Mathematical & Computational Applications, Vol. 8, No. 3, pp. 353-360, 2003
© Association for Scientific Research

A DYNAMIC LOAD BALANCING MODEL FOR A DISTRIBUTED SYSTEM

Oğuz AKAY[1] and Kayhan ERCİYEŞ[2]
[1]Ege University International Computer Institute, Izmir, Turkey
[2]Visiting Professor, University of California Davis, Computer Science Dept., U.S.A.
akay@bornova.ege.edu.tr, erciyes@cs.ucdavis.edu

Abstract- A communication protocol designed for fault tolerance in distributed real-time systems is implemented and a dynamic load balancing model is designed and implemented over this protocol. The protocol consists of cluster based, hierarchical rings which use synchronous communication. The rings are synchronous. At the lowest level in the hierarchy, there are clusters that consist of computing processors, called nodes. The higher level consists of the cluster representatives that manage the clusters of the lower level. There can be two or more levels in the hierarchy. Ring protocols in each cluster can work in parallel. Also, a fault tolerance mechanism is integrated to the protocol. The dynamic distributed load balancing module designed over the protocol aims to maximize the overall performance of the whole system by distributing the load submitted to the system efficiently and transparently among the nodes. While performing operations to achieve this goal, the module also considers the real-time constraints of the system.

Keywords- Distributed, real-time, fault tolerance, load balancing

1. INTRODUCTION

One of the main components of a distributed system is the distributed process scheduler that manages the resources of the system. The efficient usage of the large computing capacity of a distributed system depends on the success of its resource management system. A distributed process scheduler manages the resources of the whole system efficiently by distributing the load among the processors to maximize the overall system performance [1]. The distributed scheduler must perform the load distributing operations transparently, which means the whole system is viewed as a single computer by the users.

A distributed system consists of independent workstations connected usually by a local area network. Users of the system submit jobs to the system at random times. In such a system, some computers are heavily loaded while others have available processing capacity. The goal of the load distributing scheme is to transfer the load at heavily loaded machines to idle computers, hence balance the load at the computers and increase the overall system performance [2, 3].

The aim of this project is to design and implement a distributed process scheduling mechanism for the previously designed hierarchical cluster based distributed real-time system model [4-6]. The system in this model consists of clusters of computing nodes. A synchronous ring protocol is running in each cluster. Clusters are managed by their cluster representatives. Also there is a higher level ring that consists of the cluster representatives of the lower level. There may be two or more levels in the hierarchy. At the highest level, there is a leader that manages the whole system. The system is designed to provide a distributed real-time platform for distributed real-time applications [7].

The paper is organized as follows: In section 2, the implemented distributed system model and communication protocol is described. In section 3, the algorithmic description of the distributed process scheduling module designed over the protocol is explained. In section 4, the implementation of the protocol and the distributed scheduler is explained and the results of simulations are given. Section 5 contains the concluding remarks.

2. THE HIERARCHICAL RING PROTOCOL

The system consists of clusters of computing processors called nodes. The nodes in a cluster are connected to each other by a local area network. In each cluster an independent synchronous ring based communication protocol is running. Each ring is managed by a cluster representative which is a member of the higher ring. The highest level ring in the hierarchy is managed by the leader. The leader controls the operations of the whole system. The two level system architecture is seen at Fig. 2.1.

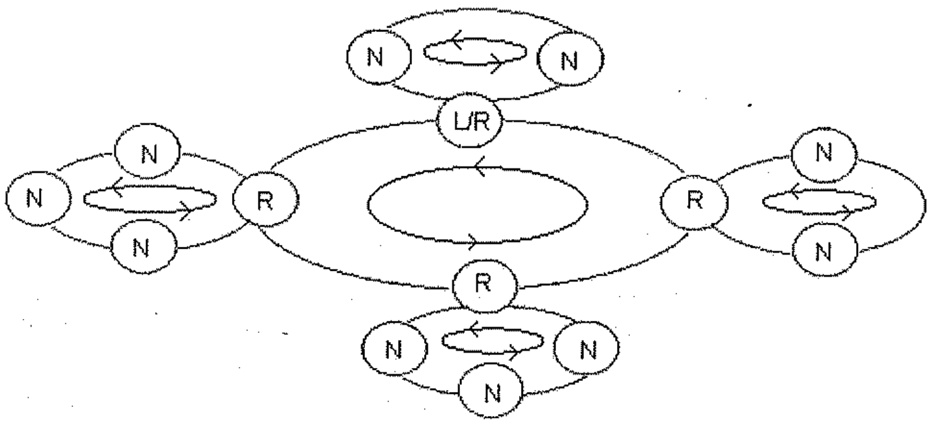

Figure 2.1. The two level system architecture

The protocol is operated by the three modules: Node, Representative and Leader. Operation of the protocol on a two level hierarchical system is as follows: The leader at specific periods releases a token called "OUTFRAME" in the outer ring to collect some information from the cluster representatives. The outframe consists of two parts: The first part contains the global information that is read only by the representatives and carries the decision of the leader for some operation. The second part is writable by the representatives and contains the information about the clusters. A cluster representative that receives the outframe first reads the global section and updates its state variables. Then it puts its local information collected from its cluster into the appropriate slot in the second part of the outframe and sends the outframe to the next representative in the outer ring. Also, upon receiving the outframe the cluster representative releases a token called "INFRAME" in its inner ring to collect the information from the nodes of its cluster. Like an outframe, an inframe consists of two parts, read only first part contains the leader's decision about the system operation and second part carries the local information of the nodes. A node upon receiving this inframe reads the global section and updates its state variables and then puts its local information into the appropriate slot in the inframe and sends it to the next node in the inner ring. By this protocol the leader collects all information from the nodes of the system, and dictates its decisions about the operation of the system back to the nodes.

Also a fault tolerance mechanism is integrated to the protocol. With this mechanism a single node, representative, leader and a single whole cluster crashes are detected and recovered by the system. When a node crash occurs, the crashed node is detected by its neighbours in the corresponding ring and reported to the representative. The representative then removes the dead node and repairs the ring. Similarly when a representative crash occurs, it is detected and reported to the leader by the neighbour representatives and then the leader removes that representative from the outer ring. At the same time, the representative crash is detected by the last node of the inner ring and after repairing the ring, that node starts the new representative. If the leader crashes, the last representative of the outer ring repairs the ring and starts the new leader.

3. THE DISTRIBUTED PROCESS SHEDULER

Over the protocol described in section 2, a distributed process scheduling mechanism is designed. In this model, nodes are processors of the system and tasks are submitted to this nodes. The leader periodically collects load levels of the nodes via cluster representatives by releasing the outframe in specific periods. The representatives collect load values from the nodes in their clusters by releasing inframes. The leader and representatives classify nodes according to collected load values in three groups: high-level nodes, medium-level nodes and low-level nodes. Representatives make this classification for the nodes in their clusters, while the leader classifies all of the nodes in the system. The CPU queue length is used as the load value of a node.

When a node receives a task start request, it first checks its load value. If the load value is below the high load level, it starts the task and updates its load value. If the current load at the node is above the high level, it sends a task transfer request message to its representative. Upon receiving this request, the representative searches its information previously collected from its cluster to find a suitable node for task transfer. If it can find a low or medium level node, it forwards the found node's address to the node that makes the request. Then the nodes perform the task transfer between each other. If the representative cannot find a suitable node in its cluster, it forwards the request to the leader. Then the leader searches to find a suitable node in another cluster. If it can find a node for transfer, sends its address to the node that makes the request. Then again the two nodes perform the task transfer operation. When a transferred task completes, the result is returned to the original node that requested the task transfer.

For real-time properties of the system, also a timing mechanism is used at nodes. If a task transfer requested by a node is not completed in a certain interval, the process is started by the original node. To simplify the task transfers the shadow process method is used. In this method each node keeps copies of process images. When a task transfer occurs only the name of the process is sent to the destination node.

The distributed scheduling module consists of Node, Representative, and Leader processes. Now we describe them in more detail.

3.1. Node process

When the node process receives "INFRAME" it puts its current load value into the appropriate slot and sends it to the next node in the cluster. When the node gets "PROCESS_START" message, it first checks its current load value, if it is smaller than the high load level, the process is started on that node and the load value of the node is increased. If the load at the node is greater than the high load level, node sends a "LOAD_TRANSFER_REQUEST" message to its representative and starts a "LOAD_TRANSFER" timer. If node receives a "LOAD_TRANSFER_DESTINATION" message as a reply, it sends a "LOAD_TRANSFER_REQUEST" message to the node whose address is contained in the message. If it receives a "LOAD_TRANSFER_COMPLETE" message from the other node it stops the "LOAD_TRANSFER" timer. This means the task transfer completed successfully. If the node receives a "LOAD_TRANSFER_REJECT" message for its transfer request, it stops the timer, starts the process itself and increases its load value.

When the node process receives a "LOAD_TRANSFER_REQUEST" message, it first checks its load value. If it is smaller than the high load level then it starts the process whose name is specified in the message, then it increases its load value and sends a "LOAD_TRANSFER_DESTINATION" message to the node that made the request as reply. If its load value is greater than the high load level then it replies the task transfer request by "LOAD_TRANSFER_REJECT".

When a running process is completed on the node, first the node decreases its load value. If the process is local (if it is submitted to that node), the result is output. If the process is remote (if it is transferred by another node), the result is sent to the original node with a "TERMINATE_PROCESS" message.

If the node receives a "TERMINATE_PROCESS" message, it realizes that a transferred process is completed and it outputs the result contained in the message.

3.2. Representative process

When the representative receives "INFRAME" that it has released previously in the inner ring, it reads the load values of the nodes of its cluster and generates "low_table", "medium_table" and "high_table" according to the load values. When the representative receives "OUTFRAME" it puts the previously collected load values from its cluster into the appropriate slot and send the outframe to the next representative of the outer ring.

If the representative receives a "LOAD_TRANSFER_REQUEST" message, it first checks the low_table. If the low_table is nonempty, it selects the first node in that table. If the low_table is empty, then the representative checks the medium-table, and if it is nonempty, selects the first node in that table and sends the address of the selected node to the node that makes the request with a "LOAD_TRANSFER_DESTINATION" message. Then the representative removes the selected node from its table, increases its load value and puts it into the appropriate table according to its new load value.

If both low_table and medium_table are empty, the representative sends the "LOAD_TRANSFER_REQUEST" message to the leader.

3.3. Leader process

Leader periodically releases an "OUTFRAME" in the outer ring. Upon receiving the "OUTFRAME" it reads the collected load values of all of the nodes in the system and generates "low_table", "medium_table" and "high_table" according to the load values.

If the leader receives a "LOAD_TRANSFER_REQUEST" message, it first checks the low_table. If the low_table is nonempty, it selects the first node in that table. If the low_table is empty, then the leader checks the medium-table, and if it is nonempty, selects the first node in that table and sends the address of the selected node to the node that makes the task transfer request with a "LOAD_TRANSFER_DESTINATION" message. Then the leader removes the selected node from its table, increases its load value and puts it into the appropriate table according to its new load value.

If both low_table and medium_table are empty, the leader sends a "LOAD_TRANSFER_REJECT" message to the node that requested the task transfer.

4. IMPLEMENTATION

The hierarchical ring protocol and the distributed process scheduling module is implemented on a network of UNIX workstations. A single multithreaded process is run on each of the workstations. The node, representative and leader modules are implemented as threads in this process. In simulations fixed 3,7KB frames and 200ms frame generation period is used.

The simulation of two level ring protocol is performed on a single machine by running multiple node and representative processes. The threads within a process communicate with each other by using FIFO queues on a shared memory that are synchronized by semaphores. Table 4.1. shows the inframe circulation times. The values show that a linear increase in the frame circulation times as the number of nodes in a cluster is increased, and as the number of clusters in the system is increased.

Table 4.1. Inframe circulation times in a two level ring (ms)

		Number of nodes in a cluster			
		4	**8**	**16**	**24**
Number of clusters	**2**	3	5	11	16
	4	4	11	21	34
	8	5	19	43	68
	16	7	21		

In Table 4.2. the outframe circulation times are shown. The measured values show that there's a linear increase in the frame circulation times as the number of clusters in the system is increased.

The distributed scheduling module is simulated on two level and three level ring protocols. Simulations are done on 16 Solaris workstations in the Ege University campus network. For the network communication UDP (User Datagram Protocol) is implemented by using Berkeley sockets [8]. In these simulations various numbers of copies of a process is submitted to the nodes. The process chosen is a CPU bound process that sorts a random 5000 integers.

Table 4.2. Outframe circulation times in a two level ring (ms)

		Number of nodes in a cluster			
		4	8	16	24
Number of clusters	2	2	2	2	2
	4	4	4	4	4
	8	12	14	14	15
	16	29	34		

Table 4.3. shows the mean response times of the two level system on some load levels for various configurations. For a comparison the first column shows the response time on a single independent machine that does not part of the distributed system. In Fig. 4.1. is a graph of the same results. The results show that, through low, medium and high load levels there's a slight increase in the mean response time, but above the high level response times increase more rapidly. The reason is above the high load level all of the workstations are heavily loaded so the distributed scheduling module cannot find suitable nodes for task transfer and task transfer requests are rejected. Also the results show that at the medium and high load levels the mean response times at various system configurations are close to each other.

Table 4.3. Mean response times in a two level system (s)

		Number of clusters:Number of nodes							
		1:1	1:4	1:8	2:4	2:8	4:8	4:16	8:16
System	25%(L)	4,7	8,8	9,2	9,7	9,3	10,5	10,1	10,7
	50%(M)	8,8	9,6	10,6	11,2	10,5	11,6	10,7	11,9
	75%(H)	11,9	11,7	12,5	11,8	11,9	12,4	11,8	12,2
	100%(H)	15,5	16,3	17,3	16,7	16,8	17,4	17,6	18,1

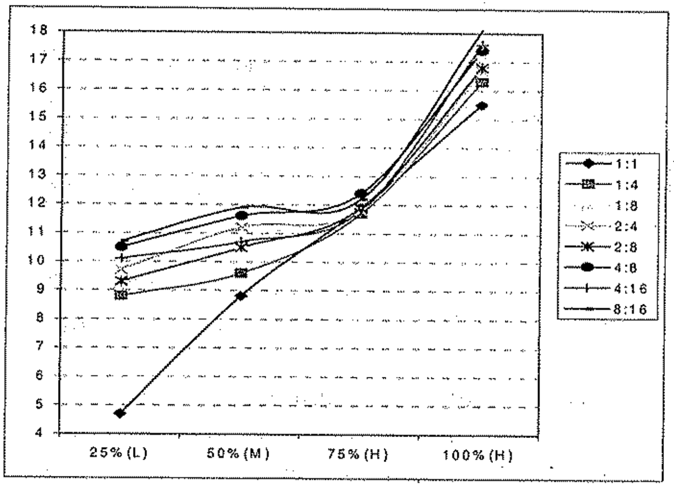

Figure 4.1. Mean response times in a two level system (s)

Table 4.4. means response times of a three level system at various system configurations are shown. Fig. 4.2. is the graph representation o these results. As in the two level simulation, at medium and high system loads response times are close to each other for various configurations, and above the high load level there's a sharp increase on response times as a result of unsuccessful task transfer requests.

Table 4.4. Mean response times in a three level system(s)

		Number of clusters(up lyr.):Number of clusters(mid lyr.):Number of nodes					
		1:1:1	**2:4:8**	**2:4:16**	**2:8:16**	**3:6:12**	**4:8:16**
System load	**25%(L)**	4,7	9,8	9,8	10,3	10,3	10,1
	50%(M)	8,8	10,6	10,7	10,9	10,6	11,1
	75%(H)	11,9	11,2	11,5	11,3	11,1	11,7
	100%(H)	15,5	16,3	17,4	17,1	17,3	17,4

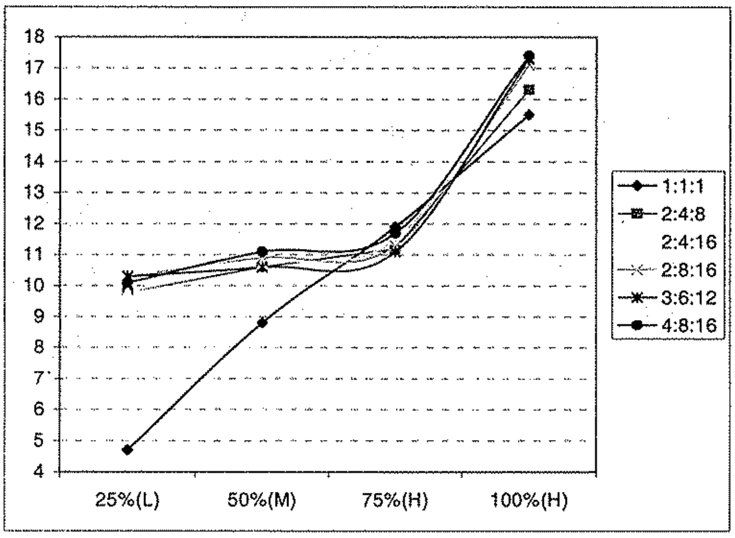

Figure 4.2. Mean response times in a two level system(s)

4. CONCLUSION

In this project, a dynamic load balancing module is developed and implemented over a cluster based hierarchical ring protocol designed for distributed real-time applications. The load information about the nodes are collected periodically by cluster representatives and the leader. The load distributing decisions are made based on these collected information. When a node requests a task transfer, first intracluster task transfers are performed, and when the cluster is overloaded, task transfers are made to the other clusters. This cluster based approach provides scalability. The ring protocol is suitable for information collection and does not increase the message traffic. Also task transfer process brings little overhead to the network and processing resources of the system. The information classification mechanism provides faster decisions for finding suitable nodes for task transfers. The system considers only newly submitted tasks for transfer so no preemptive task transfers occurred which is a much more complicated operation. Shadow process method also prevents the network overhead of transferring task images from one node to another. The simulations show that the distributed scheduling mechanism improves the system performance while bringing a little overhead.

REFERENCES

[1] M. Singhal, N. Shivaratri, *Advanced Concepts In Operating Systems*, McGraw Hill, 1994

[2] N. Aktas, K. Erciyes, Dynamic Load Balancing in a Parallel Processing System, Software for Multiprocessors and Supercomputers, Theory, Practice, Experience, *SMS TPE'94*, Moscow, Russia, September 21-23, 1994

[3] K. Erciyes, O. Ozkasap, N. Aktas, A Load Balancing Model for a Massively Parallel Processing System, *The International Symposium on Computer and Information Sciences*, Antalya, Turkey, October 1994

[4] T. Tunali, K. Erciyes, Z. Soysert, A Ring Protocol For A Cluster Based Distributed System, *BAS'98, The Third Symposium on Computer Networks*, Izmir, 1998

[5] K. Erciyes, Implementation of A Scalable Ring Protocol for Fault Tolerance in Distributed Real-Time Systems, *Computer Networks Symposium 2001 - BAS 2001*, June 20-21, Turkish Republic of Northern Cyprus, 2001

[6] T. Tunali, K. Erciyes, Z. Soysert, A Ring Protocol For A Cluster Based Distributed System, *BAS'98, The Third Symposium on Computer Networks, May 98*, Izmir, Turkey, 1998

[7] O. Ozkasap, T. Tunali, K. Erciyes, A Fault Tolerant Distributed Clock Synchronization Method For A Real-Time Group Communication Model, *ISCIS XII*, Antalya, Turkey, October 1997

[8] R. Stevens, *Unix Network Programming*, Prentice Hall, 1990