On the Performance of Real-Time Scheduling in a Parallel/Distributed Environment

+Luis F. Friedrich, +Rafael Cancian, *Rômulo S. Oliveira, +Thadeu B. Corso
+ Departamento de Informática e de Estatística * Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina
CP 476 Florianópolis SC Brasil - CEP 88040-900
{fernando, cancian, thadeu}@inf.ufsc.br, romulo@lcmi.ufsc.br

Keywords: Parallel/Distributed environment, Real-Time Systems, Scheduling Algorithms, Performance Evaluation.

ABSTRACT

The increasing complexity of several industrial applications claims for execution environments supporting parallel and real-time computing. Multicomputer systems have a great potential for high performance and reliability because of their expressive number of processors and communication channels. This paper presents an evaluation of real-time computing on a parallel environment that is based on the association of a multicomputer model and a demand driven communication mechanism. The proposed environment allows a reduction of the communication problems presented by conventional multicomputer environments and, also provides a high flexibility in dealing with the problem of mapping the inter-process communication of a parallel program into the interconnection network of a multicomputer architecture. The main goal of the proposed work is to evaluate the performance of the proposed environment for predictable parallel/distributed real-time computing.

INTRODUCTION

Industrial real-time applications are evolving from simple applications to more sophisticated ones with increasing performance and reliability requirements. Many problems in real-time domain cannot meet their requirements without highly *cooperative* computer systems providing features such as parallel computation, scaleable performance growth, and graceful degradation in presence of faults. Since message-passing multicomputers can provide high performance and reliability, they have emerged as natural candidates for parallel real-time applications [1].

Multicomputer designate a class of machines consisting of a set of computing nodes joined by a set of point-to-point physical communication channels. Each computing node possesses a processor with dedicated memory and several communication ports. The presence of multiple physical channels between nodes may contribute for the robustness of a multicomputer against individual node or physycal channel failures [2].

Multicomputers differ from each other mainly by the nature of their interconnection networks [3]. Communication ports can connect nodes through permanent point-to-point physical channels constituting static interconnection networks. A multicomputer with a static interconnection network can be viewed as a graph in which computing nodes are represented by vertices and physical channels are represented by edges. Alternatively, communication ports can connect nodes through electronic switching circuits, whose manipulation allows for the assignment of temporary point-to-point physical channels, constituting dynamic interconnection networks. Each point-to-point physical channel serves exclusively the two nodes it connects. The duration of a physical channel in a static interconnection network is equal to the time of the existence of the network itself, while in a dynamic network it generally corresponds to the transport time of a fixed-size packet.

A parallel real-time application for multicomputers can be conveniently expressed as a *logical process network* — a set of processes that interacting with each other exclusively by the exchange of messages through dedicated (point-to-point) logical communication channels [4]. A logical process network can be viewed as a graph in which vertices represent processes and edges represent logical communication channels.

The performance of a logical process network in a multicomputer depends on the solution of the problems of *process mapping* (corresponding to the assignment of processes to nodes) and of *logical channel mapping* (corresponding to the assignment of logical channels to physical channels). Together, these problems can be referred to as the mapping of *logical networks* — the logical process networks of these real-time applications — over *physical networks* — the interconnection networks of multicomputers.

The mapping of a logical network, obtained by the assignment of each process to a distinct node and of each logical channels to a distinct physical channel, in which nodes are not shared between processes and physical channels are not shared between logical channels, can be called *perfect match*. When the perfect match is possible, communications between processes always correspond to direct communications between neighboring nodes. For any communicating processes network considered, the perfect match determines the maximum simplification of the communications and the minimum execution time.

When the perfect match is not possible, the logical network mapping in order to achieve efficient execution is a rather complex task [5]. Process mapping in a dynamic interconnection network it is arbitrary, while in a static interconnection network it must consider the processes communication patterns of the logical network. Logical channel mapping, in the dynamic interconnection networks depends on the multiplexing of the physical channels for each packet, while in a static interconnection network it originates the routing of packets between non-neighboring nodes.

Processes are invoked in response to *events* that are generated by other processes. A process repeatedly receives messages from an input channel, performs some actions in response to these messages, and possibly sends messages through one or more output channels.

Generally, real-time systems are characterized by computational activities with stringent timing constraints[6] and a structure that can be classified as static or dynamic. A static structure of real-time tasks means that there is a complete prior knowledge about the task-set attributes (maximum execution time, precedence constraints, etc). A structure is called dynamic when the task-set is dynamic, there is a current task-set which can be varianble in time[7].

As a logical processes network, a real-time application can be represented in terms of its structure and behavior. The structure is either a static logical process network (resulting from static creation of process and logical channels) or a dynamic logical process network (resulting from dynamic creation of processes and/or logical channels). A static logical network may represent a real-time application with a constant resource demand model, in which the amount of resources is fixed for each execution and can be determined in advance. A dynamic logic network represents a real-time application with a variable resource demand model, in which the amount of resources changes for each execution. The behavior is represented by the connection requirements that a process or a collection of processes must have. Consequently, the mapping of logical networks is an important issue because a poor mapping contributes to an increment in traffic congestion, which can affect the performance of real-time communications.

This paper focuses on the evaluation of real-time communication capabilities and real-time scheduling solutions of a distributed/parallel execution environment based on a multicomputer model. The paper is organized as follow. Section 2 describes the proposed multicomputer-based environment and shows an initial qualitative evaluation of it in relation to the widespread multicomputers based on static interconnection network. Section 3 defines a mode of operation for the proposed environment. Section 4 describes the programming model and the scheduling approach chosen in order to evaluate the real-time scheduling capabilities of the proposed environment. Section 5 presents the performance evaluation results. At last, section 6 presents some final considerations.

THE PROPOSED PARALLEL/DISTRIBUTED ENVIROMENT

The general model of the proposed environment is based on a multicomputer architecture whose nodes are simultaneously joined by two dynamic interconnection networks [8], as shown in Figure 1. The multicomputer architecture is composed of:

- An expressive number of *work nodes*, each one possessing a processor with dedicated memory and a set of communication ports.
- A *control node* to manage the connection/disconnection of temporary point-to-point physical channels between work nodes.
- A *work network* (a crossbar) to transport messages between work nodes through physical channels, operated externally by the control node.
- A *control network* (a bus) to transport messages between work nodes and the control node.

The proposed parallel/distributed environment includes, as a fundamental component, a specific two level communication mechanism with a demand-driven logical channel mapping. In this mechanism, the work network is used to transport high-level messages of the logical process networks (real-time applications) between the work nodes in which the processes are executing. The control network is used to transport low-level messages between work nodes and the control node, carrying requests and replies for

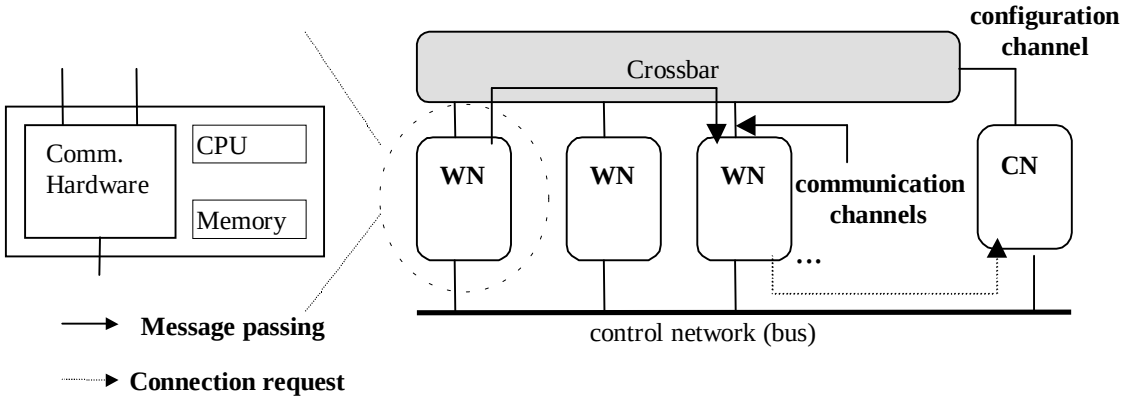

Figure 1- The proposed parallel/distributed environment

connection or disconnection of physical channels of the work network. Therefore, before proceeding to the effective communication between two work nodes, the existence of a point-to-point physical channel between them through the work network is verified. If there is a physical channel, the communication can be immediately initiated; if there is not, it must be preceded by a connection procedure.

The communication latency of this mechanism is determined by two elements: the connection procedure between two work nodes and the transport of a message through the work network. The duration of a physical channel (until its disconnection, also demand-driven) can vary from the time needed for the transport of a single message up to that of the time needed for the complete execution of the real-time application. Therefore, for physical channels that last for some time, whenever the connection procedure can be avoided, the communication the latency is predictable and depending only on the transport of a message through the work network.

A qualitative evaluation of the proposed environment

The proposed environment can initially be submitted to a qualitative evaluation regarding its simplicity and its flexibility, in relation to the widespread multicomputers based on static interconnection networks.

Simplicity

The simplicity of multicomputer-based environments can be measured by the procedures required for exchanging messages through their physical networks.

In environments based on static interconnection networks, communications involve packet routing (with all known problems of route determination, buffer management, packet handling, and control flow [5]). In the proposed environment, the communication mechanism with demand-driven logical channel mapping always provides point-to-point transport of complete messages, eliminating packet routing.

The elimination of the packet routing is responsible for a dramatic reduction in the size and complexity of the algorithms involved.

Flexibility

The flexibility of multicomputer-based environment can be measured by the procedures required for the logical network mapping.

- *Execution of a single static logical network for which it is possible to establish perfect match.*
  In static interconnection networks, the logical network mapping must be planned before loading the program based on knowledge of the topology of the logical network and the physical network. In the proposed network, the assignment of processes to nodes is determined arbitrarily by the duration of the processes and the assignment of logical channels to physical channels is established by the duration of the logical channels. The perfect match is obtained by the construction of the physical network at execution time.
- *Execution of a single static logical network for which it is not possible to establish perfect match.*
  In static interconnection networks, the logical network mapping can be planned before loading the application program when one wants to take advantage of the communications patterns through a convenient adaptation of the logical network to the

physical network. In the proposed network, the procedures described remain the same as in the preceding item this new situation. However, the connection procedure must now be activated whenever the assignment of a logical channel to a physical channel must be altered at execution time.

- *Simultaneous execution of one or several dynamic logical networks.*
  In static interconnection networks, the logical network mapping cannot be planned before loading the application programs because the topologies of the logical networks are not known in advance. In this situation, exploring the communications patterns becomes a complex task. In the proposed environment, the procedures are not affected by this new situation.

The proposed environment can support a large range of real-time applications without changes in its logical network mapping procedures.

MODES OF OPERATION

The proposed multicomputer-based environment provides several modes of operation, each one having a different manner of using and sharing resources such as processors and crossbar, and the possible solutions for real-time scheduling on that architecture depends on the mode of operation [9]. In all cases the main resources to be scheduled are the processors, the communication channels and the control network. Other possible resources in the system are shared data structures and devices. A lot of research has been done in scheduling of tasks sharing data structures, such as cyclic executive [10], priority ceiling [11] and lock-free synchronization [12]. Physical resources, such as devices connected to a processor, can be treated similarly. This paper discusses real-time scheduling, on the target architecture, restricted to the local processor, the crossbar and the control network, because these are the key elements of the architecture. This section describes how the target architecture has been operated to provide an execution environment for parallel applications with real-time constraints.

The system executes a single application, which can be composed by *processes*. Each WN executes a process that can be composed by one or more tasks. Each task is a scheduling unit and can be represented by a *thread*. Threads belonging to a process all share the same memory. Depending on the mode of operation there can be execution support tasks, in addition to application tasks, competing for the processor in a processing node.

Each processor executes one or more threads and they communicate each other through accessing shared data structure. The communication between threads of different processing nodes is done through the crossbar. In this case, for each possible destination, two different approaches can be taken: (1) there is only one local thread which is responsible for sending the messages to a destination node, or (2) many local threads can ask for sending a message to the same destination node.

At the same time, the crossbar configuration is dynamic, every time a thread in one WN needs to send a message to another WN, first it send a message through the control network to the CN. After the physical connection is done the sending thread is warned by the CN and so the communication can take place. Every logic communication channel is mapped directly into a physical communication channel, without routing. At the time a thread finished sending a message it must release the physical connection. The overhead involved in establishing the connections and in treating possible conflicts between different connection requests will affect the execution time of application threads.

PROGRAMMING MODEL AND SCHEDULING APPROACH

In this work we are considering soft real-time applications, in which a deadline miss will not cause catastrophic consequences. However, it is supposed that the quality of the application is proportional to the number of deadlines achieved. The applications are developed following the client/server programming model. This programming model results in graphs with linear precedence. The client is divided in two parts: before calling the server and after calling the server. Therefore we have a linear graph composed by 3 tasks: (1) client before call, (2) server, and (3) client after call. Figure 2 depicted the situation. In addition, a server might be client of another server and so on. In this case we could have a graph composed by 5, 7 or more tasks.

An activity is defined as a set of tasks with precedence relationship, direct or indirect. The execution of an activity is initiated by some event happening in the environment, such as the activation of a sensor or timeout signal. The result of the activity is a response to the environment from the system, depending on the event that was observed. That way, the deadlines are not

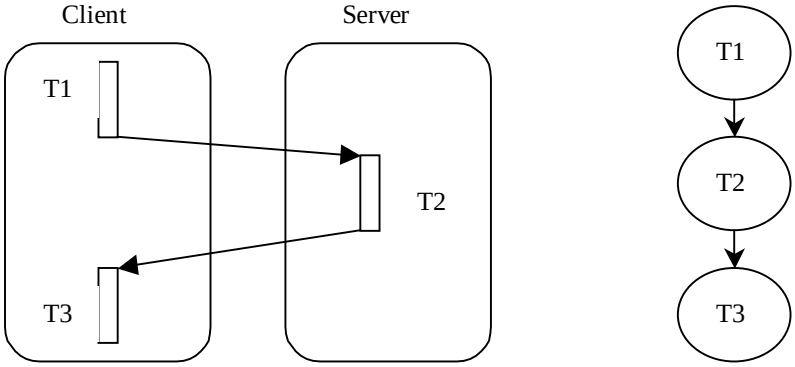

Figure 2 - Client-Server relationship translated to a graph of precedence.

individually associated with tasks; they are associated with all the activity. Therefore, an activity will accomplish its deadline only when all the tasks of the activity will finish up to the deadline.

It is supposed that the activities are aperiodic which means that we must employ a best effort scheduling approach. It is also supposed that the tasks were previously allocated to the processors and there is no task migration during the execution.

The scheduling solution which is under evaluation in this paper applies EDF (Earliest Deadline First) [13] to schedule the tasks in each processor. Each task has an associated deadline that is the same deadline associated to the activity that the task belongs to. This solution is easy to implement and presents a low overhead.

As described in section 3, when a task need to send a message to another task residing in a different processor it must request a crossbar connection to the CN in order to send the message. If the number of connection requests is greater than the crossbar capacity (number of physical channels) then we have a queue of requests that must be kept by the CN. In this case, it is necessary to decide which algorithm will be used by the CN in order to attend the queue of requests. Although a FCFS (First-Come First-Serviced) algorithm seems to be the fairest one and is probably the right choice for a conventional system, the goal of a real-time system is to maximize the number of deadlines achieved.

In this next section we show the performance evaluation, through simulation, of the following 3 different algorithms for scheduling of connection requests:

- FCFS (First Come First Served), the scheduling of the requests is done in order of arrival;
- EDF (Earliest Deadline First), the scheduling is done based on the deadline of the activity associated with the message;
- SMF (Smallest Message First), the scheduling is done based on the message size.

PERFORMANCE EVALUATION

The performance of the proposed communication mechanism for real-time applications was evaluated by simulation in a way similar to what was done in [14] for another task model and scheduling approach. It was supposed in the simulation that the implementation of the communication mechanism, which is based on a demand-driven assignment of physical channels, is performed by the run-time support composed by a central component (executed in the CN) and local components (executed in each WN). The Control Node is responsible for the crossbar operation, executing the connection and disconnection of physical channels.
A set of applications was randomly generated. Three parameters varied during the simulations:

- The algorithm used by the CN, with three possibilities: FCFS, EDF, SMF.
- The number of Work Nodes, with four possibilities: 8, 16, 24, 32.
- The number of physical channels in each WN, with two possibilities: 1, 2.

The model used to simulate the system was created using the Arena 3.01 system simulation software. The model simulates the main behavior of the environment,

composed by the Control Node (CN), the Work Nodes (WN), the Working Network and the Control Network.

Activities are composed by tasks that run on WN processors. After first executing on a client processor, tasks may communicate with another WN by sending a message through the crossbar. To send a message the task must request to the CN a connection with the other WN. To make that request, tasks use the Control Network.

When a message arrives on the destination WN (server), a new task (belonging to the same activity) is created and put in the ready queue of that processor. After getting the processor and finishing the execution, another message is generated and sent back to the client WN following the same procedure. When it arrives at the client WN, other task executes (task T3 in figure 2).

In each experiment the model simulates the execution of 1000 activities (variable *vNumAtividades* in the model). During this time it collects statistics about the system behavior. Each activity has a deadline (attribute *aDeadline*) and all tasks that belong to the activity share this value. The model chooses randomically a Work Node (WN) to start its execution. There is no workload balance.

The number of Work Nodes (WN) varies with the simulations since it is a parameter to be analyzed. It assumes the values 8, 16, 24 or 32 (variable *NumberOfWN*) in the simulations done.

An activity is formed by tasks, that are the real schedulable entities. A task executes in a WN processor for 20 units of time (variable *WNProcTime*). The WN uses a preemptive EDF scheduling algorithm for the processor, with a quantum of 1 (one) unit of time (u.t.) (variable *vQuantum*) for tasks with the same deadline.

After executing in a WN processor, the task may either send a message through the crossbar to another WN or just finish (finishing the whole activity) if there is no more communication to do. The number of messages sent by an activity is 4 (variable *NumberOfMessages*), what means the creation of five tasks (Figure 3). Three of them executing on the client, and two executing on the server(s). There may be more than one task trying to execute on the same WN processor at the same time.

The model separates some WN to create a server pool while the others WN act only as clients. When a client WN sends a message, it chooses randomly one WN from the server pool. When a server WN sends a message it always send it back to the same client WN that caused the task execution on the server.

To send a message, the task must request a connection to the Control Node (CN) through the Control Network. The period for pooling the control bus is 0.1 u.t. (variable *PoolFrequency*) and the execution time of each request on the CN is 0.5 u.t. (variable *CNProcTime*). Three algorithms were used by the CN to schedule the requests: FCFS, EDF and SMF.

The size of the messages varies from 2^8 = 256 bytes (variable *MinMessageSize*) through 2^{12} = 4 Kb (variable *MaxMessageSize*), in powers of 2, following an uniform distribution. After the connection has been established by the CN, the message is transferred through the crossbar. The time to send a 1 Kbyte message through the crossbar is 20 u.t. (variable *CrossBandwidth*).

Each WN has one or two physical channels (variable *MaxChannels*) to send messages. A message may use one or two channels to be transfered. When using two channels, the time needed to decompose/compose the message is insignificant and was not considered in this work. To evaluate the impact of the number of channels for each WN, two groups of charts are shown. The first one shows the behavior of the algorithms with one physical channel, and the second group uses two physical channels for each WN.

The CN allocates just the necessary number of channels to meet the message deadline. Once a connection is requested the CN verifies the availability of the communication channels on both origin and destination Work Nodes and also verifies how many channels (attribute *MaxChannels*) is necessary to send the message in order to meet the deadline. If there are enough physical channels available then the communication is established and the communication time is calculated as a function of message size (attribute *MessageSize*), communication delay (variable *CrossBandwidth*) and number of physical channels used (attribute *MaxChannels*). If the available communication channels are not enough to meet the message deadline, the available number of channels are allocated and the communication is established anyway. If there are no available channels then the request stays in the CN queue for scheduling. The message may miss its deadline but it is never discarded.

The deadline chosen for an activity varies from a minimum value (expression *MinDeadline*) to a maximum value (expression *MaxDeadline*) expressed as follow:

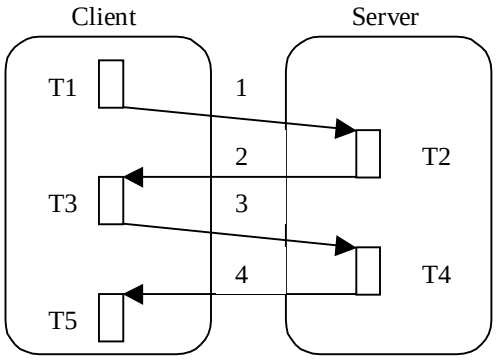

Figure 3 - Number of messages and tasks of an activity in the simulation model.

$MinDeadline = WNProcTime + PoolFrequency + CNProcTime + CrossBandwidth * 2^{MinMessageSize} / 1024$

$MaxDeadline = 2* WNProcTime + NumberOfWN * PoolFrequency + NumberOfWN * MaxChannels * CNProcTime + CrossBandwidth * 2^{MinMessageSize} / 1024$

MinDeadline is the minimum time needed by a task to execute on a WN and to send the biggest message without any delay. It consists of the execution time on the WN (*WNProcTime*), time waiting in the Control Network (*PoolFrequency*), time executing on the CN (*CNProcTime*) and time to send a $2^{MinMessageSize}$ Kbyte message. The value *1024* is used to convert bytes to Kbytes.

MaxDeadline is that time assuming maximum delay. In this case, we have assumed that we have no more than 2 tasks concurring for the WN processor. The deadline (attribute *aDeadline*) of an activity follows a triangular distribution with these parameters:

TRIA(*MinDeadline****,*** *MinDeadline* ***+ (****MaxDeadline* - *MinDeadline****)/3,*** *MaxDeadline****) .***
Where:

Point of Minimal probability = *MinDeadline*

Point of Mean probability = *(MinDeadline + MaxDeadline) / 3*

Point of Maximal probability = *MaxDeadline*

The Figure 4. shows this distribution.

The time between arrivals for activities, the activity period, is

*ArrivalPeriod = MinDeadline * Load*

where *MinDeadline* is the minimun time needed for a task to complete execution without any delay, and *Load* is a correction parameter that, in this work, was kept constant with the value 0.7. The activity period is constant for all simulations. If a task misses its deadline while executing or sending a message, it is not deleted but it is marked as missed and continues normally. Just when all tasks belonging to an activity finish then the activity is classified. If at least one of its tasks missed its deadline, then it is considered that the whole activity missed the deadline. Otherwise the activity met the deadline.

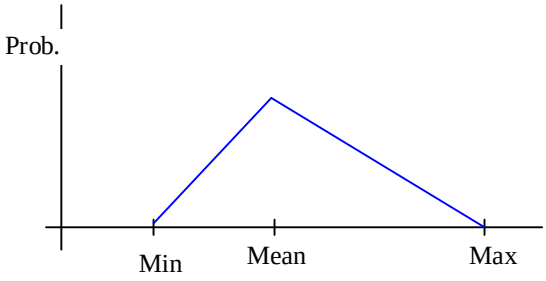

Figure 4 - Triangular distribution

Each one of the three CN scheduling algorithms were simulated 5 times with 8, 16, 24 and 32 Worker Nodes executing 1000 activities, for a total of 60 simulations (and 300000 tasks executed). For each group of 5 simulations it was calculated an average. These average simulations of the 3 algorithms with 8, 16, 24 and 32 WN are showed bellow, in two groups: with one or two physical channels for each WN.

Figure 5 shows how much time (*ut*), in average, the tasks spend in getting connected. Generally, if the communication channel is available the connection time is less than 1*ut*. However, when the solicited channel is not available than the task needs to wait for the channel to be released. The graphics show that the performance of the EDF algorithm is very close to the FCFS.

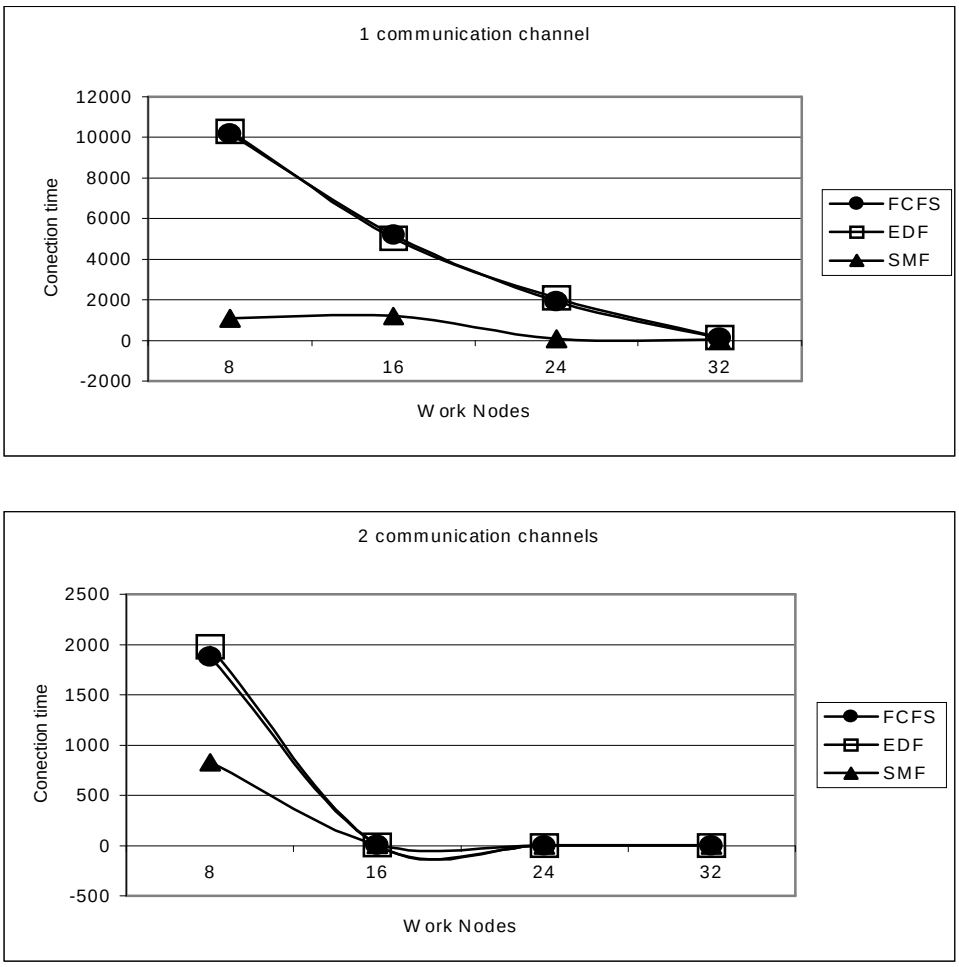

Figure 5 - Average connection time of all tasks

Figure 6 shows the percentage of tasks that missed their deadlines. Deadlines are initially associated to activities which can be divided in more than one tasks. Each task of a specific activities receives as deadline the same deadline of the activity. If a task belonging to an activity miss its deadline the activity also miss the deadline. The graphics show that when using 1 communication channel the performance of the algorithms is poor, except for the SMF. However, with 2 communication channels all three algorithms have a good performance. The use of a crossbar gives a high traffic capability to the interconnection network, similar to N dedicated buses. This capability is multiplied every time we add a new physical channel.

Figure 7 shows the minimum delay of tasks that missed their deadlines. In this case the delay is calculated by the following:

$$Delay = tnow - (aChegadaTarefa + aDeadline)$$

where,

tnow is the actual time, *aChegadaTarefa* is the task arrival time and *aDeadline* is the deadline. The graphics show that the algorithm SMF has the best performance in both cases (1 or 2 channels). However, while increasing the number of work nodes FCFS and EDF tend to have the same performance of the SMF.

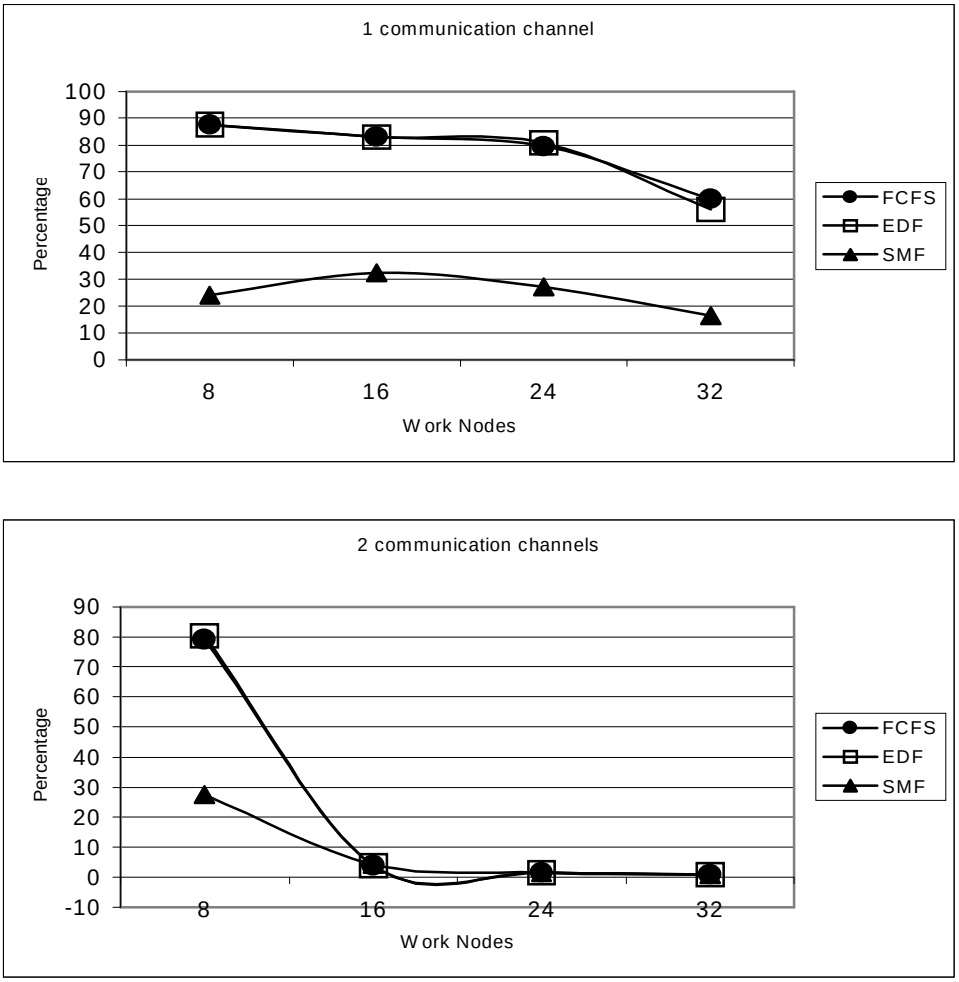

Figure 6 - Percentage of tasks that missed deadlines

CONCLUSIONS

With the high transmission rate provided by the crossbar interconnection network, messages are mostly sent without queue formation, what means that the schedule algorithm has a minor effect on the performance under normal workload. Under overloaded conditions, the choice of a scheduler for the CN may produce significant impact on the target architecture. Therefore, in order to evaluate the impact of the scheduler algorithms used by the CN, some factors in the model were aggravated to cause crossbar saturation, when the schedule effects are best shown.

The results show that algorithm SMF causes less deadline misses and produces a minor delay in almost all cases. When the transmission time is significant, small messages first transmits quickly and those short messages will probably meet deadline (best minimum delay). Large messages that would waste much time transmitting and keeping physical channels busy, are scheduled later, producing the worst maximum delay and also worst average delay with two physical channels.

The algorithms FCFS and EDF produced results not as good as SMF, in terms of deadline misses and minimum delay. Their results showed approximately the same values.

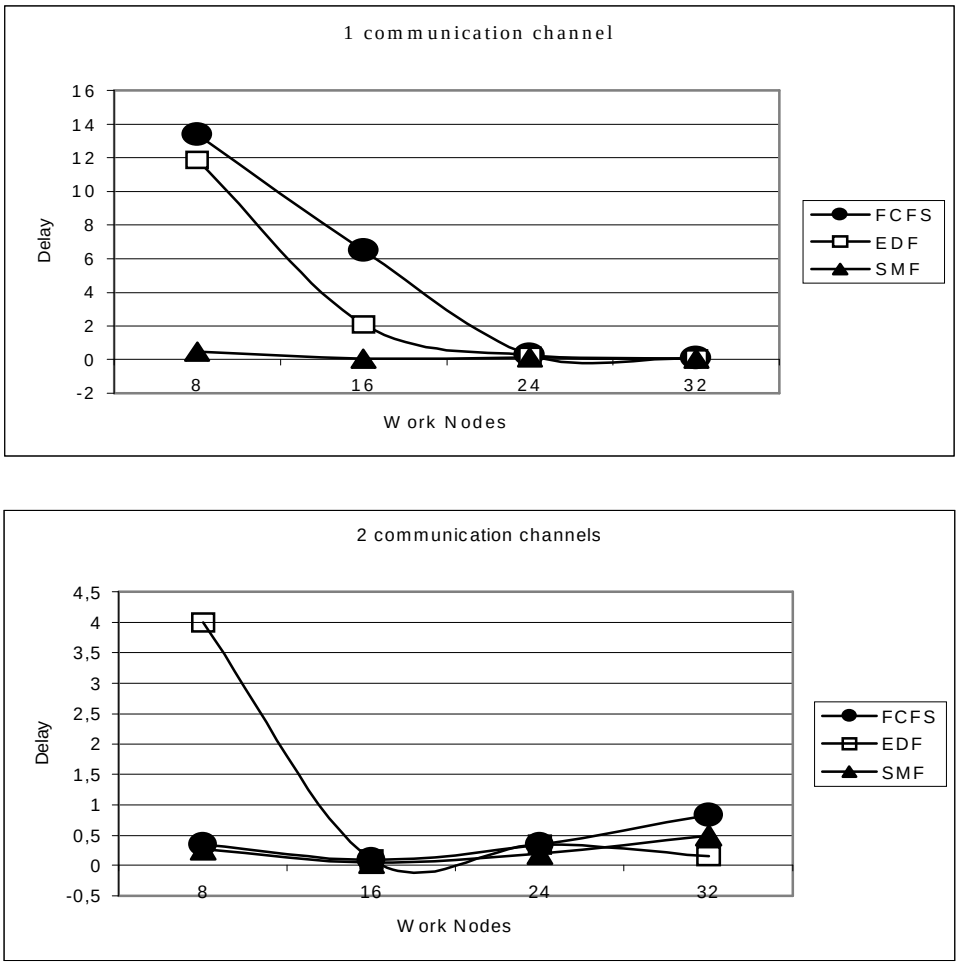

Figure 7 - Minimum delay of the tasks that missed the deadline

Because we are simulating soft real-time applications, the tasks are not discarded when they miss a deadline, but continue in the system until all the activity is executed. This mean that old tasks that had missed deadline or will certainly miss it, are scheduled first, just like algorithm FCFS, with the difference that the ordering is defined by the deadline. In hard real-time applications, where tasks are discarded as soon as they miss a deadline, the scheduler becomes free to choose a task that may still meet deadline, and in that case EDF shows a better result, but that is not the case in this study. According to the results it is worth considering, for the target architecture and the suggested programming model , a real-time scheduling solution in which not only the deadlines are taken into account but also the size of the messages the tasks use to communicate each other.

REFERENCES

1. [Shin et al. 1992] Shin Kang G., Kandlur Dilip D. Ferrari Domenico, “Real-time Communication in Multi-hop Networks”, Technical Report . Real-Time Computing Laboratory University of Michigan, 1992.

2. [Shin et al. 1992a] Shin Kang G., Kandlur Dilip D. Kiskis Daniel L., Dodd Paul S., Rosenberg Harold A. and Indiresan Atri, “A Distributed Real-Time Operating System”, IEEE Software, pp. 58-67, September 1992.

3. [Feng 1981] Feng, T.-Y., "A Survey of Interconnection Networks", IEEE Computer, v. 12, n. 14, p. 12-27, December 1981.

4. [Athas 1988] Athas William C. and Seitz Charles L., "Multicomputers: Message-Passing Concurrent Computers", Computer, August, 1988.

5. [Reed and Fujimoto 1987] Reed D. A. and Fujimoto R. M., *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, 1987.

6. [Buttazzo 1997] Buttazzo Giorgio. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, 1997.

7. [Kopetz 1997] Kopetz Hermann. *Real-Time Systems Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

8. [Corso et al. 1998] Corso T. B., Fraga J. S. , Freitas F. P.. "Demand-Driven Multicomputer environment: Design and Evaluation", SCS Western Multi Conference 1998, San Diego, USA, January 1998.

9. [Friedrich et al. 1998] Friedrich L. F., Oliveira R. S., Corso T. B. "Escalonamento Tempo Real em uma Arquitetura Baseada em Multicomputadores", I Workshop de Sistemas de Tempo Real, Rio de Janeiro, Brazil, Maio de 1998. (in Portuguese)

10. [Xu and Parnas 1993] Xu J., Parnas D. L.. "On Satisfying Timing Constraints in Hard Real Time Systems", IEEE Transactions on Software Engineering, v. 19, n. 1, p. 70-84, January 1993.

11. [Sha et al. 1990] Sha L., Rajkumar R., Lehoczky J. P. "Priority Inheritance Protocols: An Approach to RealTime Synchronization", IEEE Transactions on Computers, v. 39, n. 9, p. 1175-1185, september 1990.

12. [Anderson and Ramamurthy 1996] Anderson J. H., Ramamurthy S. "A Framework for Implementing Objects and Scheduling Tasks in Lock-Free Real Time Systems", Proc. of the l7th IEEE Real-Time Systems Symposium, december 1996.

13. [Liu and Layland 1973] Liu C. L., Layland J. W.. "Scheduling algorithms for multiprogramming in a hard real time environment", Journal of the Association for Computing Machinery, v. 20, n. l, January 1973.

14. [Friedrich et al. 1998a] Friedrich, Luis F. et al. "On the Performance of Real Time Communication in Multicomputer Interconnection Networks", 1998 Summer Computer Simulation Conference (SPECTS'98), Reno, Nevada, USA, July, 1998.