Asian Journal of Information Technology 5 (1) : 69-74, 2006

Task Scheduling in Distributed Systems Using a Mathematical Approach

Shatha K. Jawad
Department of Computer Engineering, Al-Balqa' University, Jordan

Abstract: The designed distributed control system consists of autonomous CPUs that work together to make the complete system look like a single computer. So, the microkernel design using two levels of scheduling, the first one for communication and the other for task execution. Level two is proposed by using three routines, Guarantee, Bidder, and Decision Routines. The proposed mathematical approach can be used to implement the latter algorithm.

Key words: Operating systems, distributed systems, system dynamics, Real-time simulation

INTRODUCTION

Distributed systems consist of autonomous CPUs that work together to make the complete system look like a single computer[1,2]. They have a number of potential selling points, including good price/ performance ratio, well ability to match distributed applications, potentially high reliability and incremental growth as the workload grows[3-9]. Distributed Operating System is one class of software for multiple CPUs that turns the entire collection of the hardware and software into a single integrated system. Given a collection of CPUs, some algorithms are needed for assigning CPUs. Such algorithms can be centralized or distributed, local or global and sender-initiated or receiver-initiated[5,6].

Distributed operating systems have to be designed carefully, since there are many pitfalls for the unwary[1]. A key issue is transparency-hiding all the distribution from the users and even from the application programs. In this respect, microkernels are superior to monolithic kernels. The main objective of this work is to find new methods in some services of the microkernel (exactly process allocation and scheduling) to obtain better performance.

Real-time scheduling: Real-time systems are frequently programmed as a collection of short tasks, each with a well-defined function and a well-bounded execution time[7,8]. The response to a given stimulus may require multiple tasks to be run, generally with constraints on their execution order. In addition, a decision has to be made about which tasks to be run on each processor[9,10].

The microkernel, proposed in this work, consists of three algorithms which cooperate together to implement the CPU time scheduling at each PC in the system, these are: Guarantee Algorithm, Bidder Algorithm and Decision-maker algorithm (Fig. 1).The next sections will described these algorithms.

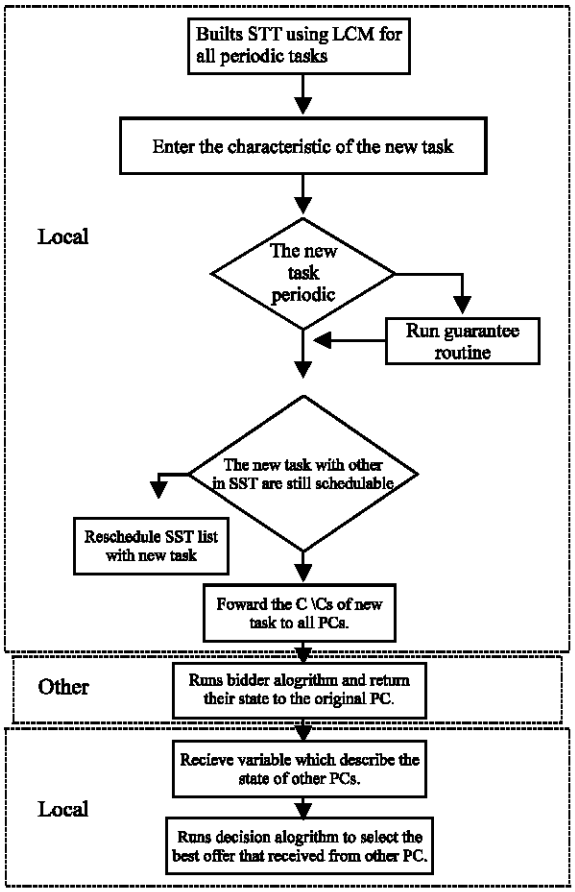

Fig. 1: Forwarding of arrived new task alogrithm

Guarantee routine: Check the possibility of acceptance of the new task by a local PC or must be migrate to another PC.To provide this algorithm , each PC maintains a System Task Table (STT) for all local periodic or aperiodic tasks guaranteed at any point in time.

In other words, the table which contains the time slot of any task and any additional information about any one of them (e.g. arrival time, execution time, etc.) known as STT. The STT can be built by using a dynamic priority assignment to implementing the EDF (Earlier Deadline First) rule. This STT is constructed along the least common multiple of the periods of all periodic tasks. Tasks in the STT are arranged in the order of their arriving times and, within each arrival time by their deadlines.

Each element in the STT list is a data structure called TASK SLOT; that memorizes the processor sharing among the tasks. Each TASK SLOT is described as a Slice Time (End Time – Start Time) assigned to a task characterized by Arrival Time, Execution Time and Deadline. The time between two tasks represents Available Time (AT), i.e. the time in which the processor is not busy. In other words, the STT list is constructed by simulating the processor behavior in advance. New Task is a data structure that contains the characteristic of the new aperiodic task and characterized by Arrival time, Execution time, and Deadline of this new task.

The two-guarantee routine methods are similar in idea but different in the procedure of calculating the Available Time. As described before, the Available Time is used to execute the new aperiodic task.

Slotted task guarantee routine algorithm: It is assumed that the tasks are preemptive, so the Execution Time of the new task can be divided into slots. The number of slots and their size depends on the AT between any adjacent tasks with deadline less than the deadline of new task in STT and the Execution time of the new task. The steps of this algorithm are described as follows:

- Search Start Task in SST list, i.e. the time of TASK SLOT that end before the arriving of the new task arrival and in the same time built a new list that will contain all tasks in SST list and the new task if it is accepte
- Search for Available Time (Inter Task Time) in the SST list, the intervals between tasks, i.e. time when the processor is idle. The search is completed when the sum of these intervals is equal to the execution time of the new task or when the deadline (New Task. deadline) is missed. The latter condition is tested by computing End of New Task as a sum of Inter Tasks Time and Tasks with deadlines less than that of the New Task. The tasks with deadlines greater than that of the new task are saved in another list of delayed tasks, let us call it Delayed Tasks List, which are arranged in order according to their deadlines.
- Check if tasks that are delayed by the new task are still schedulable.

Non-slotted task guarantee routine algorithm: It is supposed that the tasks are non-preemptive, so the new task execution time can not be divided into slots. The following steps describe how this guarantee routine works:

- As described in first step of the previous method.
- Searching the STT list for the first task with deadline greater than that of new task. At the same time accumulate the available time and the execution time of all tasks between Start Task and the first task with deadline greater than that of new task. The search is stopped at any time the accumulator time is more than the deadline of the new task
- Inserts the new task in SST list before the first task with deadline greater than that of new task and add the execution time of the new task to the accumulated time calculated in step 2. If the new adding times go over the deadline then stop and conclude that the new task can not be guaranteed by this PC.
- Checks if tasks that are delayed by the new task (tasks, which have deadlines greater than that of new task) are still schedulable.

The dispatcher is the module that gives control of the CPU to the task selected from STT. Although the dispatcher should be as fast as possible, its time that is required to stop one task and starts another must be taken in to consideration. So, it is required to include the dispatcher's execution time within every task computation time (execution time). As a result, the conclusion is to use slotted task guarantee routine method, with increasing the execution time of the new task by a factor N, where N is define as follows:

$$N = \textit{time needed for each dispatcher} \times \textit{slot's number of new task}$$

For worst case, the number of slots is equal to the execution time of new task. The time needed by dispatcher is an extra time added to the execution time of the new task that to be solved using the 2nd method (non-slotted task guarantee routine).

The new task is either periodic or aperiodic and must be examined for schedulability soon after it arrives. To facilitate this, both the bidder and the local scheduler tasks are executed as periodic tasks. The period and computation times of these tasks are determined a priori by the nature of tasks.

The above scheme is based on the assumption that there is a communication module, executed on a processor separate from the CPU on which tasks are scheduled. This is responsible for receiving information from local workcells' devices as well as from other PCs. Based on the

type of communication, this module stores the received information in an appropriate data structure so that they will be looked at when different tasks are executed.

BIDDER ALGORITHM

When the new task, which is received by a local PC, can not be scheduled locally its characteristic is broadcasted to all PC's.Then each PC (except the local one) runs a bidder algorithm. This algorithm is used to determine incomplete information about the state of each PC running it. Each PC has two bidder algorithms one for periodic and other for aperiodic task.

Bidder algorithm for aperiodic tasks: A bidder algorithm starts to run when it is inspired by a Request For Bid signal (RFB). The local PC broadcasts this signal. The algorithm should return (to the origin PC) the degree to which the PC can guarantee the task.

Some authors[11,12] implemented schemes that evaluate the available time interval between the arrival of a RFB and the task deadline. They took into account all delays encountered during the process of bidding and the percentage of periodic tasks. Others[12,13] implemented schemes that periodically broadcast the PC state, measured as accumulated computational time or total number of tasks on that PC. These schemes consider only aperiodic tasks. The proposed idea for these schemes is that it is better to make an estimation of the state quicker than to make measurement with overhead[9,14]. In our approach, searching the STT within the interval between the arrival time and the deadline for new aperiodic task only and look for parameters that can influence the schedulability of the new task. The analysis is done in cooperation with the guarantee routine and EDF rule. Among the parameters that can influence the schedulability of a task the following assumption are considered:

Available Time (AT) : Is the time between the arrival time and the deadline of the new task when the processor is idle.

TD: Accumulated execution times of already guaranteed Tasks that have to be Delayed as a result of accepting the new task.

ND: The Number of already guaranteed tasks that will be Delayed by the new task.

Note: TD and ND assumed to be as Scheduling Cost.

The bidder algorithm performs only an approximation of the above parameters and sends them to the initiator of RFB (local PC). If any PC in the system can not accept the new task no message will be returned. Because the proposal has two methods of guarantee routine, then two types of bidder algorithms are required. The difference between the two algorithms is: with slotted task guarantee routine method the related Bidder algorithm of any PC returns a message that describes the uncompleted states if the summation of the idle slots time are enough to execute the new task within its deadline. On the other hand and when using non-slotted guarantee routine method, the related bidder algorithm does not care about the pervious described case.

Bidder algorithm for periodic tasks: This algorithm is similar to the bidder algorithm of aperiodic tasks except the parameters that can influence the schedulability of the task which are, number of periodic tasks and number of aperiodic tasks.

To ensure that no PC accept any new task (periodic or aperiodic) until the local PC takes its decision, is to make each PC accept any new task after a fixed interval of time if it is not receive any message from the local PC.

The local PC waits for a fixed interval of time (estimated interval) for bids to come from all PC's. The local PC receives the bids and makes a decision to select the best bid among all bids within the waited interval time and neglects those bids that are arrived after that time.

DECISION MAKING ALGORITHM

The decision-making algorithm runs on the initiator of RFB (local PC) and uses information supplied by bidder algorithm from some PCs found in the system. The questions that should be answered by the decision-maker are: 1) Having received the bidding parameters from each PC, 2) Which is the best PC to send the task to? It is not easy to give a complete answer, but could find partial ones. For example, for aperiodic task, a large available time that verifies the relation available time > execution time of the new task can guarantee the new task. However, it does not contain any information about the delayed tasks. In the same way, a small scheduling cost offers good chances to guarantee the delayed tasks. To handle this qualitative information about the parameters delivered by the PCs and their capacities to guarantee the new task, this next section will be based on Fuzzy Sets theory,

Neuro Fuzzy algorithm to introduce a mathematical approach that can be used to choose the best offer.

THE MATHEMATICAL APPROACH

While trying to use a neural net in implementing the decision algorithm an idea to put a threshold for each layer in their neurons, as those found in the transmission scheduling, was raised. Then another idea was introduced, if the conditions for thresholds are found then why not use it directly without neural net? So the implementation of these ideas was started and the beginning point was to find the conditions to make the correct selection, in the way a mathematical approach was introduced to make the required decision without needing to fuzzy, neurofuzzy, genetic or neural.

For the mathematical approach for aperiodic new task:

- Find maximum Available Time (AT) from all offers as follows:

$$\text{Max. AT} = \max (AT_0, \ldots, AT_{n-1})$$

Where:
n= numbers of PCs which offer bids.

- Find maximum Scheduling Cost (SC) from all offers as follows:

$$\text{Max. SC} = \max (SC_0, \ldots, SC_{n-1})$$

- Find a new value for each SC, let it be called Complement Offer (CO), as follows:

$$CO_i = \text{Max. SC} - SC_i$$

Where: i= 0, 1,…,n-1

- Find maximum Complement Offer from all offers as follows:

$$\text{Max. CO} = \max (CO_0, \ldots, CO_{n-1})$$

- Find the best anticipation offer (B):

$$B = \text{Max. AT} + \text{Max. CO}$$

- Find the competing number (CN) for each offered PC as follows:

$$CN_i = AT_i + CO_i$$

- The selected PC, is a node with CN_i closest to B.

For the mathematical approach for periodic new task:

- Find minimum periodic number (PT) from all offers:

$$\text{Min. PT} = \text{Min} (PT_0, \ldots, PT_{n-1})$$

- Find minimum aperiodic number (AP) from all offers:

$$\text{Min. APT} = \text{Min} (APT_0, \ldots, APT_{n-1})$$

- Find the Competing Number (CN) for each offered PC:

$$CN_i = PT_i + APT_i$$

- Find the best anticipation offer (B):

$$B = \text{Min. PT} + \text{Min. APT}$$

- The selected PC, is a node with CN_i closest to B.

COMPARISON WITH OTHER METHODS

There are some points, which can be deduced from the results that was obtained by running the same simulated systems using fuzzy, neuro-fuzzy[5,6] and mathematical approach to make the decision algorithm, should be taken in consideration. The mathematical approach has the following specification:

- It needs little and simple calculation to find the better offer received from PCs.
- It is faster in finding its selection than the fuzzy and neurofuzzy approaches.
- It needs no temporary stored data that is necessary in calculation to find the best offer as in the case of neurofuzzy systems.
- In the mathematical approach, there is a high possibility to have more than one best offers (equals to CN). The choice here refers to system designer. If he thinks that there is no difference between the number of periodic and aperiodic tasks then any of the above systems can be selected. Otherwise, and when he prefer a system with lowest number of PTs or lowest number of APTs then a condition can be add to select the required characteristic in a very easy way and after detecting the first equal best offer.
- The experience of the designer and the flexibility to choose some parameter can be added to the fuzzy and neurofuzzy facilities when needed to implement the decision algorithm, while the mathematical approach is a mathematical solution only and the final decision are not affected by the conditions above.

PERFORMANCE EVALUATION

After running a twenty different case of systems for each methods used to implement the decision algorithm (Fuzzy with different number of rules, Functional Neurofuzzy, Structural Neurofuzzy and the Mathematical

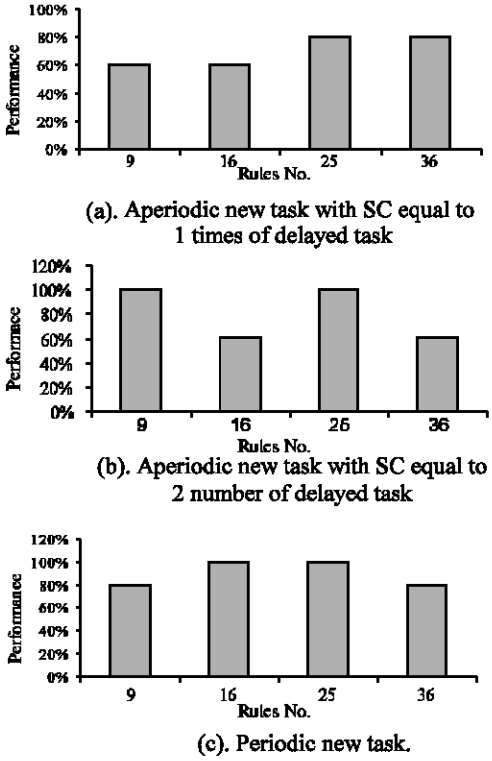

Fig. 2: Performance of fuzzy decision algorithm with different number of rules

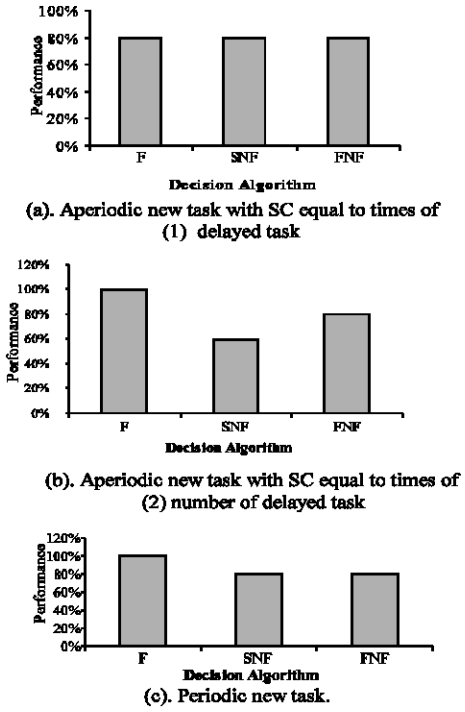

Fig. 3: Decision algorithm perfomance with diifferent approaches

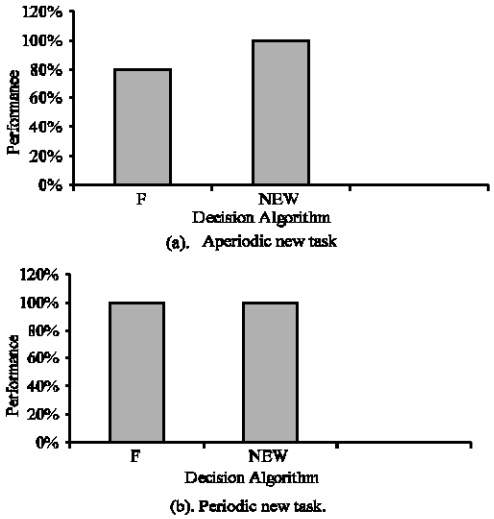

Fig. 4: Performance of mathematical approach and fuzzy decision alogrithm

approach), the final results is obtained as shown in Fig. 2, 3, and 4.

The results represented by performance of the decision algorithm are related to the percentage of the better choice accuracy.

REFERNCES

1. Anderew, S.T., 1995. Distributed operating systems. Prentice Hall.
2. Abraham, S. and B.G. Peter, 1998. Operating systems concept. Addison-Wesley Publishing Company.
3. James, E.G. and T.R. Phillip, 2000. Local area networks. John Weley and Sons, Inc.
4. Jean- Dominique, 1993. A survey on industrial communication network. ANN. Telecommunication, 48: 9-10.
5. Jawad, S.K., 2001. A proposed microkernel algorithm for fieldbus distributed operating system. Ph.D. Dissertation, Control and Computer Engineering Department, University of Technology, Iraq.
6. Jawad, S.K., A. Al-Thamer, S.M. Al-Karaawy and M.A.J. Al-Baker, 2001. Dynamic tasks scheduling in fieldbus Systems. J. Eng. Technol., pp: 20-28.
7. Krithivasan, R., A.S. Jhon and Wei Zhao, 1989. Distributed scheduling of tasks with deadlines and resource requirements. IEEE Transactions on computers, pp: 38-8.
8. George, F.C. and D. Jean, 1988. Distributed systems concept and design. Addison-Wesley published company.

9. John, A.S. ,1989. Decentralized decision making for task reallocation in a hard real-time system. IEEE Transaction on computers, pp: 38-43.
10. Marin, L., C.I. Traian and L. Jesus, 1998. Dynamic task scheduling in distributed real-time systems using fuzzy rules. Microprocessors and Microsystems, 21: 299-311.
11. John, A.S., R. Krithivasan and C. Shengchang, 1985 Evaluation of flexible task scheduling algorithm for distributed hard real-time systems, IEEE Transactions on computer, pp: 34-12.
12. Krithivasan, R. and A.S. Jhon, 1984. Dynamic task scheduling in hard real-time distributed systems. IEEE Software, pp: 65-75.
13. Kang, G. S. and Y.I. Chieh Chang. 1989. Load sharing in distributed real-time systems with state-change broadcasts. IEEE Transactions on computers.
14. Cherkassk, V. and F. Mulier, 1998. Learning from data. Wiley- Inter science Population.