A REAL-TIME NETWORK MANAGER FOR DISTRIBUTED IMPRECISE COMPUTATION

Hidenori Kobayashi *,1 Nobuyuki Yamasaki *,2 Yuichiro Anzai *,3

* Faculty of Science and Technology,
Koio University,
3-14-1, Hiyoshi, Kohoku-ku,
Yokohama, Kanagawa, 223-8522, JAPAN

Abstract: This paper describes a real-time network manager which is suitable to client-server models in distributed real-time systems, where the server tasks are based on the imprecise computation model. An imprecise task is a task that is capable of trading off the quality of the result to the amount of the execution time. However, the task lacks the ability to determine the appropriate amount of work. The proposed real-time network manager determines, on behalf of the server task, the adequate amount of execution time as well as the adequate transmission time by holding the intermediate results in an EDF manner. Copyright © 2001 IFAC

Keywords: Real-time communication, Real-time operating systems, Distributed models, Communication environments, Multimedia

1. INTRODUCTION

Significant advances in recent processors and network technologies have made it possible to construct real-time systems over a network. In these distributed real-time systems, tasks communicating over the network are often in the client-server model. This model is very useful in systems where the resources are physically distributed. An example of such systems is a video-on-demand system where the client and the server are located on different nodes. Another example is a robot which consists of several nodes where sensors and actuators are attached to different nodes. In the latter example, the model is usually called the sensor-actuator model.

One of the important characteristics of the distributed real-time systems is that the end-to-end timing constraints often must be met over the network. There are several crucial and distinct problems in guaranteeing the end-to-end timing constraints between a client and a server. They are related to the network, the operating system, and the application software.

Firstly, the greatest source of unpredictability is the network. The network traffic generally varies from time to time, so does the network latency. An exclusive line or a centralized management scheme is generally necessary to cope with this problem.

Secondly, the main issue in operating systems is the scheduling algorithm. Besides selecting an appropriate scheduling algorithm, the protocol processing of messages must be integrated to it in order to guarantee end-to-end timing constraints.

Lastly, two problems are concerned with the execution time of a server task. One of them is that the execution time of the server needed to produce the result of the same quality is not always the same. For example, execution time of a multiple target tracking task varies according to

1 kobahide@ics.keio.ac.jp
2 yamasaki@ics.keio.ac.jp
3 anzai@ics.keio.ac.jp
the input data and the number of the tracks the task was holding at the time. In addition, different clients may request different timing constraints and different quality of results for the same operation. To cope with such situation, a server task must be able to dynamically trade off the quality of the result for its execution time. The other problem is the estimation of the optimal amount of work, which should be completed before the result is actually transmitted back to the client, provided that the server is allowed to terminate at an arbitrary point. Since a task does not have any information on the available execution time nor network latency to the destination node, it is impossible for a server task to determine its appropriate execution length.

One solution to all the above problems is to statically schedule all the communications and the executions of tasks on every node together, which certainly is an unrealistic approach in practice. In contrast, the main purpose of this paper is to provide a real-time guarantee for dynamically established client-server relations. It describes a network manager architecture that, by imprecise computation, solves both the operating system and the application design problem stated above. The network manager also handles unexpected network congestion to some extent by means of imprecise computation and some features of the communication hardware, Responsive Link, described in (Yamasaki, 1997).

2. IMPRECISE COMPUTATION

Imprecise computation was introduced in (Lin et al., 1987) as a technique to trade off the quality of the result for the amount of execution time. A task based on the imprecise computation model consists of two parts, the mandatory and the optional part. The mandatory part merely produces an acceptable result, while the optional part enhances the quality of the previous result. In other words, in real-time systems, the mandatory part must always complete in time, while the optional part need not execute when an overload occurs or when the timing constraint was too strict.

A number of researches on imprecise computation were carried out from various aspects. Some of them regard imprecise computation as a means to achieve graceful degradation in quality of service. For example, (Feng and Liu, 1996) applies the imprecise computation technique to real-time communication services in an ATM network. The scheme uses the cell loss priority bit in the header to distinguish mandatory packets from optional packets. Some of other works focus on controlling the executions of imprecise tasks so that the performance of the whole system would be maximized. This includes vast numbers of scheduling algorithm. Examples are found in (Feng and Liu, 1997), (Kim et al., 1998), and (Barua and Hickey, 1998). Others are related to the implementation of imprecise computations. In (Hull et al., 1996), Imprecise Computation Environment (ICE), an environment for developing imprecise systems is described. The ICE helps the application programmer by automatically generating support code for imprecise tasks.

3. DISTRIBUTED CLIENT-SERVER MODEL

In this section, characteristics of the client-server model, especially in distributed real-time systems, are pointed out. As previously stated in section 1, the required execution time of the server is not constant at all. Accordingly, it is reasonable to assume that the server task is based on the imprecise computation model. Under this supposition, there are two issues concerning the distributed client-server model.

One of them is related to the amount of the execution time given to the server task. Since the server task is based on the imprecise computation model, longer execution of its optional part usually leads to higher quality in the result. However, longer execution possibly results in higher deadline miss ratio of the client task. In order to find out the balancing point of this trade off, both information on the logical structure of the imprecise task and the available execution time is necessary. The reason that the logical structure of the task is needed is that even an imprecise task would not produce any result before the completion of its mandatory part. In addition, the result produced by an actual imprecise task is often a staircase function of allocated time. This means that the optimal point to terminate the task is at where the results of higher quality are produced. As a matter of course, a task itself is capable of determining the optimal point that it should terminate if the length of available execution time is given. However, in a multiprogrammed environment, a task has no knowledge on the amount of execution time available to it. Note that making the task ask the scheduler about its available execution time is not an efficient method, as the available execution time is subject to dynamic task arrivals and deletions. Thus, it introduces too much overhead and no guarantee. On the other hand, the operating system conversely has no knowledge of the logical structure of the task. Therefore, neither the task nor the operating system can set the termination point of the imprecise task appropriately. This problem will be solved shortly in section 4.
4. HARDWARE INDEPENDENT ISSUES

4.1 The Real-Time Network Manager

The real-time network manager lies between the application tasks and the network, whose overall architecture is shown in figure 1. Its aim is to transmit in time the best possible result produced by an imprecise task server without actually figuring out the balancing point of the previously stated trade off in execution time. This is mainly achieved through interactions between the tasks and the real-time network manager.

A typical case of such interaction is illustrated in figure 2. At time $t_{REQ}$, the client task makes a request through the local and the remote real-time network manager. The message contains the server identifier, the request identifier, and the latest time the result should arrive, $t_{DUE}$. The request arrives at $t_{ST}$, and at the same time the server task starts to execute a corresponding service routine. The server task completes its mandatory part at time $t_{MEND}$. At this time, a pointer to the result, the size, and the deadline of the message, $t_{DUE}$, is handed to the network manager with the transmission request. In contrast to the case where a task is responsible for deciding when to transmit the data, the network manager is responsible for it, while the server task now has to take care not to overwrite the previous result contained in the buffer. The network manager holds the requests in a queue in the ascending order of the latest transmission time. For the sake of simplicity, it is assumed here that no other tasks put requests to the network manager. In this case, the network manager immediately sets itself to sleep until $t_{TRANS}$. Meanwhile, the server task continues to execute its optional part and produces a higher quality result at $t_{OEND1}$. This time the server task tries to update the information held within the network manager. The network manager recalculates the latest transmission time and again sets itself to sleep. As shown in the figure 2, the server task would have produced a result of even higher quality if it were allowed to execute till $t_{OEND2}$. Suppose this more precise result can not be delivered to the client node in time. Then, the result is of no use to the client and the server task had just executed in vain. This situation can be avoided by the network manager. It wakes up at $t_{TRANS}$ and transmits the result. And then, it signals the server task to stop its execution.

In summary, the proposed real-time network manager guarantees the best possible result be delivered to the client in time, by reserving necessary transmission time for intermediate results. As a matter of fact, the server task had no idea on the shortage of its execution time until it was signaled to terminate its execution.
Fig. 3. Reserved transmission time

4.2 Transmission Queue Management

On processing the request, the network manager treats two types of data differently. In this Paper, the data that must be transmitted timely is called the mandatory data or the mandatory result. On the other hand, the data that does not have to be transmitted if it can not meet its timing requirements is called the optional data or the optional result. For example, a result of a mandatory part is a typical kind of the mandatory data, where the entire data must be transmitted to the opponent timely in order to make sense. In comparison, a result of an optional part could be categorized as both cases. If the result of the optional part was something that should be added to the previous result of the mandatory part, the added data is an optional result. If the data were to replace the previous result, the new complete data will also be categorized as a mandatory result.

The proposed real-time network manager fully utilizes this flexible nature of results to resolve contentions between transmission requests. First, the network manager calculates the latest time which the transmission of the result should end at by subtracting the network latency and the reception protocol processing overhead from the latest acceptable arrival time of the result. This is labeled as \( t_{\text{END}} \) on the time axis in figure 3. It then calculates the latest time the transmission of the whole data and the mandatory part should begin, \( t_{\text{OST}} \) and \( t_{\text{MST}} \) respectively, by subtracting the transmission protocol processing overheads of the corresponding size from \( t_{\text{END}} \). From these calculated values, the manager checks whether any conflict between requests exists. Passed requests are held in the ascending order of \( t_{\text{END}} \). The only check is made on whether the necessary time to transmit the mandatory result, 1 in figure 3, overlaps that of the neighboring requests. The overlap in transmission time of the whole data, 2 in figure 3, is not checked at all here. One reason for this is that even checking both overlaps can not always provide full guarantee that the whole data is transmitted on time. If the network latency varies unexpectedly and suppose it was larger at the start of the transmission, it may well be appropriate to discard the optional part in order to meet the deadline. In conclusion, checking both overlaps does not pay.

If a new transmission request was rejected, the task has no guarantee in transmitting the result.

Fig. 4. Transmission of a denied request

However, it does not mean that there is no time to transmit the data. That is, the server task still has a chance to transmit the result in time. This is illustrated in figure 4. Boxes on the time axis show previously accepted requests. Suppose a newly arrived request had \( t_{\text{MST}} \) at 4 and \( t_{\text{END}} \) at 6. This request conflicts with one of the accepted requests, hence can not be accepted by the network manager. However, transmission of the data is possible at time zero. This sort of transmission is not always possible, so the server task should stop its execution immediately on the rejection of its first request and try to transmit the result as early as possible. If it was an update request, the server task just terminates. The conflict between this direct transmission and the reserved transmission by the real-time network manager is avoided by scheduling the network manager with the highest priority.

4.3 Imprecise Transmission

The network manager wakes up at \( t_{\text{OST}} \) to transmit the result. At this moment, it recalculates \( t_{\text{OST}} \) and \( t_{\text{MST}} \) using newer values of corresponding network latency. These recalculated values are used to check whether the reserved amount is still adequate or not. If the current time of the system was earlier than \( t_{\text{OST}} \), it transmits the whole result. Otherwise, it transmits only the mandatory data. Since the network manager neither checks the existence of any optional data nor validates its timing requirements, this is an optimistic algorithm.

5. HARDWARE DEPENDENT ISSUES

This section describes issues related to the actual implementation of the proposed scheme. Furthermore, a method for mitigating the effect of unexpected variations in network latency is also described.

5.1 Responsive Processor and Responsive Link

The proposed real-time network manager is implemented in a distributed system, which consists of Responsive Processors connected via maximum of four Responsive Links. Details of Responsive
Processor and Responsive Link are described in (Yamasaki, 1997). One Responsive Link consists of one Event Link and one Data Link, whose packet size is 16 bytes and 64 bytes respectively. The network manager uses the Event Link to transmit requests and the Data Link to transmit results. A packet has two priority bits in its header field. This is used for two purposes in the switch. One is the arbitration and the other is the routing of the packet. The priority may be changed when the packet crosses the switch, depending on the routing table. This paper takes these advantages to rank the messages. The last feature is that it uses DMA to transmit and receive data. Although this allows faster transmission, it introduces great unpredictability to the CPU memory references.

5.2 Calculation of Communication Delay

The end-to-end communication delay between two tasks is calculated as a summation of software protocol processing overhead in the sender node, network latency, the software protocol processing overhead in the receiver node. Note that the delay caused by the local task scheduling is not included. This is because a node in distributed systems never holds the latest information of remote node schedules. In a system that uses Responsive Link, protocol processing overheads are calculated by a function of the number of the packet. These overheads will remain the same regardless of the network congestion. On the other hand, network latency can be calculated by a function of the hop count, provided that no contention occurs. Note that the data size does not affect the network latency. This is because, as soon as the first packet comes in, the network manager starts execution so that the reception of the subsequent packet overlaps the protocol processing of the preceding packets. However, if any contention exists, throwing probing packets may be the only way.

The proposed real-time network manager calculates the latency from the hop count if it was the first transmission to the destination node. The network manager receives a feedback from the destination node if any message did not arrive in time. It then holds that value in a table. This value will be used on next transmission to the node and will be updated on another deadline miss.

5.3 Priority Assignment to Packets

In a system where exclusive lines are not available, all the communication must be controlled by a centralized management scheme. However, in systems where this approach is not feasible, there must be at least some way to discriminate real-time messages from non real-time messages. Since the Responsive Link switch is capable of handling four levels of priority, the real-time network manager assigns four levels of priorities to each message in the following manner. The lowest priority is assigned to non real-time messages, the second lowest to soft real-time messages, the second highest to the optional result, and the highest to the mandatory result.

6. EVALUATION

An experiment was performed to show that the proposed real-time network manager actually is capable of realizing graceful degradation in the quality of the result produced by an imprecise server task.

The experiment system consists of two Responsive Processor nodes connected by a Responsive Link. Throughout the evaluation, the SPARC Lite was set to 100 MHz and the Responsive Link 43 Mbaud. Under this condition, preliminary experiments were made to formalize the communication delay of the Data Links. Based on the result, it was possible to calculate the sender’s overhead(S), network latency(N), and the receiver’s overhead(R) by the following equations, where P is the number of the packet and H is the hop count. All the values are in micro seconds.

\[ S = 1.46 \times P + 17.05 \]  
\[ N = 1.13 \times H + 18.23 \]  
\[ R = 8.88 \times P + 31.30 \]

In addition to these values, an end-to-end deadline guarantee requires that the time between the packet reception and the actual wakeup time of the destination thread be also taken into account. Here, it is assumed that it takes as much as 1 tick long, which is 100 \( \mu s \).

The experiment was performed in the following manner. On one of the nodes runs a client task and the network manager. On the other node runs a server task, the network manager, and some periodic tasks. The server task is based on the imprecise computation model. The worst case execution time of the server task is set to 10 \( ms \), where the mandatory part occupies 2.5 \( ms \). The rest of the server task belongs to the optional part. The optional part consists of five parts, each of which requires 1.5 \( ms \) to enhance the quality of the former result. This represents the staircase function in producing imprecise results. The quality of the result produced by an imprecise task is measured in terms of value. The value is zero at the point where only the mandatory part is completed and will be one if all of the optional part is completed. The periodic tasks on the server...
node are used to estimate the effect of dynamic arrival and deletion of tasks with higher priorities. This is accomplished by scheduling both nodes according to the fixed priority algorithm, where priorities are assigned in the descending order to the network manager, the client task, the periodic tasks, and then the server task.

The result is shown in Figure 5. The horizontal axis represents the CPU workload of the server node and the vertical axis represents the value of the result obtained by the client. The client requests the server in two different ways with respect to the relative deadline. The solid line shows the case where the relative deadline is set to 10 ms and the dotted line 12.5 ms. Since the periodic tasks have higher priorities, the available execution time of the server task will decrease in response to increases in the total amount of their execution time. If the relative deadline is 10 ms, the value obtained by the client never reaches 1, even if there is no workload at all. This is because the requested deadline is too tight. Although the relative deadline just covers the worst execution time of the server task, it will not be sufficient when the communication delay and other software overheads are taken into account. At the point where the workload is 70%, the sum of the CPU utilization of the periodic workload and the mandatory part of the server task approaches 1 if the relative deadline is 10 ms. On the other hand, when the relative deadline is 12.5 ms, it actually sums up to 1 if the workload is 80%. In either cases, the server task merely has enough time to complete the mandatory part. When the workload of periodic tasks gets higher than these points, the server task will not have enough time to complete even the mandatory part. Hence, all the transmission request from the server task will be rejected by the network manager.

As for the transmission requests which were accepted by the network manager, the average time it took for the client task to receive the result were 9.68 ms and 12.2 ms respectively. This result confirms that the result is transmitted on adequate time. As a matter of fact, deadline misses were scarcely observed during the experiment.

7. CONCLUSION

In distributed real-time systems, it is crucially important that the end-to-end timing constraints be met. However, in a dynamic real-time system, it is a difficult problem to determine an appropriate amount of execution time for a server task. The proposed real-time network manager solves this problem by means of imprecise computation. The experiment showed that the proposed scheme is capable of determining the adequate amount of execution time as well as the adequate transmission time.

Future works are planned on the management of the network latency. This includes developments of routing algorithms and the management of virtual circuit that are suited to Responsive Link.

8. REFERENCES


