Analysis and Simulation of a Distributed Real-Time System

Chris Baumler
CPRE 558

Project Summary

Real-time distributed systems are an increasingly popular solution to the problem of increased system performance demands. They offer advantages such as load migration and redundancy. A major concern when designing real-time distributed systems is scheduling tasks in such a manner that deadlines will be met. Methods to achieve this goal generally involve the use of a global scheduler that has a transfer policy, an information policy, a selection policy, and a location policy. One such global scheduling algorithm is the Buddy Strategy. This project simulates this algorithm with some restrictions to decrease scope. The simulation models are created using Simulink, a block based modeling software. Experiments are designed and performed, and results are reported. The conclusion of the project is that the Buddy Strategy is a flawed, but valid solution to the problem of finding a viable global scheduler for distributed real-time systems.

1. Introduction

In recent years, real-time systems have become more complex, and their performance requirements have grown more burdensome. As a result, new methods are being employed to provide flexibility, reliability, and performance enhancements. In particular, real-time distributed systems are becoming increasingly popular. These systems are composed of networks of nodes that can share computational tasks amongst themselves. This allows heavily loaded nodes to shift work to lightly loaded nodes, providing better overall performance in terms of meeting deadlines. Such systems vary in structure with some utilizing local area networks and others using wide area networks. Additionally, the specifications of the nodes that make up the system may be identical or may vary widely from node to node. The makeup of a particular system is generally application specific [1].

Distributed real-time systems, in their various forms, offer several advantages that address the needs of modern applications. They are particularly well suited for real-time applications that are already distributed in nature such as military command and control, air traffic control, avionics, and online stock trading systems [2]. These and other applications benefit not only from improved performance through task sharing, but also from the inherent redundancy of distributed systems. Since each node is connected to multiple other nodes in the network, failure of a single node or link will not necessarily result in system failure. Also, the nodes' interconnectivity allows higher system throughput when transmitting simultaneously on multiple links [1].

To be useful in a real-time environment, a distributed system must overcome the primary concern of all real-time systems: the system must be able to schedule tasks in a manner that guarantees the tasks' deadlines will be met. Accomplishing this means correctly scheduling tasks for local processing on each node as they dynamically arrive and also scheduling the messages that pass tasks and state information between nodes. This two part problem is addressed by using a three layered architecture that includes a local scheduler, a message scheduler, and a global scheduler [1].

The first component, the local scheduler, is responsible for scheduling tasks for processing that are arriving dynamically at a node. The tasks that arrive may be periodic or aperiodic. Periodic tasks are commonly handled by preemptive scheduling algorithms such as rate-monotonic scheduling (RMS) and earliest deadline first scheduling (EDF). Both algorithms assign the highest priorities to tasks with the shortest periods. They differ in that RMS assigns priorities statically, while EDF assigns priorities dynamically. Tasks that arrive aperiodically may be scheduled using a periodic server such as a polling or deferrable server, or they may be scheduled using a dynamic algorithm such as the myopic scheduling algorithm [1].

In addition to running scheduling algorithms, the local scheduler may provide services such as resource reclaiming and fault tolerance. Resource reclaiming involves reallocating unused resources that result when tasks running on multiprocessor systems take less time to execute than predicted. Care must be taken to ensure that the reclaiming process does not result in run-time anomalies where tasks that were deemed schedulable before the reclamation miss their deadlines due to the adjustments made. Fault

tolerance involves the use of redundancy or other mechanisms to guarantee timely service in the presence of faults [1].

The second component, the message scheduler, is tasked with establishing a real-time channel between the local node and other nodes in the distributed network. This channel must be able to handle both periodic and aperiodic messages. Periodic traffic requires deterministic guarantees for parameters such as packet length and end-to-end delay. Aperiodic traffic relies on statistical guarantees, since bit rates vary [1].

The last component, the global scheduler, handles load balancing between nodes using four policies. The information policy is in charge of the transfer of information between nodes. It determines what type of data is collected and how often it is collected. The transfer policy keeps track of a nodes loading and decides whether the local node should attempt to send tasks to other nodes or accept tasks from other nodes. The selection policy chooses tasks for transfer from a heavily loaded node. Lastly, the location policy finds lightly loaded nodes to receive tasks from the heavily loaded local node. These policies are a central part of any distributed real-time system and are the main focus of the analysis and simulation performed for this project [1].

2. Objectives and Scope

The problem selected for consideration in this project is the design of an effective real-time distributed system. Effectiveness is gauged through observation and is quantified through the use of performance metrics such as the guarantee ratio. A task is guaranteed when the scheduler can be sure that the task will meet its deadline. The guarantee ratio is the total number of guaranteed tasks divided by the number of tasks that have been given to the node. Another metric is the miss ratio. This is the ratio of tasks that are not scheduled locally to the total number of tasks.

The proposed solution under consideration uses the Buddy Strategy algorithm for global scheduling. This algorithm has been chosen for its straightforward design and common use. The details of the algorithm are discussed in section 3. The scope of this project is limited to analysis of parameters pertaining to the Buddy Strategy. To simplify analysis of the proposed solution, a multiple access network is assumed as opposed to a multi-hop network. This affects the Buddy Strategy by causing the buddy set of each node to include every other node on the network rather than just a subset of the nodes on the network. Choosing an ideal subset is a complex problem worthy of a separate project. Additionally, local scheduling is performed using a simple scheme and only to the extent necessary to carry out the simulations. Lastly, the effects of message scheduling are ignored.

The primary objective of this project is to observe the proposed solution in a simulation environment and evaluate its performance in different scenarios.

3. Solution Approach

3.1 Algorithm

The proposed solution to design an effective real-time distributed system is to use a network of nodes that operate using the Buddy Strategy for global scheduling. The Buddy Strategy uniquely implements the four policies that constitute a global scheduler. Each policy is discussed in detail.

3.1.1. Transfer Policy

The transfer policy uses three states to determine what role a node plays in the network. These states are determined by comparing the node's current load against a series of thresholds. The load is defined as the number of tasks queued for execution on the node's processor. If the load exceeds the upper threshold T_v , the transfer policy judges the node to be in an overloaded state. Likewise, if the load is less than or equal to the lower threshold T_u , the node is said to be underloaded. The node is fully loaded if the length of the queue is greater than a third threshold T_f, but less than or equal to the upper threshold [1]. For this project T_f is assumed to equal T_u.

3.1.2. Information Policy

The Buddy Strategy uses a state driven information policy. When the transfer policy determines that the state has changed, the information policy announces the new state to a subset of the nodes on the distributed network known as the buddy set. Determining an ideal buddy set requires the consideration of some factors. First, size of the buddy set must be considered. If too large of a group is chosen, the network overhead of passing so many state exchange messages is significant. If too small a group is chosen, nodes are less likely to find eligible candidates for task transfers. Next, thresholds must be considered. The thresholds determine the frequency with which tasks are transferred. If T_v is lowered, tasks are transferred more often. Lastly, thrashing must be considered. Thrashing occurs when several overloaded nodes simultaneously send tasks to an underloaded node, causing the underloaded node to become overloaded and to transfer tasks elsewhere. Thrashing may be reduced by assigning priorities to the list of nodes in a buddy set [1].

3.1.3. Selection Policy

The selection policy used by the Buddy Strategy is straight forward. As tasks arrive at a node, they are checked for schedulability. If they cannot be guaranteed, they are considered candidates for transfer to another node [1].

3.1.4. Location Policy

The location policy receives information about the states of the nodes in the buddy set from the information policy. It selects underloaded nodes for task transfers. To avoid thrashing, nodes may be prioritized [1].

3.2 Illustrative Example

As an illustration of the Buddy Strategy, the following scenario is considered. A simple network consists of three nodes as depicted in figure 1. For each node, $T_v = 3$ and $T_u = 1$.

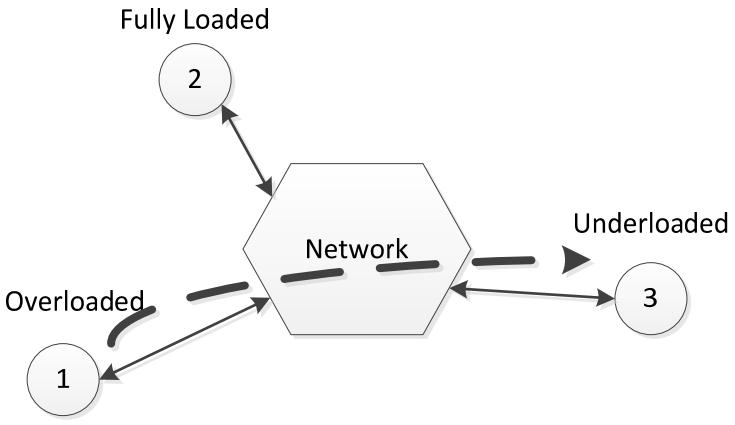

Figure 1 – Distributed Network

Time 0:

Node 3 accepts a locally generated task with computation time and deadline (ci, di) of (4, 12). Nodes 1 and 2 each accept locally generated tasks of (3, 9), (4, 12), and (5, 15). At this time, the transfer policies of nodes 1 and 2 transition them to the fully loaded state, since each has 2 tasks queued to execute when the current task completes. Likewise, the information policies have sent the updated states to the other nodes. Node 3 is still in the underloaded state. The node states, queue lengths, and earliest start times (EST) at this point in time are shown in table 1.

Time 1:

Node 1 receives another locally generated task of (4, 15). This additional task causes the transfer policy to move the node into the overloaded state, since there are now 3 tasks in the queue. Likewise, since $EST + c_i > D_i$, the new task cannot be guaranteed, and the selection policy chooses this task for migration. The location policy examines the state data provided by the information policy and chooses to transfer the task to node 3, because it is underloaded.

Time 2:

Having transferred a task to node 3, node 1 is no longer overloaded.

Time	Node 1			Node 2			Node 3		
	Queue	EST	State	Queue	EST	State	Queue	EST	State
0	2	12	Full	2	12	Full	0	4	Under
1	3	16	Over	2	12	Full	0	4	Under
2	2	12	Full	2	12	Full	1	8	Under

Table 1

4. Simulation

4.1 Simulation Model

Simulations are conducted using Simulink, a block diagram environment for simulation and design. In addition to the basic functional blocks provided by Simulink, the SimEvents block library is used. This library provides a construct called an entity which approximates a message or task. Entities can be assigned attributes and passed from block to block. The library also provides blocks that act as servers, queues, and routers. These blocks are used throughout the simulation environment. The specific construction of the individual model components is described in detail in each subsection.

Figure 2 shows the highest level of the simulation environment with a typical configuration containing four nodes . Each node has a pair of ports for sending and receiving tasks. Additionally, each node has a pair of ports for sending and receiving node state information. The transmit ports connect to a Tx MAC block which routes messages (entities) into the Rx MAC block. The Rx MAC block contains a switch block which routes messages into the correct node/port combination based on the attributes of the message.

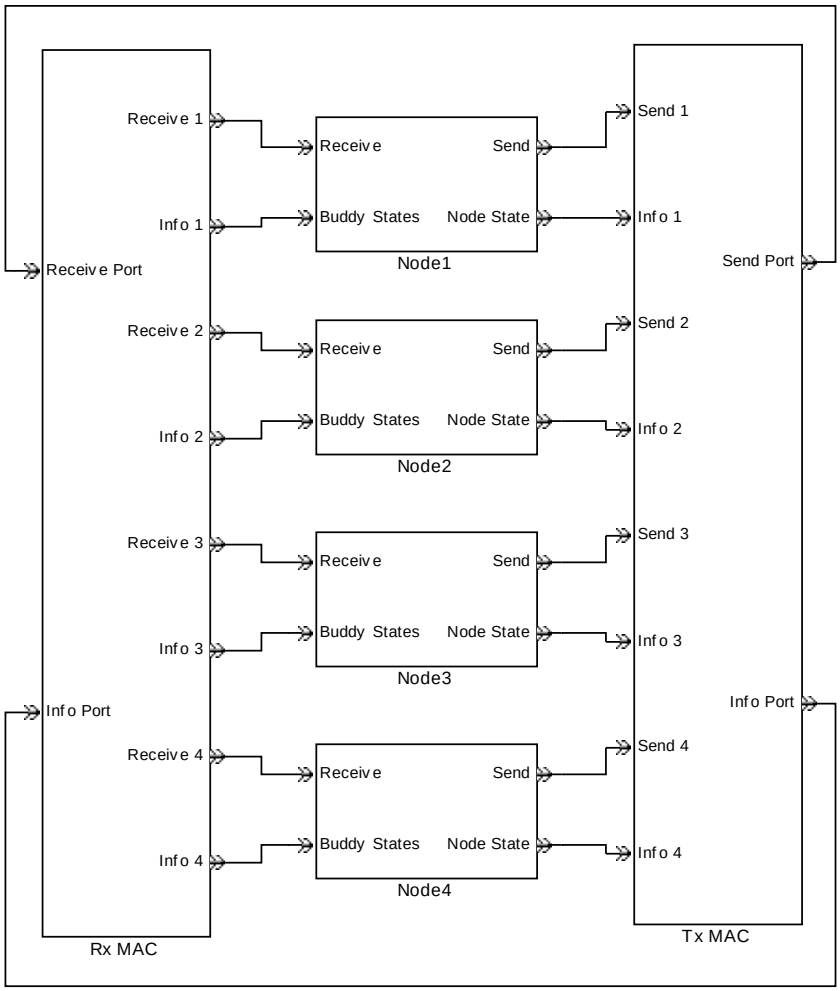

Figure 2

Figure 3 shows the structure that exists within the Node block. The aperiodic task generator block generates local tasks as specified in the setup parameters for the simulation. These local tasks are combined with incoming tasks from other nodes via the path combiner block. Meanwhile, the transfer policy block receives the number of tasks in the queue from a stored memory location and computes the current state. The state is passed the information policy block which sends it out to the other nodes when a transition occurs. The information policy block also receives information messages from other nodes. The selection policy block receives the state and the incoming tasks and determines a destination for each task. The port number of the destination is provided to the output switch block which routes the task to one of three locations. The first possible destination is the local processor. In this branch, the tasks computation time is added to the total computation time on the processor, and the task is added to a first-in-first-out (FIFO) queue. When the currently executing task is complete, the next task is pulled from the queue. The second branch contains the location policy block, this block determines a suitable buddy node to receive a task when this node is overloaded. The third branch contains a server block and a sink. These blocks simply allow the task to be discarded.

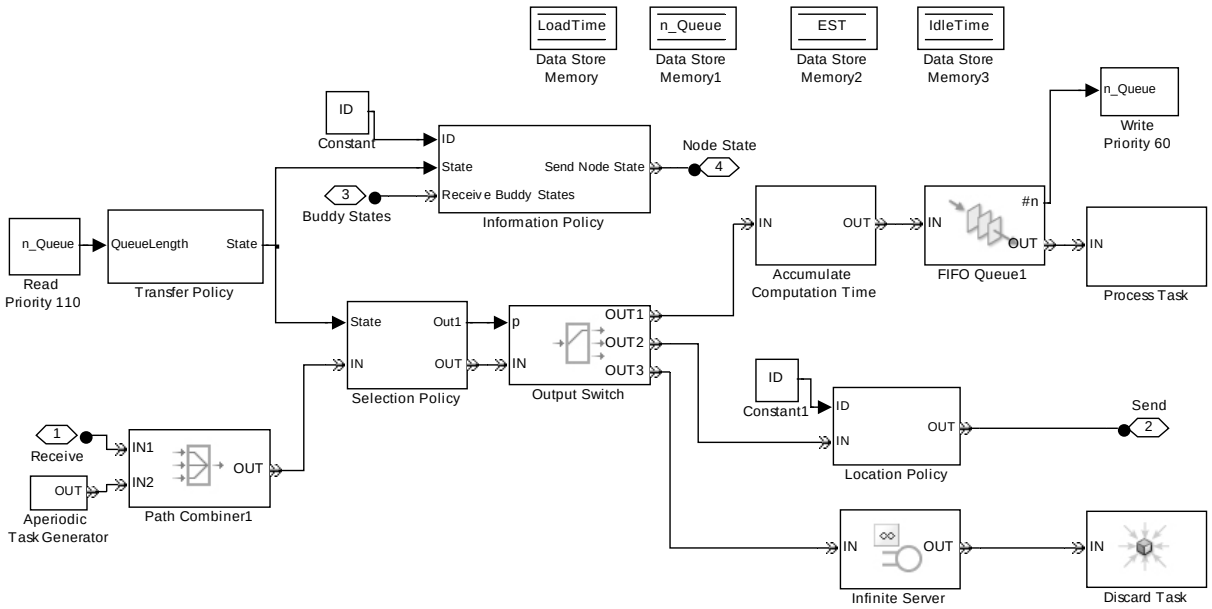

Figure 3

4.1.1 Transfer Policy Block

The internals of the transfer policy block are shown in figure 4. This block compares the queue length to the thresholds T_u and T_v and executes the appropriate if action subsystem to output the nodes current state.

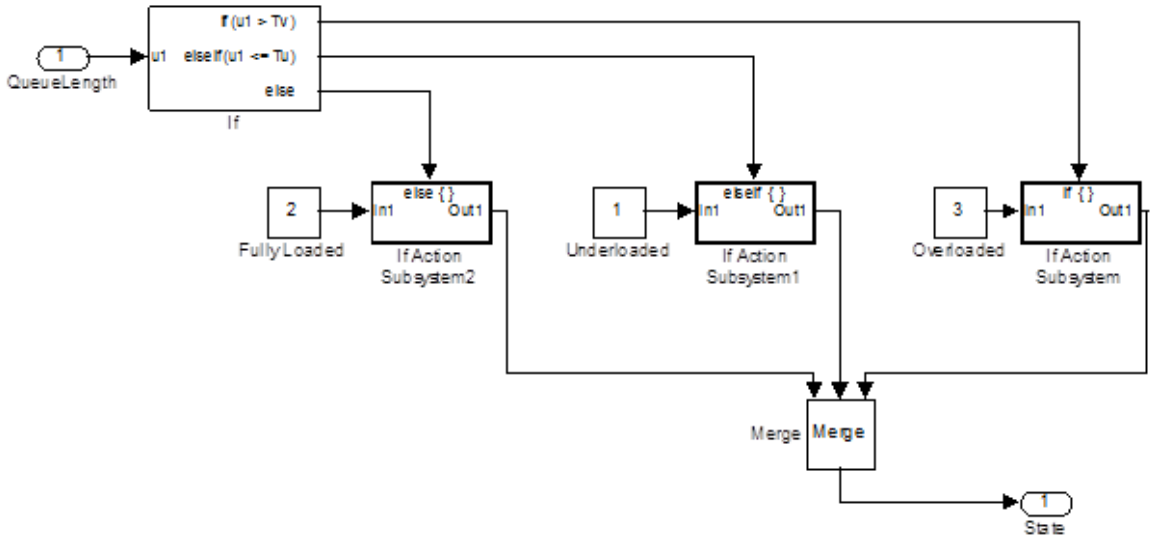

Figure 4

4.1.2 Information Policy Block

The internals of the information policy block are shown in figure 5. The information policy block performs two functions. First, it receives messages from other nodes with state updates. These enter through the "Receive Buddy States" port. The IDs of the nodes sending the messages and the new states are read from the messages and recorded in a table of node data. Then the messages are discarded. Second, when this node's state changes, a new message is generated, and the message is transmitted.

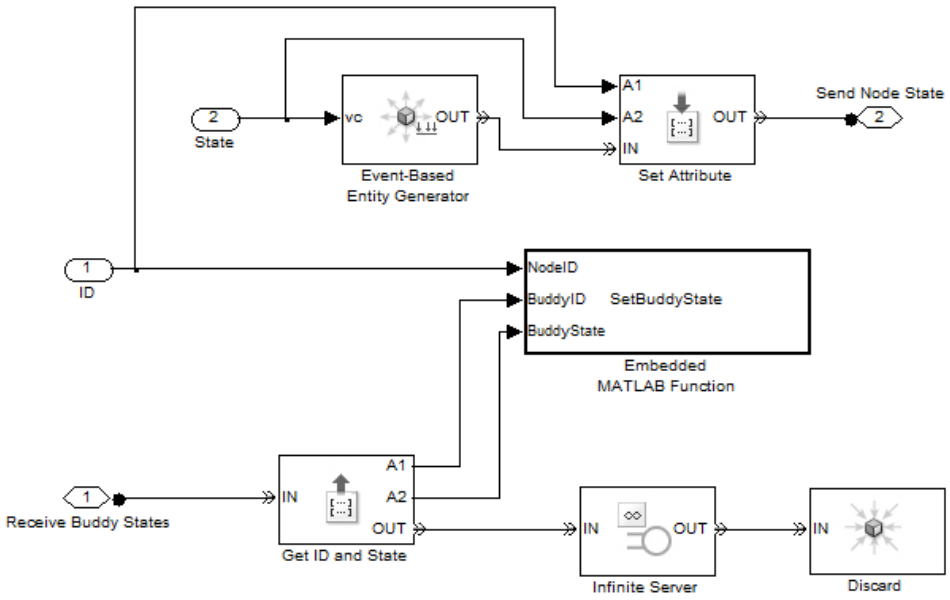

Figure 5

4.1.3 Selection Policy Block

The internals of the selection policy block are shown in figure 6. This block receives tasks through the IN port and computes a destination port value. The “Select Destination Port” function block is called at a sampled rate determined by the time-based entity generator block. When the function is called, it reads in the state of the node, the total processor load time (sum of all tasks that have executed or are queued for execution), the total processor idle time, the new task’s computation time, and the new task’s deadline. The load time combined with the idle time represents the earliest start time for the new task. If the earliest start time plus the computation time of the task is less than the task’s deadline, the task is routed to the local processor. If the task is not schedulable on the local processor, the state of the node is checked. If the node is in the overloaded state, the task is routed to the location policy. Lastly, if none of the above conditions is met, the task is discarded.

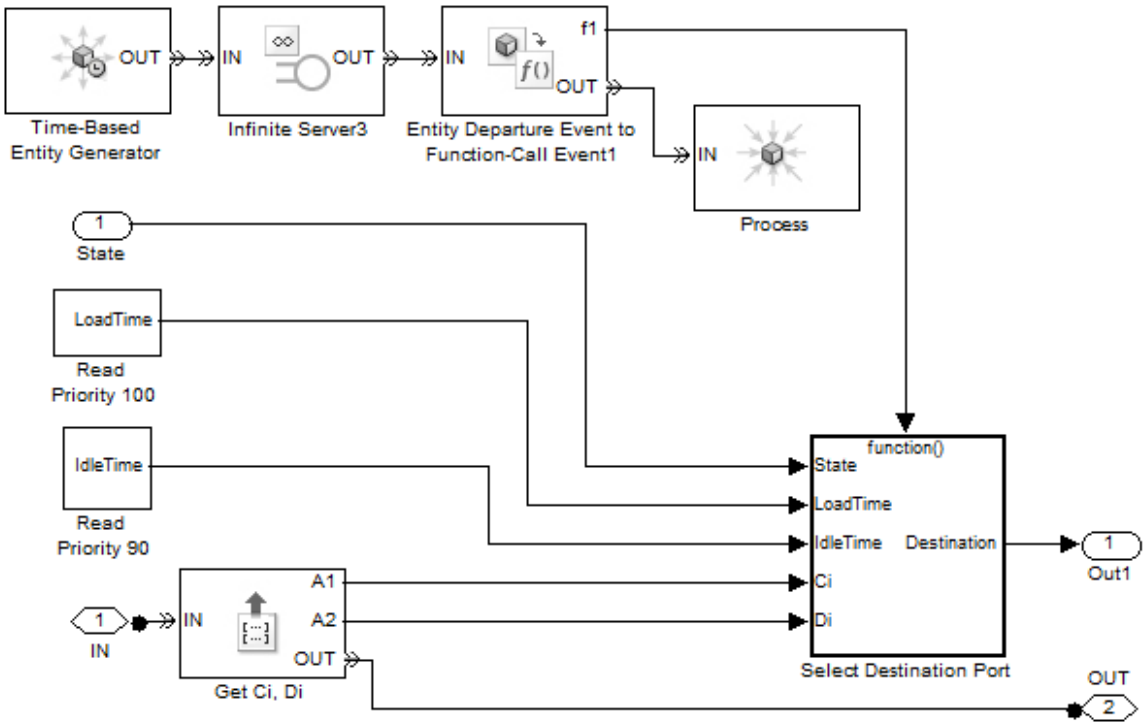

Figure 6

4.1.4 Location Policy Block

The internals of the location policy block are shown in figure 7. This block reads the node data table created by the information policy block to determine a suitable receive node for the outgoing task. If no eligible node is located, the task is discarded. Otherwise, the task is sent out onto the network. A unique ordered list of priority nodes is provided for each node to reduce thrashing.

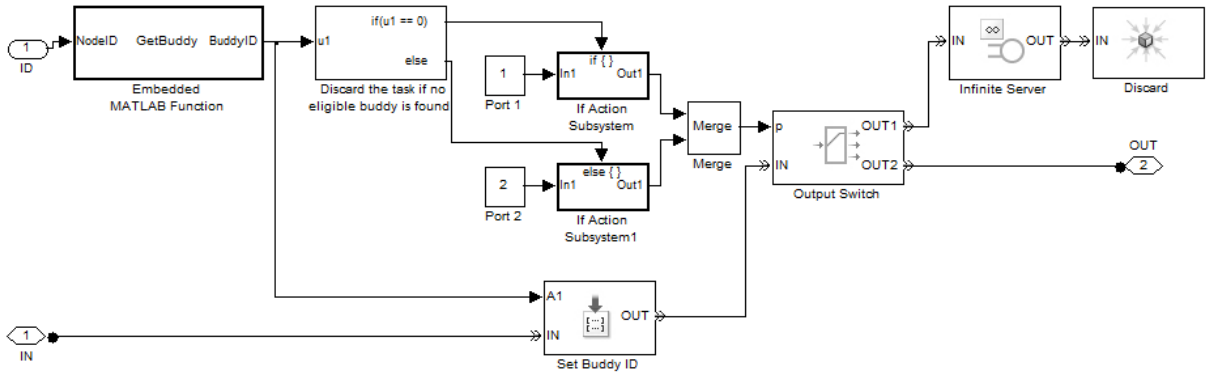

Figure 7

4.2 Experiments and Analysis

Experiments performed in this project include configuring the simulation model to contain four nodes and subjecting the nodes to a randomly distributed set of tasks with increasing average total load. Performance is analyzed for three different threshold levels under each load. Metrics measured include miss ratio and guarantee ratio.

Analysis shows that, as expected, guarantee ratios decrease as loading increases and miss ratios increase as loading increases. Interestingly, the miss ratio is nonzero at low average loads. This is likely attributed to a disadvantage of the Buddy Strategy with regard to how the load is computed. The disadvantage arises because the Buddy Strategy uses the number of tasks awaiting execution in the queue as its metric for determining state. Such a metric is not very sufficient for real-time systems since it does not take the deadlines or computation times of individual tasks into account. As an example, a node might receive several tasks with very small computation times or very long deadlines. The next task that arrives may very well be schedulable. However, since the number of tasks in the queue exceeds the overload threshold, the new task will be migrated. Alternately, a node might receive a small number of tasks with large computation times or short deadlines. When a new task arrives, it may not be schedulable, but since the number of tasks in the queue is small, the node is in an underloaded or fully loaded state, and the task will be discarded rather than migrated.

4.3 Performance Data

Performance data is recorded in figures 8 and 9. Average load is the ratio of the average task computation time and the average task deadline over time.

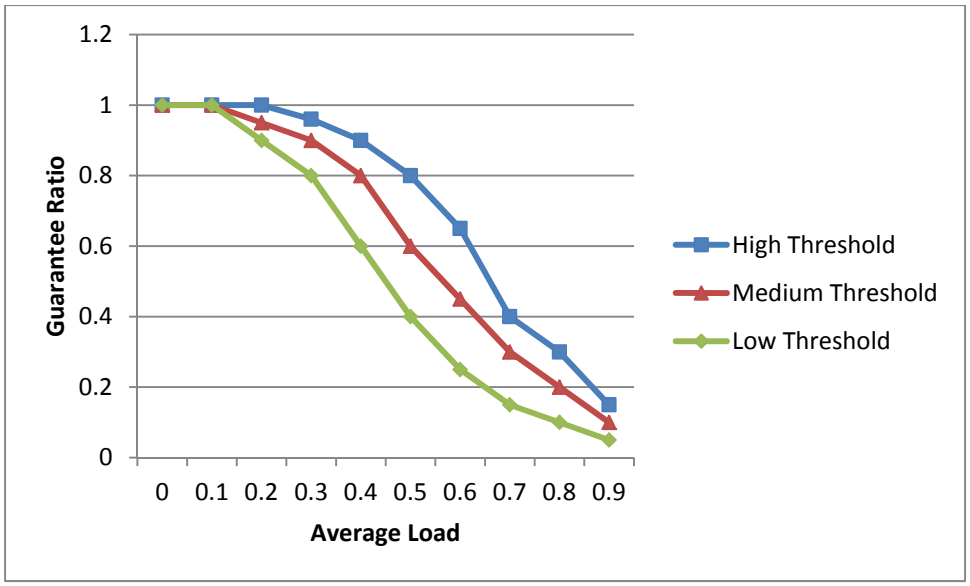

Figure 8

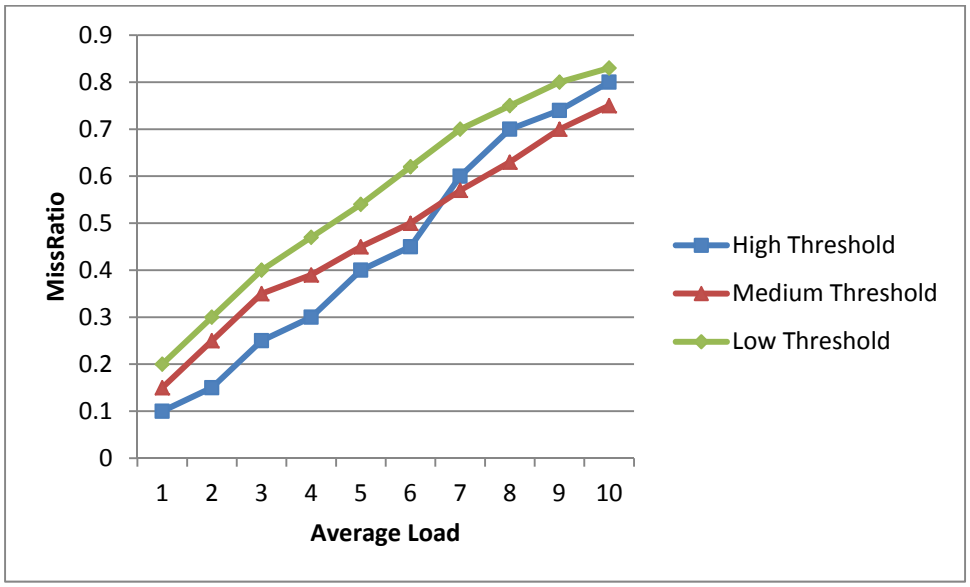

Figure 9

5. Conclusions

5.1 Report Conclusion

Distributed real-time systems are a valid solution to the dilemma posed by increasing performance requirements. They offer advantages such as redundancy and load balancing. Further, through simulation, the Buddy Strategy has been found to be a valid, though flawed, solution to the problem of implementing a distributed real-time system. Given the limited scope of this project, there are several

items to be addressed in future work. One area of future work is to experiment with different sized networks and strategies for creating buddy sets. Another area for future study is the possibility of implementing the Buddy Strategy with a modified transfer policy that uses a better real-time metric than the number of tasks currently awaiting execution in the queue.

5.2 Lessons Learned

Through undertaking this project, I have gained knowledge of how distributed real-time systems work. I have learned how the transfer policy, information policy, selection policy, and location policy interact in a system to create a global scheduler. I have gained experience using the simulation program Simulink to build models and perform experiments. Lastly, I have gained experience in planning projects and estimating required effort. I made the initial mistake of planning an overly ambitious project that involved implementation of a distributed real-time system on hardware running a real-time operating system and a real time Ethernet stack. This proved to be a significantly larger effort than was possible, and I was required to rethink my project and implement something with much more limited scope.

5.3 Instructor Feedback

These projects seem to be designed for two students. As an off-campus student, who did not know of any other off-campus students taking the course, I was unable to locate a partner, and I completed the project as an individual. Theoretically, working alone, I should do half the work of a pair of students. However, I was not entirely sure what the expectations were for an individual working on the project. For example, the suggested report length on the example report format webpage is 12-15 pages. I was unsure whether my expectation was to write a report that was half that length or some percentage of that length or the full length. This may be my own fault for not communicating my confusion, but perhaps this could be made clearer in the future.

References

[1] C. Murthy and G. Manimaran, *Resource Management in Real-Time Systems and Networks*

[2] L. DiPippo, V. Fay-Wolfe, J. Zhang, M. Murphy, and P. Gupta, “Patterns in Distributed Real-Time Scheduling”