% of the bandwidth is allocated to the parallel job. The larger the THRESHOLD_send value is, the more bandwidth will be allocated to the parallel job. If $THRESHOLD\_send = 1$, the bandwidth is allocated equally between the two queues. If $THRESHOLD = \infty$, local messages can only be sent when the parallel queue is empty.

The value of THRESHOLD_send should be adjusted based on local user communication demand, communication frequency of a parallel job, communication patterns of a parallel job, and the response time requirements of parallel jobs.

5.3 Adjusting the sending rate from receiving workstations

In order to make a parallel program run efficiently, only giving a higher priority to the parallel job to send frames is not enough. When some parallel job's frames are being received on a receiving workstation, it is necessary to notify the local user sender to slow down or even stop sending the frames to the same node at the moment. This will allow the parallel job's frames go through as soon as possible.

We adopt a credit-based approach [10] in our system to adjust the sending rate of local user jobs to a receiving workstation. The credit is a positive integer value which tells how many frames can be sent at most in the next step. It reflects the current available size of the receiving buffer. Initially, the credit for local users' frames is the receiving buffer size. If a workstation receives frames from both a parallel job and local jobs, it will adjust the credit for the local jobs to be the value of THRESHOLD_recv, and send this credit back to the workstation where the local jobs run through an acknowledgement to control its sending. If THRESHOLD_recv is set to 0, it tells the sender to stop sending frames. After the communication of the parallel job finishes, the credit for local users will be reset by assigning the current receiving buffer size for local user jobs, and resume the message-passing of a local user job as normal. The procedure is illustrated as follows, where the value of THRESHOLD_recv is used to adjust the sending rate of a local job from a remote machine. These operations of the procedure are done before a frame reaches the receiving queue of either parallel jobs or local jobs.

```
CONSTANT THRESHOLD_recv;
    /* used to limit message flow of a local job */
int current_buf_p = length of the receiving queue for parallel jobs;
int current_buf_s = length of the receiving queue for local jobs;
while (1) {
    receive a frame;
    if ( this frame is from a PARALLEL job) {
```

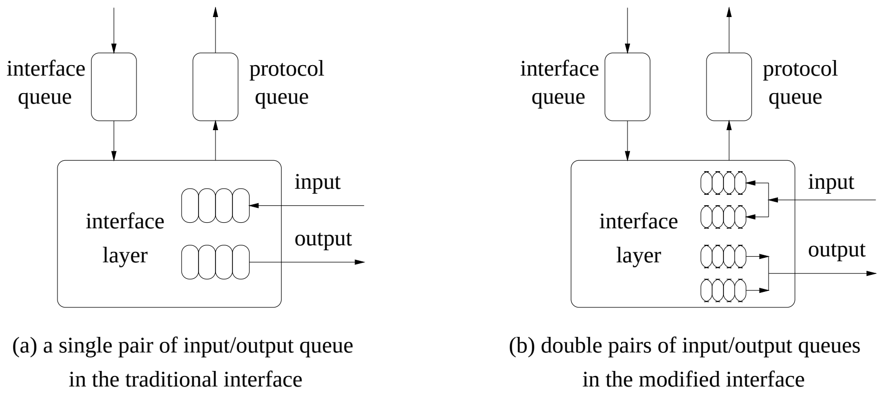

interface
queue
protocol
queue
interface
layer
input
output
(a) a single pair of input/output queue in the traditional interface
interface
queue
protocol
queue
interface
layer
input
output
(b) double pairs of input/output queues in the modified interface

Figure 6: Input/output queues in the network interfaces.

other for local user messages. The sending order of the double queues determines the bandwidth allocation ratio between the two queues. Because we give parallel jobs' messages a higher priority for transmission, their queue will be served more frequently than the other queue. The simplest way to allocate the bandwidth is not to send messages in the queue for local jobs until the queue for parallel jobs is empty. It has two disadvantages. First, it is unfair to local jobs whose performance may be affected considerably when a parallel job exists. Second, it may cause starvation for local jobs. We propose a simple and efficient threshold-based algorithm to attack the above problems.

```
CONSTANT THRESHOLD_send;
    /* this used for bandwidth allocation */
int count = 1;
while (1) {
    if (parallel queue != NULL) {
        send one frame from the parallel queue;
        count = count + 1;
        if (count > THRESHOLD_send) and (local queue != NULL) {
            send one frame from the local queue;
            count = 1;
        }
    } else
        if (local queue != NULL)
            send one frame from the local queue;
}
```

The algorithm shows that the bandwidth allocation ratio is determined by the value of THRESHOLD_send, which is the maximum number of frames to be sent from the parallel queue within a time unit. For example, if the physical link is able to transmit 5 frames continuously within a parallel job performance degradation.

We propose a scheme called Double Queue Scheme to reduce the interaction effects. It extends the traditional network interface in two ways. First, a traditional network interface uses a single FIFO queue for input messages and a single FIFO queue for output ones (Figure 6(a)). Either queue does not distinguish the message type. Small and urgent messages from parallel jobs may be blocked by large and non-urgent messages from local user jobs. Consequently, it may degrade the performance of parallel jobs significantly. We first divide the input/output queue in the network interface into two separate queues respectively: one for parallel jobs' messages and the other for local users' messages. Thus, input and output messages are distinguished by the two types (Figure 6(b)). Second, we set different priorities to different queues upon the demand of applications. Parallel jobs' messages could be given a higher priority for the message-passing activities. Besides giving a higher priority to the queue for parallel jobs' messages in a sending workstation, we use acknowledgements from a receiving workstation to control the local users' message flow coming from other workstations. All the traditional protocol features are retained. When there is no messages from a parallel job, the interface works the same as the traditional one. If both types of messages exist, parallel jobs' messages are given a higher priority to be transferred as long as they do not affect the performance of local jobs' communications very much.

The double queue scheme is different from the Quality of Service (QoS) provided in some switch based networks such as ATMs. First, it is a general approach and can be implemented in any other network interfaces such as Ethernet. Second, it is flexible without additional overheads. In a QoS network, usually the bandwidth is statically allocated. Dynamic QoS has been proposed by some research projects. But the additional management overhead can not be ignored. Finally, there is no admission control in our scheme. In fact, it is sort of a best effort service with two priority levels.

Giving a priority to parallel jobs does not mean local jobs cannot go ahead in a given time. There are two ways to keep local jobs going. First, some resources are allocated to local jobs within the given period while parallel jobs have the priorities to use these resources. Second, using the power preservation method [6], we can guarantee the local jobs obtain the system resources they need in a given period.

Two issues must be addressed. The first is how to allocate the available bandwidth on a sending machine between the two types of jobs. The second is how to handle messages on a receiving machine and how to adjust the sending rates of local jobs' messages coming from other machines. We will discuss them separately in the following subsections.

5.2 Bandwidth allocations in a sending workstation

When entering the network interface, messages have been divided into the same size frames. There are two output frame queues on a sending workstation: one for parallel job messages and the buffers managed by the kernel's pool. And then, the socket layer calls the TCP output routine and transfers the pointer of the mbuf chain to it. The TCP output routine calls the IP output routine and provides it with the pointer of the mbuf chain as well. When the IP output calls the interface output, the mbuf chain is added to the end of the output queue for the interface. From the socket layer to the interface queue, no data is copied and only the data pointer is transferred. If the interface is not currently busy, the interface's "start output" routine will be called. It copies the data in the mbuf chain to the transmit buffer of the interface; otherwise, the mbufs in the front of the queue will be processed. The output queues in most modern network interfaces such as Ethernet and ATM are FIFO queues. Similarly, to receive a message, the network interface is triggered asynchronously by the hardware interrupts. The interface input routine reads data from the device into an mbuf chain. A software interrupt is scheduled to cause the IP input process routine to be executed. The interface input queue is also a FIFO queue. The length of this queue is limited. When the IP input routine is executed, it will process each IP datagram on its input queue until the queue is empty. Because the input processing use an asynchronous software interrupt to transfer data from the interface to the IP layer, when the input data will be processed is not guaranteed. When the interface buffer is full, the coming segments will be dropped. So, if some segments which belong to some local jobs' messages occupy the interface buffers, the coming segments which belong to some parallel jobs will be dropped and need the retransmission. In another word, in a NOW, the network does not distinguish the messages from a parallel job or a local job. So they will affect each other.

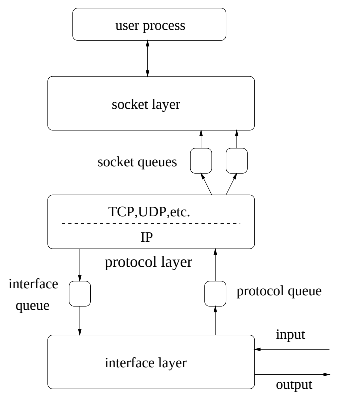


Figure 5: Communication layers and buffers in TCP/IP.

From the application standpoint, most local users' communication are `rlogin`, `telnet`, `ftp`, and WWW, etc. They usually involve two machines. A little performance loss of these operations will not affect the whole job significantly because the response times of these operations are much longer than the response time of a parallel job. In contrast, a parallel job always involves more machines in general. A small communication delay in a parallel process may result in a large difference between two platforms was also insignificant. However, Configuration $N8$ was worse than Configuration $N4$ because the values of β were unchanged in both platforms but the values of $\delta(N)$ were different. This is consistent with the modeling of many-to-one communication pattern.

IS is a program that has all-to-all communication pattern. An instance of IS which performs 10 rankings of 2^{21} integer keys in the range of $[0, 2^{10}]$ was used here. The message size was small but the frequency of exchanging messages was high. So β for the program was very high. By the formula (4.19), the model shows that η for IS increased not significantly with the increase of α compared with MG and FT. This is consistent with the experimental results (see Figure 3 and 4). Overall, the effect of local communication on IS was not great ($< 25\%$). Figure 4 presents the communication delays for both EP and IS programs.

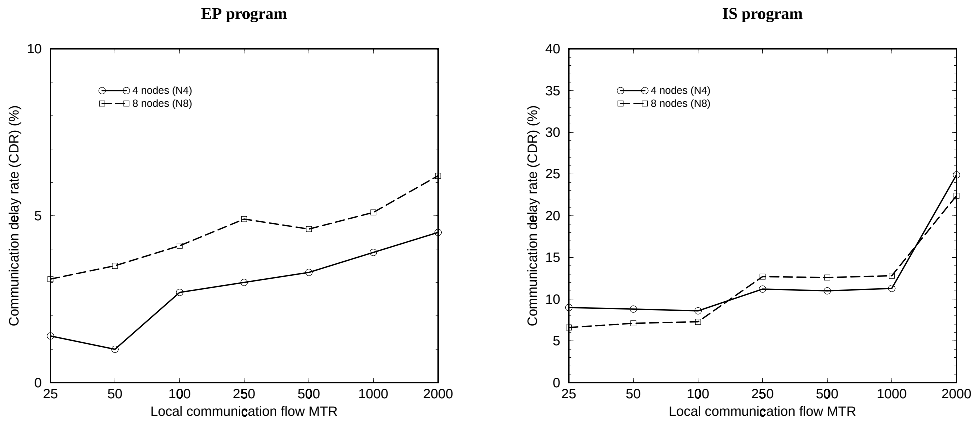


Figure 4: Local communication effects on program EP and IS

5 Double Queue Scheme

5.1 Rationale

In the past sections, we model and analyze the communication interactions between parallel and local jobs. We find the interaction (with local jobs' communication) affects the performance of various parallel jobs moderately and significantly. In the following sections, we propose a scheme to reduce the negative effects of communication interactions between parallel and local jobs. It increases the performance of parallel jobs while incurs only a slight overhead on communications of local jobs.

The interaction and its effect result from the communication protocols and network interfaces. In general, TCP/IP is the most popular protocol used in practical systems. Figure 5 illustrates the communication layers and buffers in a typical system using TCP/IP. To send a message, a user process uses some communication library functions to send the message to the socket layer. The socket layer copies the message to a memory buffer called *mbuf* chain. Mbufs are structured of β by a factor of 2. More nodes involving in communication lead to more congestions. However, η reduced. This can be easily derived from the model. By (4.19), the coefficient of α for $N4$ divided by that for $N8$ is:

$$\frac{7 \times \delta(2) \times \delta(7) \times \beta_8}{3 \times \delta(2) \times \delta(3) \times \beta_4} = \frac{7 \times \delta(7) \times 2\beta_4}{3 \times \delta(3) \times \beta_4} = 1.34,$$

where β_4 and β_8 are the β of MG on $N4$ and $N8$ platforms respectively. From above discussion, $\beta_8 = 2\beta_4$. The ratio of the two co-efficients is greater than 1. It indicates that for a given local flow (α), it affects the performance of MG on $N4$ greater than that on $N8$.

The program structure of FT is similar to MG. The program consists of 6 iterations. For each iteration, there are two communication phases separated by computation phases. The main communication pattern is transpose. For $N4$ and $N8$ NOW configurations, the numbers of messages sent were 186 and 826, the total bytes of messages sent were 336MB and 392MB respectively. Figure 3 shows that the interaction results of FT is very close to that of MG. The difference of FT from MG is that FT has a constant computation size between two communication phases in each iteration while MG decreases its computation size by 2 units iteration by iteration to its minimum one and then back to its initial size. Another difference is that the average message size of FT is much larger than that of MG. So its β is large. By the formula (4.19), if α is small, the local traffic flows affect the performance of FT not too much. Figure 3 shows that when α was 50, 100, 250, its effect was 17%, 18%, and 18.2% respectively on $N4$, and 8.6%, 9.0% and 9.4% on $N8$. When α increased to 500, 1000, and 2000, η increased almost linearly.

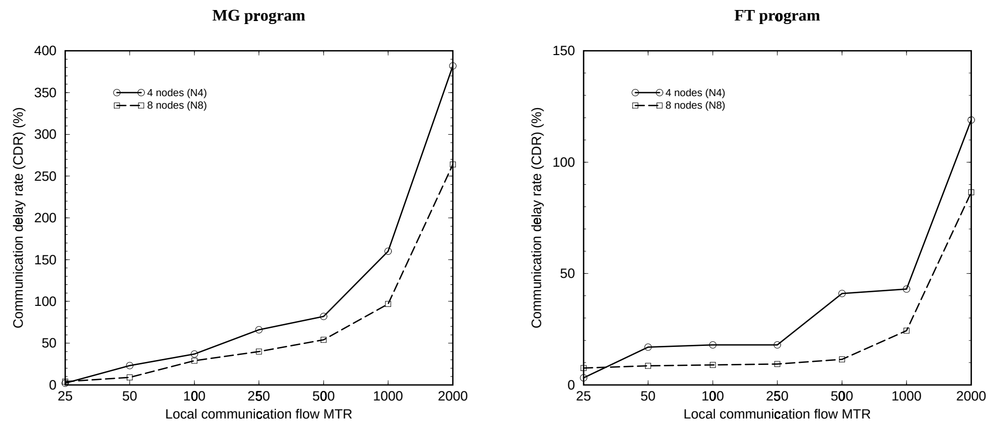


Figure 3: Local communication effects on program MG and FT

The structure of EP program is very simple. It is composed of only one computation and one communication. The communication pattern is all-to-one, and is only performed once at the end of the program. The message length is extremely short. The effect of local communication on it was very small, and increased very little as the increase of local traffic's α (see Figure 4). The

The analytical model indicates that the interaction effect between two different flows are similar to the effect of two identical flows. The interaction effects between a flow of type A and a flow of type B, two As and one B, one A and two Bs, two As and two Bs, were measured separately. Table 2 lists the average δ slowdown of flows types A and B when they interacted. The measured results are reasonably close to the modeling results. For example, when there were one A and one B, the average slowdown for the flow of type A was 92% and 90% for the flow of B, while the analytical value was 88%.

flow B	*flow A*			
	1		2	
	A	B	A	B
1	0.92	0.90	0.83	0.85
2	0.81	0.85	0.75	0.74

Table 2: Interaction of two type flows A and B

4.3 Interactions between parallel and local traffic flows

Four parallel programs MG, FT, EP, and IS were run on two NOW configurations: $N4$ and $N8$.

A $128 \times 128 \times 128$ MG was run on the two NOW configurations. The number of messages sent by 4-node NOW was 816 and the total bytes sent and received were 24.5MB. For the 8-node NOW, the number of messages was 1472 and total bytes were 48.8MB. The MG program follows the BSP model. After each computation, the node P sends messages to node $(P+i)$ mod N $(i = 1, ..., N-1)$ where N is the number of workstations, here it is either 4 or 8. For each platform, local traffic flows were added and their α were varied from 25 to 2000 messages per second. The packet size was 8K bytes. Figure 3 gives the performance effects on MG by the local traffic flows. The x-axis is the α of the local traffic, and the y-axis is η for MG. The curves in Figure 3 show that with the increase of the local flows' α, its effect on the performance of MG increases. In addition, its increase is proportional to the increase of α. For example, when $\alpha = 500$, η was 82%, and when $\alpha = 1000$, η became 160%. When α further increased to 2000, the ratio jumped to 382%. This result is consistent with our modeling results. By the analytical model, MG is the program which has each-to-many communication pattern. In addition, in MG, $m = N-1$, and $n_j^i = n = N-1$ for any i and j. Substituting n into formula (3.18), η is proportional to α:

$$\eta = \frac{1}{(N-1)\delta(N-1)\delta(2)} + \frac{1}{(N-1)\delta(N-1)\delta(2)} \frac{\alpha}{\beta} - 1. \qquad (4.19)$$

Further we compare the curves of $N4$ and $N8$. The effects of local traffic on MG running on $N8$ was less than that on the $N4$ platform. Comparing with the $N4$ platform, when the same size MG ran on $N8$, the computation size on each node decreased to half. This resulted in the increase

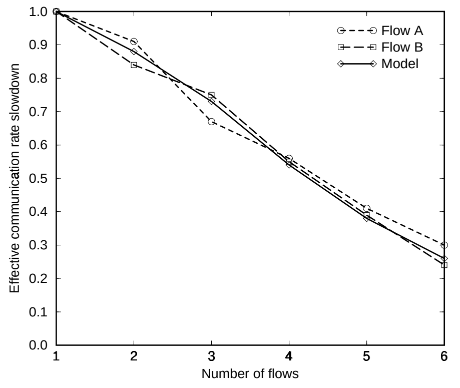


Figure 2: Independent traffic flow interactions.

size and the mean of the message interarrival time were adjustable. PP ran at each workstation to emulate the local users' communication such as `ftp`, `telnet`, `rlogin`, etc. All experiments were conducted for more than 5 times, and the average results are reported in the paper.

4.2 Independent traffic interactions

PP was used to generate two types of traffic flows, types A and B, to evaluate the communication interaction between several independent flows. The message size of type A was 8 Kbytes, and 512 bytes of type B. Type A represented large message size traffic, and B for the small one. We set $\alpha = 2000$ messages per second, which was large enough to generate the congestion when two or more such flows interacted. Through these experiments, we measured the effective communication ratio δ and compared it with the analytical results from the formula (2.2).

We measured the transmission times of the messages more than 10 times and averaged them. Suppose the transmission time is t seconds for a fixed size message. The effective communication ratio δ is determined by: $\delta = \frac{\frac{1}{t}}{\gamma}$. The measured latency for a message of type A was about 500 μs when it was the only flow on the network. However, when two flows of type A sent messages from different workstations to the same destination, communication interaction occurred at the receiver input link. The latency for each flow increased to 550 μs. To avoid measuring γ, the slowdown of δ was used to reflect the effects of interaction. The slowdown is the ratio of δ when the interaction occurs with respect to that without interaction. Consequently, the slowdown for the above flow is 91%. The number of flow A was increased from 2 through 6. The same experiment was done with the flow of type B as well. The results of the experiments were compared with the analytical ones derived from the interaction model (formula 2.2). Figure 2 gives the δ slowdown for flows of type A and B, and the analytical model results respectively when the number of flows increased. It indicates that the slowdown changes for both flows of types A and B are very close to the model results (the difference is within 6%).

Further assume that any workstation in the NOW receives the same number of messages, denoted as p. That is

$$n^i = p \qquad (for \;\; 1 \leq i \leq N). \tag{3.17}$$

By (3.16) and (3.17), (3.15) can be simplified into

$$\eta = \frac{1}{p\delta(p)\delta(2)} + \frac{1}{p\delta(p)\delta(2)} \frac{\max_{i=1}^{N} \alpha_i}{\beta} - 1. \tag{3.18}$$

For given p and β, η is determined by the strongest local flow ($\max_{i=1}^{N} \alpha_i$) and is positively proportional to it. When $\max_{i=1}^{N} \alpha_i$ is fixed, the larger β is, the larger η is, which is similar to that of the many-to-one pattern.

4 Model Evaluation

In the last section, we model the communication interactions and analyze quantitatively the effects of the interactions on the performance of parallel jobs. This section evaluates and verifies the analytical results through measurements.

4.1 Evaluation environment and parallel applications

The experimental NOW platform used for evaluation consisted of 8 Sun UltraSparc workstations connected by one FORE ASX-200BX ATM network with FORE SBA-200 adapters. This is a typical NOW environment supported by high speed networks. We used two different NOW configurations: one was composed of 4 workstations ($N4$) and the other of 8 workstations ($N8$). We varied the workstation number in order to evaluate its effects on interactions. The operating system on each workstation was Solaris 2.5. The communication protocol was TCP/IP.

Four programs from the NAS parallel benchmarks [3]: MG, FT, EP, and IS, were used. MG (Multigrid) is a 3D multigrid benchmark program for solving a discrete Poisson problem. This application exercises various lengthy messages and the communication patterns are highly structured. The FT uses FFT on an array to solve a 3-dimensional partial differential equation. Its communication patterns are transpose. Kernel EP (Embarrassing Parallel) executes a loop, where a pair of random numbers are generated and tested for whether Gaussian random deviates can be made from them by a specific scheme. Each parallel process only participates in the communication of a 44 byte packet once at the end of processing. Kernel IS (Integer Sort) performs 10 rankings of 2^n integer keys in one specific range. Message-passing is frequent.

All these programs were written in C and ran under PVM [16]. The option was set for each application to allow it to send/receive messages by TCP/IP. We wrote a point-to-point communication program called PP in C using TCP/IP. It sent fixed-size messages randomly. The message

If there are no local user communication traffic flows, and parallel traffic flows are managed properly without congestions at any point in the switch, the average communication time from workstation i to workstation A^i_j is determined by

$$t(i, A^i_j) = \frac{M}{\frac{\gamma}{n^i_j}} = \frac{Mn^i_j}{\gamma}.$$

Because workstation i sends messages to m workstations sequentially, the average total communication time for workstation i is

$$t_i = \sum_{j=1}^{m} t(i, A^i_j) = \frac{M}{\gamma} \sum_{j=1}^{m} n^i_j.$$

The average total communication time of the system is

$$t = \max_{i=1}^{N} t_i = \frac{M}{\gamma} \max_{i=1}^{N} \sum_{j=1}^{m} n^i_j. \tag{3.12}$$

When there are local traffic flows, congestions may occur at senders' output links and receivers' input links (see Figure 1(c)). Consequently, by (3.8), the average communication time for workstation i sending messages to workstation A^i_j is:

$$T(i, A^i_j) = \frac{M}{\gamma \delta(n^i_j)\delta(2)(\frac{\beta}{\alpha_i + \beta})} = \frac{M}{\gamma \delta(n^i_j)\delta(2)}(1 + \frac{\alpha_i}{\beta}),$$

and the average total communication time for workstation i is

$$T_i = \sum_{j=1}^{m} T(i, A^i_j) = \frac{M}{\gamma \delta(2)}(1 + \frac{\alpha_i}{\beta}) \sum_{j=1}^{m} \frac{1}{\delta(n^i_j)}. \tag{3.13}$$

The average total communication time of the system is

$$T = \max_{i=1}^{N} T_i. \tag{3.14}$$

By (3.12) and (3.14), η for the each-to-many pattern is:

$$\eta = \frac{T - t}{t} = \frac{\max_{i=1}^{N} ((1 + \frac{\alpha_i}{\beta}) \sum_{j=1}^{m} \frac{1}{\delta(n^i_j)})}{\delta(2) \max_{i=1}^{N} \sum_{j=1}^{m} n^i_j} - 1. \tag{3.15}$$

For given i, workstation A^i sends messages to m other workstations, denoted by A^i_j $(1 \leq j \leq m)$. Suppose each of these m workstations receives the same number of messages, denoted as n^j, from all workstations in the NOW, we have

$$n^i_j = n^i \qquad (for \;\; 1 \leq j \leq m). \tag{3.16}$$

3.2 One-to-Many communication

Without losing generality, we can assume workstation P sends messages to other n workstations with the MTR of β. In addition, there is a local communication traffic flow in workstation P with the MTR of α (see Figure 1(b)).

When $\alpha = 0$, the optimal time, t_i, for sending messages from workstation P to workstation i is

$$t_i = \frac{M}{\beta}.$$

When local user traffic flows exist, network contention would degrade the MTR of the links by a factor of $\delta(2)$ (see Figure 1(b)). Consequently, the average communication time to workstation i is determined as follows:

$$T_i = \frac{M}{\delta(2)\gamma(\frac{\beta}{\alpha+\beta})} = \frac{M}{\gamma\delta(2)} + \frac{M\alpha}{\beta\gamma\delta(2)}. \tag{3.8}$$

Since we assume that the network does not support multicast, the one-to-many communication is performed sequentially or concurrently. Thus the average optimal communication time is:

$$\sum_{i=1}^{n} t_i = \frac{Mn}{\beta}, \tag{3.9}$$

and the average affected communication time is:

$$\sum_{i=1}^{n} T_i = \frac{Mn}{\gamma\delta(2)} + \frac{Mn\alpha}{\beta\gamma\delta(2)}. \tag{3.10}$$

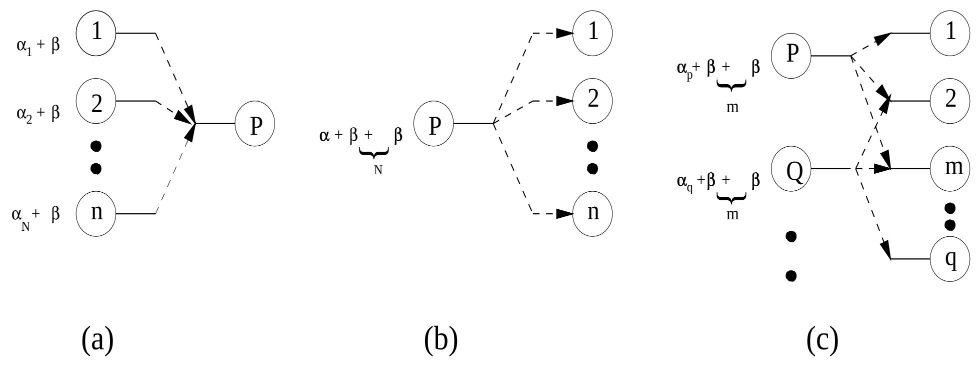
By (3.9) and (3.10), we have η for such a pattern:

$$\eta = \frac{\beta}{\gamma\delta(2)} + \frac{\alpha}{\gamma\delta(2)} - 1 = (\alpha+\beta)\frac{1}{\gamma\delta(2)} - 1. \tag{3.11}$$

Based on (3.11), when parallel jobs have such a communication pattern, its performance is affected equally by α and β. For given any one of both MTRs, η is positively proportional to the other one. It is different from the many-to-one pattern where η decreases when β increases. No matter how α and β vary, if their sum is fixed, their overall effect on the performance is determined.

3.3 Each-to-many communication

Assume there are N workstations in the system, each workstation sends messages to m specific workstations. Examples of such communication patterns are *neighbor* communications and *transpose* communications. Let A_j^i denotes the jth $(1 \leq j \leq m)$ workstation to which workstation i sends messages. Workstation A_j^i receives n_j^i messages by all other nodes at one time.



$\alpha_1 + \beta$
$\alpha_2 + \beta$
$\alpha_N + \beta$
1
2
n
P
$\alpha + \underbrace{\beta + \quad \beta}_{N}$
$\alpha_p + \underbrace{\beta + \quad \beta}_{m}$
$\alpha_q + \underbrace{\beta + \quad \beta}_{m}$
Q
m
q
(a)
(b)
(c)

Figure 1: Three communication patterns and possible congestion points.

Optimally, if there is no local traffic ($\alpha_i = 0, 1 \leq i \leq n$), n traffic flows are managed properly without congestion, and each one transmits at the rate of $\frac{\gamma}{n}$, then it would take averagely t_i time unit at workstation i to finish sending M messages:

$$t_i = \frac{M}{\frac{\gamma}{n}} = \frac{Mn}{\gamma}. \tag{3.4}$$

Otherwise, two types of congestion may occur: (see Figure 1(a)) (1) parallel and local user traffic flows contend for the output link at each workstation; (2) n traffic flows cause congestion at workstation P's input link. Consequently, by (2.2) and (3.3), the average communication time becomes:

$$T_i = \frac{M}{\frac{\alpha_i+\beta}{\sum_{j=1}^{n}\alpha_j+n\beta}\delta(n)\gamma(\frac{\beta}{\alpha_i+\beta})} = \frac{M}{\gamma\delta(n)}(\sum_{j=1}^{n}\frac{\alpha_j}{\beta}+n). \tag{3.5}$$

By (2.1), (3.4), and (3.5), we have the η of such communication pattern:

$$\eta = \frac{\max_{i=1}^{n} T_i - \max_{i=1}^{n} t_i}{\max_{i=1}^{n} t_i} = \frac{1}{n\delta(n)\beta}\sum_{j=1}^{n}\alpha_j + (\frac{1}{\delta(n)} - 1). \tag{3.6}$$

Formula (3.6) indicates that, for given β, η is positively proportional to the sum of α_j ($1 \leq j \leq n$). This means that the larger the sum, the greater their effects on the performance of parallel jobs. In contrast, if the sum is given, η is negatively proportional to β. It indicates that a stronger flow of a parallel job (β is large) will be affected less by the local jobs' traffic flows. For the special case when $\alpha_j = \alpha$ ($1 \leq j \leq n$), (3.6) can be simplified to

$$\eta = \frac{1}{\beta\delta(n)}\alpha + \frac{1}{\delta(n)} - 1. \tag{3.7}$$

Thus, for a given β, η is positively proportional to α.

When several traffic flows merge, congestions may occur. Assuming the slow-start mechanism of TCP [18] is used, and the n flows of the same type start simultaneously. So the mechanism increases the congestion window size of a flow by one segment after successfully transmitting a segment and receiving an acknowledgement for the flow. It opens the window exponentially. In TCP/IP, when the window size is increased to one specific threshold, the slow-start phase is over, and congestion avoidance takes over, which gives an additive increase in the window size. For simplifying the problem, we assume congestion is reached during the slow-start phase. The window is doubled until the congestion occurs. Then it sets the window size to be one segment again. The above procedure repeats. So the time from the start to the point of reaching congestion during slow-start phase is: $\lfloor \log_2 \frac{\gamma}{n} \rfloor$. Here, we ignore the round-trip delay because it is very small in the network considered, and ignoring it would significantly simplify the discussions while not losing much accuracy. As a result, the maximum number of messages that can be transmitted during this time interval is: $\lfloor \log_2 \frac{\gamma}{n} \rfloor \gamma$, and the number of effective transmitted messages is:

$$n \sum_{i=0}^{\lfloor \log_2 \frac{\gamma}{n} \rfloor - 1} 2^i.$$

Consequently, by the definition of δ, the average δ can be quantified by the following formula:

$$\delta(n) = \frac{n \sum_{i=0}^{\lfloor \log_2 \frac{\gamma}{n} \rfloor - 1} 2^i}{(\lfloor \log_2 \frac{\gamma}{n} \rfloor)\gamma} = \frac{\frac{n}{\gamma}(2^{\lfloor \log_2 \frac{\gamma}{n} \rfloor} - 1)}{\lfloor \log_2 \frac{\gamma}{n} \rfloor}. \tag{2.2}$$

Formula (2.2) indicates that in general δ is a decreasing function of n. When the number of flows, n, increases, δ decreases, which indicates that the effective network utilization decreases. When $n \to \gamma$, $\delta(n) \to 0$. It means that the network is jammed when the number of traffic flows approaches to the maximum MTR.

3 Analysis of Communication Interactions

In this section, the effects of local communication traffic on the performance of parallel jobs with the three patterns of communications are quantitatively analyzed.

3.1 Many-to-one communication

Figure 1(a) shows that n processes from n workstations send messages to target workstation, P. The MTR from each of n workstations is β. Further assume that the MTR of local communication at workstation i be α_i $(1 \leq i \leq n)$. So at workstation i, its total MTR is:

$$\alpha_i + \beta. \tag{3.3}$$

The Communication Delay Ratio η quantitatively reflects the communication effects of local jobs to the communication performance of a parallel job.

We assume that there is only one local user traffic flow in each workstation. We use α to denote the MTR of this local user traffic flow. We assume there is only one parallel job running on the NOW. Because the parallel programming model is BSP model, the interarrival times of the communication events in a parallel traffic flow are determined by the computation times. Two cases are considered here. In the first case, the computation time is fixed in each iteration, so the communication events occur periodically. Assume that the computation time is t_1. Then the communication events occur after every t_1 time. The interarrival time is a constant t_1. Its mean is t_1. In the second case, the computation time varies from different iterations. Assume the computation times follow an exponential distribution, and its mean is provided. We use β to denote the MTR of parallel jobs traffic flows. It can be determined by the mean of the traffic flows.

In this paper, the network maximum MTR is referred to as γ. Suppose the network bandwidth is B Mbps, and the average message size κ bytes. The maximum MTR over the network is calculated by:

$$\gamma = \frac{B}{\kappa \times 8}.$$

For example, if the link speed $B = 155$ Mbps (OC-3), and $\kappa = 8192$ (8K) bytes, the γ over the link equals to 2480 messages per second. The actual measured γ for this message size may be less due to the the various communication software overhead.

Table 1 summaries all parameters used in the following discussion.

N	The number of workstations in the system
B	Network bandwidth (Mbps)
κ	Average message size
M	The total number of messages sent by parallel and local jobs
t	Communication time of parallel job without interactions
T	Communication time of parallel job with interactions
n	The number of traffic flows
δ	Effective Communication Ratio
η	Communication Delay Ratio
MTR	Message Transmission Rate
γ	The maximum MTR over a network
α	The MTR of local users flows
β	The MTR of parallel jobs flows

Table 1: Parameters used to model the communications

Communication patterns of a parallel job is another factor that must be considered for performance. In this paper, three representative types of communication patterns in parallel programs are considered:

- *many-to-one:* Some processes send messages to one specific process.
- *one-to-many:* A process sends messages to some other processes.
- *each-to-many:* Each process sends messages to a set of processes.

Three major parameters are used here to model the traffic flows and the flow interactions. They are defined as follows.

- Message Transmission Rate. The Message Transmission Rate (MTR) is defined as the average number of messages of one traffic flow transmitted per second over a network link.

Let C_{ij}^t denotes that at time t, workstation W_i begins to send messages to workstation W_j. Communications C_{ij}^t and C_{pq}^t interact to each other in a switch if and only if $i = p$ or $j = q$. In addition, if the MTRs of C_{ij}^t and C_{pq}^t are α_1 and α_2 respectively, and $\alpha_1 + \alpha_2 > \gamma$ where γ is the maximum MTR the link can offer, the link congestion occurs. It is called as output link congestion if $i = p$, or input link congestion if $j = q$.

When the congestion occurs, flow control mechanisms of the communication systems try to back off the traffic flows. For example, TCP (BSD 4.3-Reno) [15, 18] uses *slow-start* mechanism, while Ethernet protocol adopts Carrier Sense, Multiple Access with Collision Detection (CSMA/CD) [14] for traffic control. As a result, a certain amount of the bandwidth is wasted.

- Effective Communication Ratio, δ. The effective communication ratio, denoted as δ, is defined as the ratio between the effective amount of message transmission when congestion occurs and the maximum amount of the message transmission the physical link can offer.

So the MTR becomes $\delta \cdot \gamma$ when the congestion occurs. Obviously, δ is the function of the number of traffic flows.

- Communication Delay Ratio, η. Assume t is the communication time of a parallel job without interactions with local user communications, and T is the time with the interactions of local user communications. The Communication Delay Ratio caused by the local user communications is defined as:

$$\eta = \frac{T - t}{t} \times 100\%. \tag{2.1}$$

receiving workstation. The blocking at both ends would affect the performance of parallel jobs more significantly than that of local user jobs because the communication time of a parallel job is more critical than that of a local job. In order to effectively improve communication performance of a parallel job without considerably affecting the local job communication time, it is necessary to distinguish and prioritize messages between the both types of jobs at the network interface level. We propose a double queue scheme in the network interface. By distinguishing and prioritizing the communication messages between the two types of jobs, this scheme reduces the communication interaction effects significantly.

The rest of this paper is organized as follows. Section 2 gives a model of the communication interactions, and Section 3 analyzes the communication delay for parallel jobs caused by the interaction. In Section 4, we verify the analytical model through the experimental measurements. Section 5 proposes the double queue scheme in the network interface to reduce the effects of communication interactions. Section 6 evaluates the performance of the scheme. Finally, Section 7 summarizes the paper.

2 Communication Interaction Model

A NOW can be abstracted as a connected graph $HN(W, Net)$, where

- $W = \{W_1, W_2, ..., W_N\}$ is a set of workstations (N is the number of workstations).
- Net is a standard interconnection network.

If a NOW consists of a set of identical workstations, the system is homogeneous; otherwise, it is heterogeneous. In order to focus on communication issues, we consider a homogeneous system here. We further assume that Net is a switch-based network. Each workstation W_i ($1 \leq i \leq N$) uses the same type of adaptor to connect to the switch. The connection is bidirectional, the network latency for any pair of workstations is identical. This is a representative NOW environment.

The parallel job considered in this paper follows the bulk synchronous parallel (BSP) programming model which is the structure of many parallel applications [13]. The job consists of one process per workstation on a fixed number of workstations throughout the execution. The size of each process is similar. The process completes after a number of loops. In each iteration, phases of local computation alternate with phases of communications and synchronizations. The basic structure of the model is as follows:

```
Loop
  simultaneous tasks for local computation;
  communications for data exchange or synchronization;
end Loop.
```

1 Introduction

The wide availability of workstations and the improving speeds of networks have made Networks of Workstations (NOWs) become important platforms for parallel computation. Usually a NOW is not dedicated to parallel jobs. Local users may do some jobs which involve communications on their workstations. Consequently, the parallel and local jobs will share not only the processors but also the networks, thus affecting the performance of each other. This paper focuses on the communication interaction and how to adjust and schedule the communications of parallel and local users' jobs on a NOW environment.

To address the communication issue on a NOW, current research can be classified into two main directions. The first is to provide performance insights and to identify overhead sources of communications (e.g. [4, 9, 11]) to try to find the performance bottlenecks in the communication systems. The second is to propose and implement communication models for the purpose of overhead reduction (e.g. U-net [7, 8], Fast Message [12], and remote access model [17]). There are even several new network architectures based on high-speed switches proposed, for example, the Fiber Channel Standard [1], High Performance Parallel Interface (HIPPI) [2], and Fiber-Optic LANs using Asynchronous Transfer Mode (ATM) protocols. But only improving the network speed does not address the issue of communication interaction thoroughly.

Most of local user jobs' communication involves using remote machines (`rlogin` and `telnet`), file transfers (`ftp`), and WWW operations. They are not very sensitive to response time, and only involve two machines. In contrast, a parallel job involves more machines in general, and some barrier operations should be performed during its lifetime. A local job which fetches a large remote file using `ftp` may delay the execution of a parallel job significantly if the parallel job is performing a barrier at that time. In the worst case, the whole waiting time for each synchronized process in the parallel job may be as long as the time of the `ftp` operation.

We first examine the communication interactions between parallel jobs and local jobs on a nondedicated NOW. TCP/IP is a popular communication protocol widely used. Based on this protocol, the interaction process is quantitatively modeled, and the communication delays of a parallel job caused by the interaction are measured. Measurement results on a network of workstations support the analytical model. The performance study shows that the interaction affects the performance of parallel jobs significantly.

Secondly, we propose a scheme to reduce the negative effects of interaction. In general, the network interface below the TCP/IP protocol forms a communication bottleneck during interactions, because a standard network interface has a single input/output queue, and is not able to distinguish communication requests from both types of jobs. The interactions occur at both sending and receiving workstations. On the sending workstation, because there is only one FIFO sending buffer queue in the network interface (such as Ethernet adapters, FORE ATM SBA-200 adapters), parallel jobs' messages may be blocked by local jobs' messages. Similar blocking occurs on the

Characterizing and Scheduling Communication Interactions of Parallel and Local Jobs on Networks of Workstations [*]

Yingfei Dong
Department of Computer Science, University of Minnesota
Minneapolis, MN 55455

Xing Du, and Xiaodong Zhang
Department of Computer Science, College of William and Mary
Williamsburg, VA 23187

Abstract

Networks of workstations (NOWs) are cost effective platforms to do parallel computation. Usually, a NOW is not dedicated to parallel jobs. Local users may run some applications in their workstations which involve communications as well. This paper examines the effects of communication interactions of parallel and local jobs on a non-dedicated NOW. Three representative communication patterns of parallel jobs are considered. A quantitative model to characterize the interactions is proposed. Measurement results on a NOW support the analytical model, and indicate that the network interface in the TCP/IP protocol forms a communication bottleneck during interactions because a standard network interface with a single input/output queue is not able to distinguish communication requests from parallel and local jobs. Therefore, small but important communication messages of a parallel job, such as a barrier synchronization, could be easily blocked by a communication request of a local job, which would degrade the performance of the parallel job significantly. A double queue scheme in the network interface is proposed. Using available information from the protocol layer, the scheme is able to distinguish the two types of communication requests and give a higher priority to parallel jobs' communication requests. The simulation results show that the scheme could improve the performance of parallel jobs without significantly affecting the performance of local jobs.

Keywords: Message-passing, network interface, networks of workstations, time-sharing.

[*]This work is supported in part by the National Science Foundation under grants CCR-9102854 and CCR-9400719, by the Air Force Office of Scientific Research under grant AFOSR-95-1-0215, and by the Office of Naval Research under grant ONR-95-1-1239.