Decision Models for Adaptive Resource Management in Multiprocessor Systems*

Daniel Paul†, Sudhakar Yalamanchili†, Karsten Schwan† and Rakesh Jha‡

† Georgia Institute of Technology
Atlanta, GA 30332
{ dpaul, sudha }@ee.gatech.edu
schwan@cc.gatech.edu

‡ Honeywell Technology Center
3660 Technology Dr.
Minneapolis, MN
jha@htc.honeywell.com

Abstract

This paper addresses the problem of effective, on-line adaptive resource management in parallel/distributed architectures. The class of applications is very data-dependent, resulting in highly dynamic demands for available resources. Adaptive resource management is an alternative to engineering real-time systems toward worst-case application behaviors. To meet performance constraints, the system must react swiftly to run-time load variations and accurately redistribute resources in real-time. We propose decision models that operate on dynamically monitored performance data to determine *when* resource reallocation is necessary. The proposed decision models operate at two levels. The first is a low-level approach involving a Bayesian probabilistic decision model. The second is a high-level approach based upon state transitions and a Markovian decision model. The framework for our evaluation is a synthetic environment capable of simulating event driven, multitask applications where each task is partitioned into subtasks executing on individual processors.

1 Introduction

This research addresses a class of parallel applications that can be modeled as a collection of multiple, precedence-constrained data-parallel tasks or stages. We encounter such classes of event driven, data dependent applications that are very sensitive to run-time changes in event rates and input data content [4]. Consequently, execution is heavily data dependent and imposes highly dynamic resource demands upon the host system. A primary example of relevant applications are real-time defense systems that must constantly react to changes in an external physical environment. The environmental changes result in highly data-dependent processing loads. For example, Automatic Target Recognition (ATR) systems experience widely varying processing loads as a result of their heavy dependence on scene and algorithm parameters [1]. As the distances to targets of interest fluctuate, the number of regions of interest to process changes, resulting in significant variation of the computational

*Funded in part by DARPA through the Honeywell Technology Center under Contract No. B09332478 and Contract No. B09333218, and by NSF equipment grants CDA-9501637, CDA-9422033 and ECS-9411846

load. Because of their data-dependent nature, the resource requirements of the parallel tasks will vary significantly during run time.

A possible solution to meeting the performance requirements of such applications is to statically assign enough resources to accommodate the worst case application behavior. This solution is often infeasible because of the excessive level of resources required to address all possible situations[4]. The alternative is adaptive resource management to re-allocate limited resources dynamically in response to an application's needs. In real-time environments, efficient, low latency reallocation is crucial to the ability of such applications to meet deadlines. A critical component of this reallocation process is the decision model that determines when a reallocation of resources is necessary.

Our work is based on the operational model depicted in Figure 1. Applications are modeled as an acyclic graph of data-parallel tasks. Data frames are pipelined through this graph and each of these data-parallel tasks can be further structured as a collection of subtasks, each running on an individual processor. The number of subtasks within a task varies as processors are dynamically allocated to and deallocated from the original task. The subtasks are instrumented to provide performance measurements in real-time. These instrumented streams of data are processed by detectors that produce detection events signaling major changes in performance metrics. Decision models process these streams of detection events to determine if resource reallocation is necessary, and if so, to initiate procedures for the computation and enactment of new reallocations. In this paper we only address the reallocation of processors among tasks to maintain a minimal frame latency through the task graph.

The majority of existing research on resource allocation and reallocation is focused on algorithms that determine *how* to most effectively allocate or reallocate resources. There is an extensive literature on dynamic resource allocation, typically in the context of load balancing algorithms (for example see [8, 15, 21, 12, 18, 9]). Strategies typically focus on *where* tasks must be scheduled as function of available resources. More recent research has studied dynamic processor scheduling algorithms in multiprocessor systems[14, 13] and even algorithms for dynamic control of communication resources[16] in parallel/distributed applications. These resource allocation algorithms rely on the existence of a mechanism that determines when they are invoked, for example, at task arrival time. This does not permit reaction to run-time load variations within the application. We argue that for run-time reallocation, it is critical to be able to determine *when* such resource reallocation algorithms must be invoked during task execution. Accurate timing can avoid thrashing during transient workload changes, permit low latency reallocation, and in some instances preempt performance degradation by predicting reallocation needs. This focus on decision models complements (and is distinguished from) the recent work on the online adaptation of systems for real-time applications[18, 19]. Such frameworks incorporate mechanisms for run-time monitoring, adaptation enactment, and processor reallocation. We argue that effective decisions models must be incorporated into such frameworks if they are to be successfully applied to online adaptive resource management functions.

This paper proposes the effective use of decision models for dynamic resource allocation in high-performance parallel/distributed systems. Specifically this paper proposes a combination of a low latency decision model that is reactive in nature with a (relatively) more complex decision model that is predictive in nature. We show that such a model is quite insensitive to transient workload shifts or "spikes", thereby reducing ineffective reallocations. The model is also quit effective in predicting impending workload changes. Experiments are presented that relate characteristic of the application, such as noise, and parameters of the decision model. Thus, the decision model can be "tuned" based on some knowledge of the application behavior. Using a synthetic benchmark generator, we experimentally demonstrate an increase in performance and a decrease in overhead across a range of input data parameters. While the current implementations are focused on a class of computationally

intensive sensor-processing applications, these decision models are more generally applicable to asynchronous, event-driven computational models. Throughout this paper the presentation will rely on an automatic target recognition (ATR) application to illustrate the behavior of the proposed model.

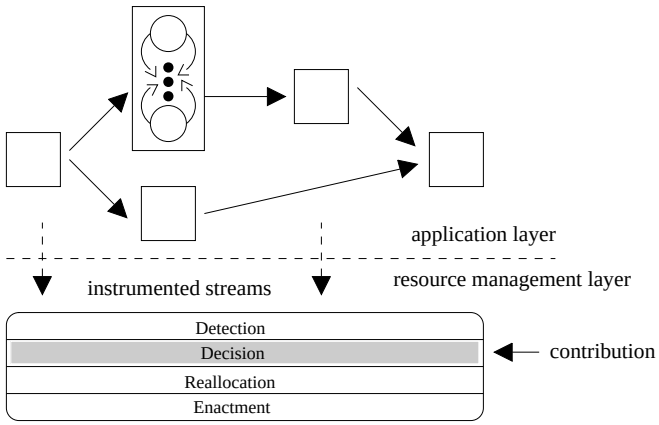

Figure 1: Operational model for dynamic resource allocation

The remainder of this paper is organized as follows. Section 2 provides a description of our system architecture and the problem we are addressing. In section 3, we provide a detailed description of our proposed decision models. Lastly, section 4 presents our experimental results and a review of the contributions of this paper.

2 Problem Description

2.1 System Overview

We consider an ATR application processing a stream of sensor data frames. The vision processing must extract targets from the background terrain and maintain track and identification information. The goal is to maintain a certain processing frame rate. As illustrated in Figure 1, resource reallocation is managed by a system consisting of four major components: detection, decision, reallocation, and enactment.

Currently, monitoring is accomplished by a real-time instrumentation system that can detect significant changes in a number of performance metrics [5] [6]. These monitors produce instrumented streams of sampled parameter values. Sample parameters include subtask execution time, subtask communication time, communication volume, input frame rates, and other measures of application performance or resource utilization. We may also choose to monitor application-specific measures such as the frequency of specific message types, access patterns to internal data structures or any other measure that is representative of the application's resource usage. Detectors operate on these streams to produce detection events corresponding to potentially significant deviations in performance guarantees. Decision models analyze streams of detection events to make assertions about the current global state and implicitly about the future state of the system. If a decision to reallocate is made, a cost evaluator creates a new specification of resource assignments. In this paper we only consider the (re)assignment of processors to tasks. Tasks are data parallel and the number of subtasks of a task is equal to the number of processors assigned to that the task. This new resource assignment must then be enacted and adopted by the system. In this case processors within one task may be reallocated to another task to maintain the frame rate.

2.2 Problem Definition

Monitoring and detection occur constantly as data frames move through the task pipelines. Decision models must constantly react to the detection event streams being produced by the detectors. However, the cost evaluation and resource reallocation only occur if the decision model signals the reallocation module. There is some computational overhead involved in calculating a new resource mapping, so ideally one would only incur this penalty under the assurance of improved overall system performance. Because of imperfect monitors and noisy input, it is difficult to determine accurately the optimal times to initiate a remapping. An effective reallocation decision policy must weigh the costs of remapping against the potential performance benefits [3]. In a worst-case scenario, a reallocation may be enacted only to discover that the previous conditions were transient and the system is more unbalanced than before remapping.

There are two competing factors governing the reallocation decision process. The first is the desire for fast detection and reaction. This is important because of the real-time requirements of the majority of these applications. Long and complex decision algorithms can result in large decision and enactment overheads. This overhead can cause multiple events to miss their deadlines because of quickly changing environmental conditions. In addition, many of the dynamic input conditions are highly transient and unstable. For example, background foliage that appears only in a few successive image frames can produce sudden, transient change in the processing workload. Therefore, the second important factor is the global performance of the application using the new resource mapping. Ideally, new resource mappings will lead to configurations which are stable and improve the long-term performance of the application. While quick decisions may result in locally improved performance, they are by nature not globally cognizant, i.e., will the change in terrain features persist for a relatively long period of time? Thus, it is possible to continually make locally optimal task-based decisions while the overall performance of the application steadily moves toward a less efficient state.

2.3 Solution Strategy/Approach

This paper proposes the use of a decision model structured as two component models. The first is a *Bayesian decision model*, which operates at the lower-level of the decision process. This probabilistic model acts as a filter between the monitoring system and the reallocation module to reduce false detection and incorrect decisions. This will increase the stability of the application and reduce the amount of unnecessary overhead incurred by reallocation in response to false detections. The second is a *Markovian decision model* which operates at a higher level of the decision process. The Markovian model is designed to keep track of global application performance by monitoring the state transitions of various performance metrics. Using the state transition data, the Markovian model is able to predict the steady-state system performance and react to potential future performance degradations.

3 Decision Models

The simplest decision model in our framework (Figure 1) has been optimized for low latency rather than high accuracy decisions. In this model, referred to as the *baseline model*, any detection event immediately triggers a cost evaluation and potential reallocation. It is apparent that this decision model can lead to unnecessary overhead, in part because the cost evaluation penalty is incurred *every* time a detection event occurs. Often, because of imperfect monitors, a detection event is not indicative of the overall performance falling beneath the required limit. In this situation, referred to as *false detection*, the overhead

involved in cost evaluation is incurred with no potential for benefit. In response, we propose a two-level decision model for such systems. The first model is an adaptation of a Bayesian decision model originally formulated by Nicol [2]. The second is a Markovian decision model formulated to predict global trends in application performance.

3.1 Bayesian model

There are several sources of difficulty that can make the baseline decision model ineffective. First, the application may be subject to transient load spikes. Since the application is data dependent, it is sensitive to any changes in the input stream. A problem occurs with noisy data or "spikes" that represent transient load changes rather than a stable shift in computational loads. In such situations, a detection event may not be indicative of a longer term change in load, necessitating a resource reallocation. Rather, it may represent a transient condition, in which case a reallocation may actually negatively affect performance. In terms of our ATR example, changes in scenery can result in input spikes. In the absence of new targets, these spikes are transient and should not be grounds for reallocation. An effective resource management system must be capable of discerning stable computational shifts from spikes. Second, the detectors themselves possess some degree of unreliability. It is possible for detectors both to generate a false detection event or fail to detect a genuine detection event. Increasingly accurate detectors are computationally intensive and increase the latency between the occurrence of a load imbalance and corresponding detection. On the other hand, simple load detectors may not be effective in accurately detecting stable load changes. They can significantly raise computation overhead by increasing the number of false detections. The proposed Bayesian decision model will allow for quick detection of critical events using simpler detectors while reducing reports of false detections.

3.1.1 Model description

Faced with computational overhead and potentially poor performance in the event of an unnecessary reallocation, it is not beneficial to run the cost evaluator on the basis of a single positive report. As illustrated in Figure 2, the Bayesian decision model adds an extra component, operating as a *smart filter*, to the system. In this proposed configuration, a collection of monitors still record execution parameters as frames pass through the application. However, in this scheme, all detection events pass through the smart filter. The smart filter uses a Bayesian decision model to determine if the cost evaluator should be invoked. It is the goal of the smart filter to minimize the number of false detections passed to the cost evaluator. Ideally, the cost evaluator will only be signaled if a potential remapping benefit is very likely. Conversely, whenever a remapping benefit becomes likely, the cost evaluator should be signaled as soon as possible.

At each frame time, we compute the probability that performance can be improved by reallocating resources. This probability is referred to as the *gain probability* and is represented by the symbol p_n [2]. The Bayesian decision process must constantly strengthen or weaken the gain probability based only on information from the detectors and information about the quality of their detections. These detectors operate on the instrumented streams returned from the monitors and look for various metrics at the task level. While the overall application behavior may be within real-time bounds, the detectors can notice if a specific task's performance starts to decrease. By operating at the lower level, this model can determine improved resource mappings even if the real-time requirements are not immediately threatened.

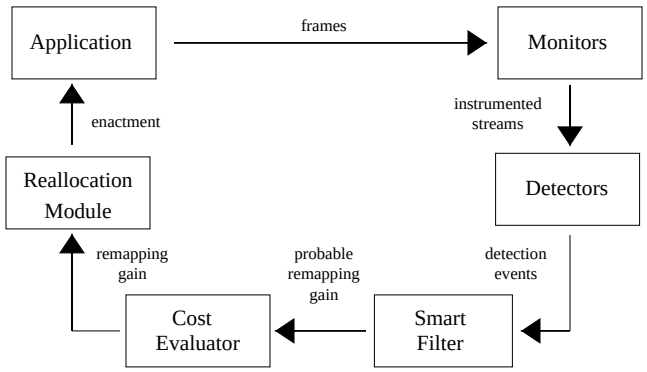

Figure 2: Block diagram of the Bayesian decision model.

3.1.2 Model calculation

To implement this Bayesian model, three distinct parameters must be defined [2].

1. ϕ – The probability that a performance gain can be realized by remapping after the current frame, given that it has not been realizable in the previous frame.
2. α – The probability of the detectors prematurely reporting a remapping condition.
3. β – The probability of the detectors failing to report an existing remapping condition.

The parameters α and β encompass the fact that the detectors contain some inherent inaccuracy. Based on the above parameters, we calculate the probability p_n that a performance gain can be realized on frame n based upon the results returned by the detectors.

First, we define the *pretest probability*, p_a, that a performance gain is achievable on frame n, given that $p_{n-1} = p$:

$$p_a(p) = p + (1 - p)\phi$$

Bayes' Theorem states that for two events A and B, the probability of A given B is:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B \mid A) \cdot P(A) + P(B \mid A^C) \cdot P(A^C)}$$

Now, if the detectors return a positive remapping report, we want to calculate the probability that an actual performance gain exists. Using the above notation, A is the event where a performance gain exists and B is the event of a positive report. Therefore, $P(A)$ is given by the pretest probability, p_a, and $P(B \mid A)$ is given by the probability of an accurate positive report. Since β is the probability of the detectors failing to report a gain condition, $(1-\beta)$ is the probability that the detectors accurately report a gain condition. $P(A^C)$ is the complement of $P(A)$, which is given by $(1 - p_a)$. Lastly, $P(B \mid A^C)$ is the probability that a positive report is returned given that no remapping gain exists. Recall that this is the exact definition of the parameter α. Substituting these expressions into Bayes' Theorem gives us the following gain probability for a positive report:

$$p_n = \frac{(1 - \beta) \cdot p_a(p)}{(1 - \beta) \cdot p_a(p) + \alpha \cdot (1 - p_a(p))}$$

Using similar logic, the gain probability in the event of a negative report can be computed as follows:

$$p_n = \frac{\beta \cdot p_a(p)}{\beta \cdot p_a(p) + (1-\alpha) \cdot (1 - p_a(p))}$$

In either case, this probability, p_n, represents a weighted measure of the potential for a remapping gain to exist after frame n. When p_n crosses a suitable threshold level, it is deemed likely that a remapping gain exists. This in turn justifies a cost evaluation to determine if a better resource allocation is realizable. If so, the reallocation module is signaled and the remapping is enacted.

3.2 Markovian model

Bayesian decision models are sensitive to the accuracy of model parameters and as such are implemented at a low level within the application. These models tend to be *reactive* as they wait for events signaled by the detectors. At a higher level of abstraction, we can track the application performance based on the degree to which user specified performance bounds are met. Toward this end, we formulate a Markovian decision model. This model is *predictive* as it uses performance state data to predict the future behavior of the application. In terms of our ATR application, the Markovian model analyzes scenic trends. While an individual scene may not provide much information, a collection of scenes can provide insight about future direction and the resources that may be required by specific tasks. The Markovian model uses these trends to predict the need for a resource reallocation before it actually occurs.

3.2.1 Model description

By working at the lower level, the Bayesian model is able to detect improved resource mappings even when the application is conforming to the real-time specifications. The Markovian model, which operates at a higher level, is triggered solely by the level of conformity with the real-time bounds. It will only trigger a remapping if the application is predicted to violate these bounds with a high probability. The Markovian decision model can be viewed as a *watchdog* for the Bayesian model. If the Bayesian model is able to maintain performance within the desired specifications, the Markovian model will never intercede. However, if the system performance appears to threaten the real-time specifications, the Markovian model will override the Bayesian model and force a resource reallocation. An example of this process is presented in Section 3.2.2.

Figure 3 presents a block diagram of the system incorporating the Markovian decision model. Its primary function is to monitor specific evaluation metrics of global application performance, e.g. end-to-end frame latency. This is done by comparing actual measured statistics with the real-time specifications. The ratio of the measured statistic to the desired bound serves as a metric of the level of conformity of the application. This level of conformity is then used to map the application into one of a set of previously defined *performance states*. These performance states and the transitions between them provide the underlying framework of the Markovian model.

The inherent differences between the two decision models enable them to be coupled in a synergistic manner. The Bayesian model is constantly checking the detection streams and updating the gain probability. When the gain probability exceeds a threshold, it signifies a high potential for a remapping gain. To take full advantage of this potential and reduce the chance of missed deadlines, the Bayesian model is coupled with a simple reallocation

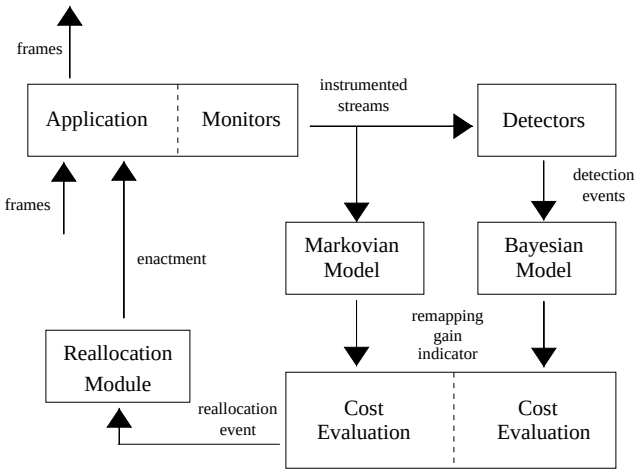

Figure 3: Block diagram of the coupled decision model.

algorithm to provide a low latency solution. While these decisions allow a quick response and often provide an immediate improvement, they may not represent the best resource allocation. To correct for this, the Markovian model is invoked at periodic intervals. If the predicted performance of the application is sufficiently poor, a (relatively) more complex reallocation algorithm can be invoked. This algorithm performs a more extensive (and therefore costlier) assessment of remapping potential. In these instances, a high quality remapping which can provide system-wide improvement is more effective than a high speed decision which can provide immediate but local improvement.

3.2.2 Model calculation

We first provide an intuition about the application of the model through the following example. Consider the state space of an ATR application as represented in Figure 4 where we wish to maintain a frame analysis rate above 33 frames/sec. Therefore, we must maintain end-to-end frame latencies below 0.03 seconds. The maximum frame latency we expect is .10 seconds and the latency range is divided into five states. States 0, 1, and 2 are considered acceptable and states 3 and 4 are considered unacceptable.

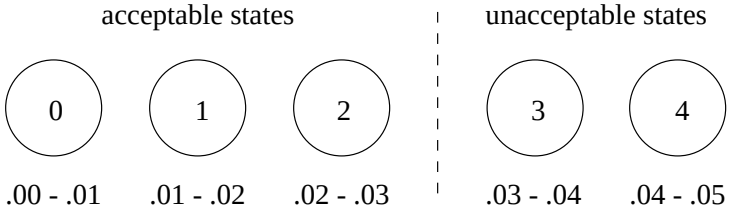

Figure 4: A high level example of the Markovian decision process.

Over time, frames are periodically generated by the source and injected into the system. At specific instances, referred to as *frame intervals*, completed frames are consumed at the sink. The time between frame generation and consumption is the end-to-end frame latency which we are trying to control. At each frame interval, the monitored information streams can be used to generate a current snapshot of the application in terms of frame latency

performance. This snapshot is used to categorize the system as residing in a particular "performance state." As shown in Figure 4, the range of performance states is easily partitioned into a collection of acceptable and unacceptable states. As the data content of the input frames varies over time, the current performance state of the system will fluctuate. Resource reallocation is used to improve the performance of the system when it resides in an unacceptable state. The goal of the Markovian model is to predict when the system is moving toward an unacceptable state and to avoid it by triggering a resource reallocation in advance. This is accomplished by recording statistics about the state transitions experienced by the application. Using the state transition information and Markov theory, we can predict the steady-state behavior of the application. At periodic intervals, this steady-state prediction is used to determine if resource reallocation is necessary. This interval is referred to as the *Markov invocation interval*.

The following description is based on sensor applications where data frames are arriving at some rate. The model can be generalized in a straightforward manner to more general event driven applications [4] where the events such as a frames may arrive asynchronously rather than at a fixed rate.

Given the number of states (n), the maximum performance measurement (μ) and the current performance measurement (ρ), the current state is determined by:

$$\frac{\rho \cdot n}{\mu}$$

To ensure proper operation, any performance measurement exceeding μ is automatically categorized into the largest state. Note that in this model, performance metrics are mapped to states numbered from 0 to $n-1$. There is a natural mapping for metrics such as latency, since higher latency values map to higher numbered states. The mapping may be different for metrics such as throughput where lower throughput values map to higher numbered states so that we have a consistent interpretation of higher numbered states corresponding to lower performance.

The Markovian decision model tracks the efficiency of the current mapping by calculating the performance state of the system as described above. These performance states can be viewed as a Markov chain, with the application transitioning between them at each frame time based upon the current data and resource mapping. As the application moves between the performance states, the Markov model maintains statistics about the state transitions. With this information, at any point during execution, the Markov model has an accurate picture of the state transition probabilities of the Markov chain. These transition probabilities are conditional probabilities for the system to transition to a particular state, given the current state of the system. If there are n states in the Markov chain, than the collection of all possible one-step transitions can be collected in an $n \times n$ matrix called the *state transition matrix*. This state transition matrix can then be used to calculate the the *steady-state probability vector*:

$$\mathbf{S} = [p_0 \; p_1 \; p_2 \; \cdots \; p_{n-2} \; p_{n-1}]$$

The steady-state probabilities predict, based on the current picture of the system, the probability of the system settling in each state. This provides a measure of the long-term global application behavior. While the Bayesian attempts to make locally optimal decisions, the steady-state probabilities may show that the system is heading towards an increasingly unbalanced state. The steady-state probabilities are examined to determine whether reallocation is necessary. We use the following approach which accounts for the gradient of unacceptable states. A detailed description of the model can be found in [7].

Using the steady-state vector, the following inner product is computed.

$$\omega = \sum_{i=1}^{n} p_i \cdot i$$

The above computation produces a weighted state, ω. If this weighted state falls within the set of undesirable states, then a reallocation is invoked. By basing its decisions upon the steady-state probabilities, the Markovian decision model represents a *predictive* process, while the Bayesian decision model represents a *reactive* process.

3.3 Model Parameters

For these decision models to correctly determine when a remapping gain is probable, they must have accurate knowledge of certain application-specific parameters. For the Bayesian model, these are ϕ, α, and β as discussed earlier, and they are measures of how often a remapping gain is achievable and how accurately the detectors can recognize this situation. For the Markovian model, the number and range of performance states and the threshold between acceptable and unacceptable states must be defined. In addition, the statistics for transitions between these states and the duration of the interval between Markov invocations are necessary for calculating the steady-state distribution.

A detailed explanation of the experiments to determine these parameter values can be found in [7]. For our experiments, the following values were used for the Bayesian parameters:

$$\alpha = 0.125037 \quad \beta = 0.34588 \quad \phi = 0.09500$$

Our experiments use Markov chains consisting of 20 states representing the frame latency. To provide a reasonable tolerance for an occasional missed deadline and a stricter tolerance for any consecutive misses, the Markov demarcation threshold was chosen to be 12. The frequency of change in the input stream is characterized by a parameter called the *stability interval*. The stability interval refers to the number of frames over which the application remains stable, i.e., does not require reallocation. As the Markovian decision model is essentially *sampling* the input stream at each invocation, this translates to a Markovian invocation interval of half the length of the input stability interval.

4 Performance Evaluation

This section compares the performance of the decision models under varying input conditions. Our experimental platform consisted of two components. An 8-node IBM SP-2 was used for our empirical studies which generated the application specific model parameters. The "ATR application" was a synthetic workload generator that can be configured to represent a range of ATR workloads. Simulations were run with the synthetic workload generator running on a uniprocessor. Our synthetic workload generator allows us to compare the different models while changing a number of important input characteristics. The two primary input characteristics we investigated were rate of change of workload and the presence of noise. Rate of change refers to how frequently the input frames cause substantial workload changes in the tasks of the application requiring reallocation. Noise refers to both the frequency and size of input workload spikes. These parameters were chosen because they are most applicable to the types of event-driven applications that these models are designed to improve. They are also characteristics for which values can often be ascertained a priori. For example we may know the maximum rate at which new targets can appear in the scene or we may know something about he quality of the detectors and sensors, or be familiar with texture of the

terrain. The factors affect the presence of workload spikes and rate at which we can expect stable shifts in workload.

Three sets of experiments were performed in this section. The first set of experiments were designed to evaluate the improvement of the Bayesian decision model over the simple model used as our baseline. The second set of experiments were designed to illustrate the predictive capabilities of the Markovian decision model and its ability to improve application performance when used in conjunction with the Bayesian model. The final set of experiments test the fully coupled Bayesian and Markovian models under various input conditions characterized by their rate of change and the presence of noise. For comparison purposes, the baseline decision model refers to the model where every detection event invokes a cost evaluation.

4.1 Bayesian model

The following graphs illustrate performance improvements provided by the Bayesian model over the baseline decision model. Figure 5 demonstrates the reduction in false detection percentage achieved by the Bayesian model. Figures 6 and 7 indicate the end-to-end latency of the injected frames and the number of resource reallocations required to achieve that latency using each decision model. These graphs provide a comparison of the number of resource reallocations enacted by the decision models in response to identical input streams. Figure 8 plots the end-to-end frame latencies for both models on the same axes. This graph is used to compare the overall performance of the two models.

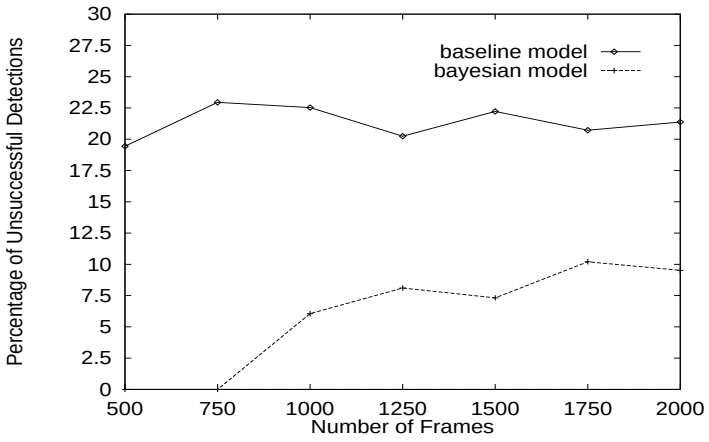

Figure 5: An illustration of the false detection rate for the two decision models.

It is apparent from Figure 5 that the Bayesian decision model significantly reduces the percentage of unnecessary invocations of the cost evaluator. The smart filter is able to successfully pare the number of detection events that will not result in a remapping gain, thereby reducing the amount of unnecessary cost evaluation overhead. Furthermore, by decreasing the the total number resource reallocations, the Bayesian model also reduces the amount of unnecessary reallocation overhead. This is accomplished by filtering the detectors response to noise and input spikes. The baseline decision model often reacts to input spikes by reallocating resources at frames where the spike is detected and on immediately successive frames. This behavior represents unnecessary reallocation overhead as the input spikes are transient and provide very limited rewards following a resource reallocation.

The combination of reducing both false detection percentage and unnecessary reallocations improves the behavior of the application in a number of ways. First, less computation time is spent in the cost evaluation module, and ultimately more processing power can be used for useful computation. Second, by ensuring that the cost evaluator is invoked only when there is a high probability for a remapping gain, a more complex evaluation mechanism can be enacted with (relatively) less overhead. The following two figures compare the number of resource reallocations required by the baseline and Bayesian models and the latency characteristics under which the reallocations occur.

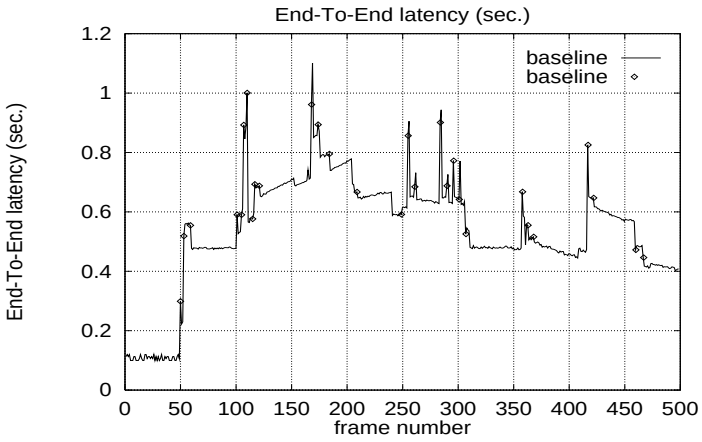

Figure 6: An illustration of frame latency and reallocation points for the *baseline model*.

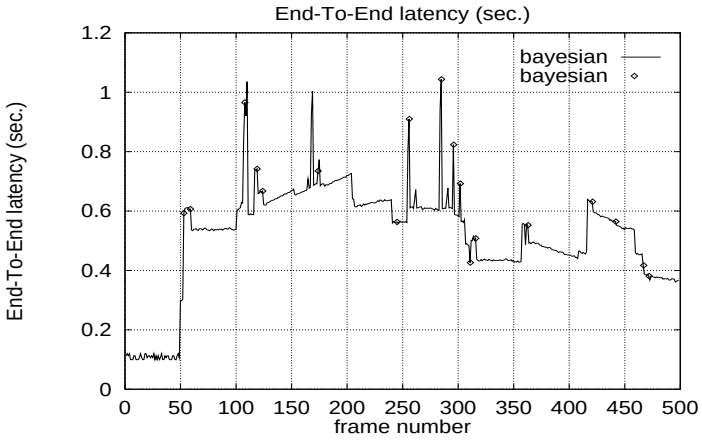

Figure 7: An illustration of frame latency and reallocation points for the *bayesian model*.

To fully understand the benefits of the Bayesian decision model, one must look at the remapping behavior of the system along with the frame latency through the system. Figure 6 shows that the baseline decision model remaps a total of 29 times, while Figure 7 shows that the Bayesian decision model remaps a total of 18 times. This reduction in the number of resource reallocations primarily results from the filtering of detection events directly associated with the presence of input spikes.

Because of their transient nature, input spikes cause detection events based upon conditions that do not persist after the short duration of the spike. Spikes can lead to new

resource allocations predicated on information which is not indicative of a long-term behavioral change. Resource reallocations resulting from input spikes do not provide large enough performance benefits to justify their enactment overhead. However, given that an input spike has already caused a remapping, it may not be necessary to automatically remap after the spike to correct the situation. A spike-based reallocation may not provide an increase in performance, but it also may not perturb the system enough to cause a decrease in performance. It is not always beneficial to reallocate resources immediately following an input spike because the benefits may not outweigh the enactment overhead. Because input spikes produce easily detectable changes both on the way up and on the way down, the simple decision model often invokes a resource reallocation both before and after the spike. The smart filter embedded in the Bayesian decision model allows it to reduce the number of initial reactions to input spikes. Despite this, large or long-lasting spikes will cause the Bayesian model to react. In these situations, the Bayesian model may also filter post-spike reactions if the performance benefits are not significant enough to warrant a resource reallocation. By utilizing both pre- and post-spike filtering, the Bayesian model reduces the total number of resource reallocations and the effect of their enactment overhead on frame latency. Figure 8 shows a comparison of end-to-end frames latency for the baseline and the Bayesian decision models.

Figure 8 demonstrates that the Bayesian model provides consistently improved latency performance over the course of the application's execution. The Bayesian decision model spends less time reacting to input spikes and makes smarter decisions than the baseline model. The benefits of the Bayesian decision model are twofold. First, it can provide improved end-to-end latency performance during execution. Second, it significantly reduces the number of resource reallocations necessary to provide this performance.

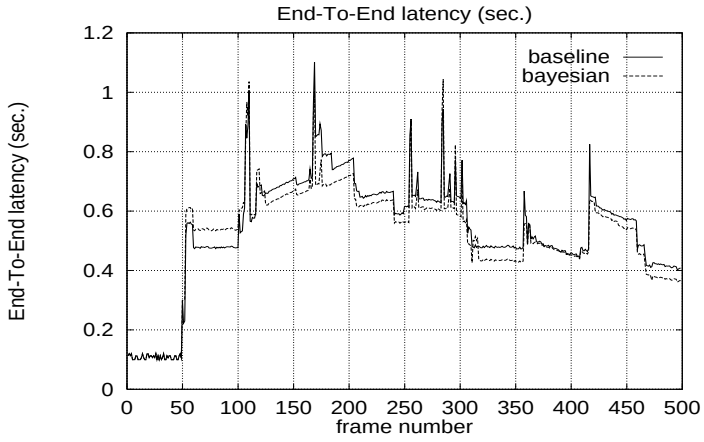

Figure 8: A comparison of frame latencies for the baseline and Bayesian models

The ability to filter the effects of input spikes is at the heart of the Bayesian model's improved performance. A trade-off exists between filtering input spikes and reacting to real load changes. As shown in Figure 7, the Bayesian model is not able to filter very large or prolonged input spikes. Attempting to filter these conditions could result in a large number of real input load changes being undetected. This potential is illustrated in Figure 8 during frames 50 through 100. This is the only case in our tests for which the simple model provided better performance than the Bayesian model. In this situation, the Bayesian model filtered a detection event signifying a stable load change and was therefore slow to react to the actual conditions. The profiling scheme used to fine tune the Bayesian parameters to

this application limits the occurrence of this situation to an initial startup transient of the Bayesian model or after extended periods of stable input with no activity. An important feature of the Markovian model is its ability to detect these remaining occurrences and reduce their effect to negligible levels.

4.2 Addition of the Markovian model

This section presents the results of coupling the Bayesian and Markovian decision models. Figure 9 demonstrates the potential of the coupled model over the pure Bayesian model.

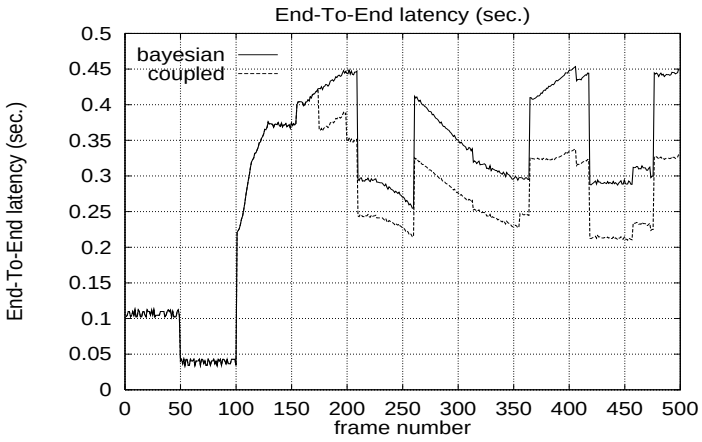

Figure 9: A comparison of frame latencies between the *Bayesian model* **and the** *coupled model*

As illustrated, the two experiments show similar performance until frame 175. Up to this point, the application has been conforming to the real-time specifications and the Markovian model invocations have continually predicted acceptable performance. All resource reallocation have resulted from the Bayesian models. At frame 175, the coupled system invokes the Markovian model and it predicts that system performance is heading toward an unacceptable state. This results in a remapping which is represented by the latency drop in Figure 9. Following this decision, the coupled model has a better resource allocation than the pure Bayesian model. This is confirmed by the lower end-to-end frame latency shown in Figure 9. For the next 175 frames, the Markovian model is again inactive as the application is within the real-time bounds. Both systems are again making purely Bayesian decisions, which accounts for the similarity in the shapes of the two graphs. When the Markovian model is invoked at frame 350, it again predicts that the system is heading toward an unacceptable state. Another Markovian reallocation occurs, which accounts for the slight deviation in the shape of the graphs between frames 350 and 400. The pure Bayesian system shows an increasing latency slope beginning around frame 360, while the coupled system does not show an increase in latency until around frame 385.

These results clearly show the ability of the Markovian model to monitor the global application performance and initiate a new resource allocation when the real-time specifications are threatened. This predictive capacity makes the Markovian model perfectly suited to act as a watchdog over the Bayesian model.

4.3 Coupled model performance

This section presents a final set of results. We have demonstrated both the benefits of the Bayesian model over the baseline model and the benefits of adding a Markovian watchdog to the Bayesian process. We now investigate the performance of the coupled decision model with respect to two important input parameters: input rate of change and noise. These two conditions represent important dimensions of the applications that may utilize adaptive resource management. For our ATR application, we want to maintain an acceptable frame rate in the presence of both a high number of targets (input rate) and a large amount of scenic variations (input noise).

A particular input stream can be described in terms of the range of rate of change and noise. We studied the performance of the coupled model under average and extreme conditions as illustrated in in Figure 10.

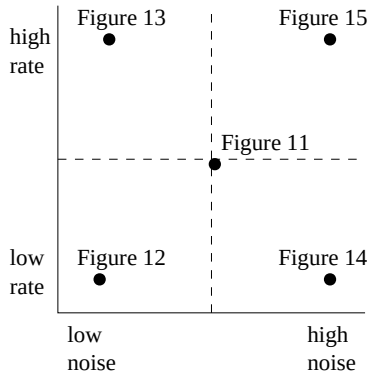

Figure 10: A representation of the input characteristics for our experiments and reference to the corresponding figures.

Our first experiments consisted of testing the coupled decision model on "average" input streams with median values for rate of change and noise. The input streams used in these experiments have stability intervals on the order of 50 frames and input spike probabilities on the order of 15 percent. Performance results from these experiments comparing the behavior of the coupled and baseline models are provided in Figure 11.

As evidenced in Figure 11, the coupled decision model is able to improve end-to-end frame latency throughout the course of execution. Under conditions containing median levels for both input rate of change and noise, both the Bayesian and the Markovian models contribute toward improved performance. By reducing the total number of decisions and improving decision quality and timing, latency performance is improved while decision and enactment overhead is reduced.

The following two experiments investigate input streams with a low probability of input noise. The average probability of input spikes used in these experiments was 5 percent. We further divided these experiments into input streams containing either a high or low rate of input change. Low rate of change input streams used stability intervals on the order of 100 frames, while high rate of change input streams used stability intervals on the order of 20 frames. Figure 12 compares the results from the low noise and low rate of change experiments, and Figure 13 compares the results from the low noise and high rate of change experiments.

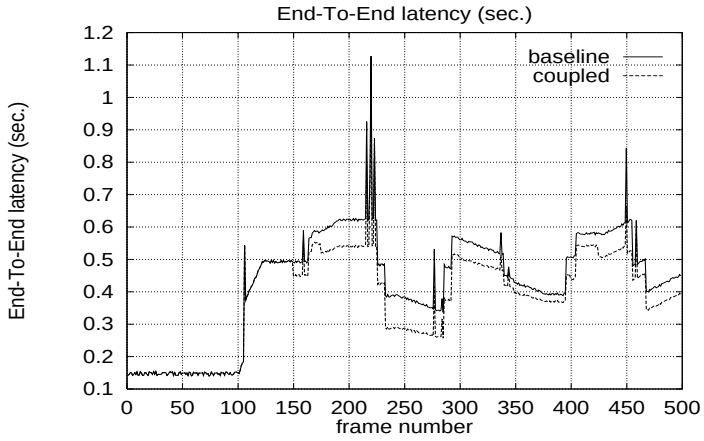

Figure 11: A comparison of frame latencies between the baseline model and the coupled model under *average* input conditions.

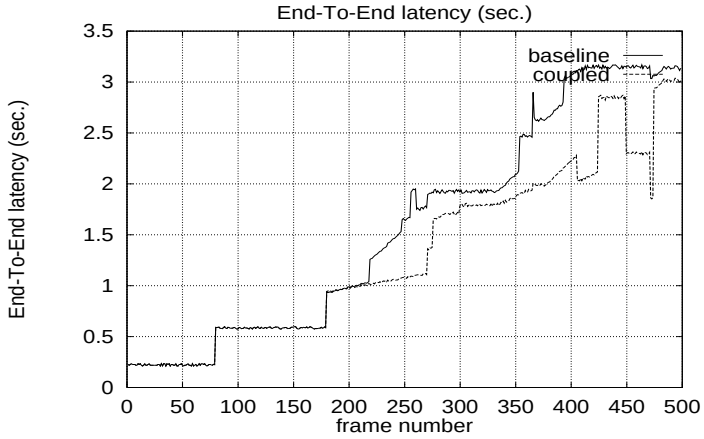

Figure 12: A comparison of frame latencies between the baseline model and the coupled model with *low rate* and *low noise*.

Under low noise conditions, the coupled decision model performs significantly better than the baseline decision model. The reduction in input spikes resulting from the low noise behavior reduces the dependence on the Bayesian smart filter. In these experiments, the Markovian model contributes more significant decisions than the Bayesian model. Low levels of noise increase the accuracy of the Markovian predictions. This allows the Markov model to make better decisions about when and where to best allocate resources. Figure 12 demonstrates that good Markovian decisions can significantly improve the performance in a low rate of change input stream. Because of the low rate of change, both the baseline and Bayesian models do not receive many detection events and therefore do not trigger/enact many reallocations. The Markovian model is able to push the system into a more globally efficient state which persists because of the low input variation. Figure 13 demonstrates that in a low noise environment the coupled decision model is also effective for a high rate of input change. By adjusting the sampling frequency of the Markovian model to account for the increased input data rate, the coupled model is able improve the performance of the application over a large number of frames.

The final two experiments investigate input streams with a high probability of input noise. The average probability of input spikes used in these experiments was 50 percent. We further divided these experiments into input streams containing either a high or low rate

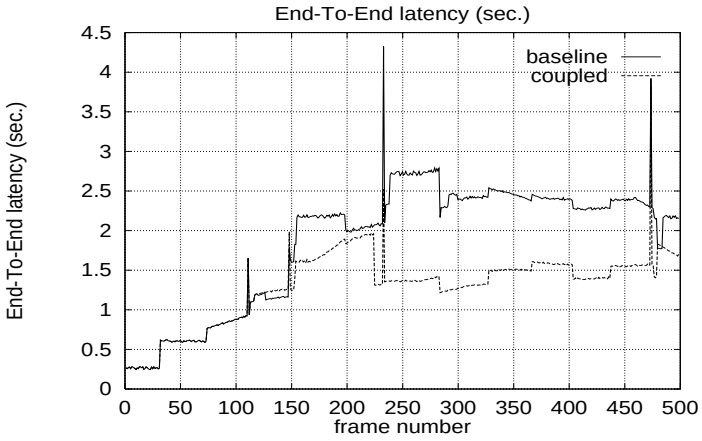

Figure 13: A comparison of frame latencies between the baseline model and the coupled model with *high rate* and *low noise*.

of input change. Low rate of change input streams used stability intervals on the order of 100 frames, while high rate of change input streams used stability intervals on the order of 20 frames. Figure 14 compares the results from the high noise and low rate of change experiments, and Figure 15 compares the results from the high noise and high rate of change experiments.

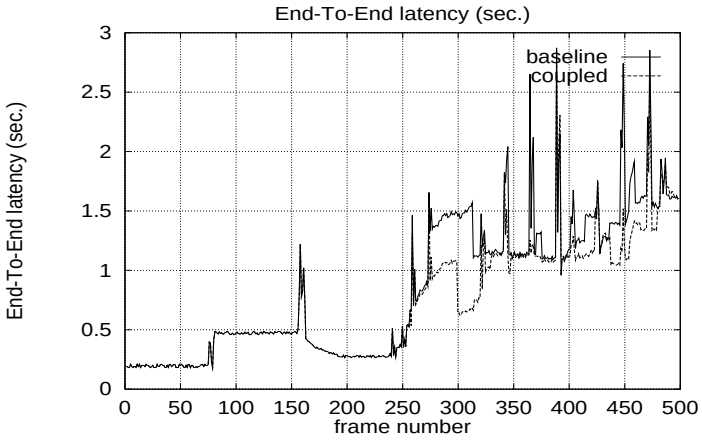

Figure 14: A comparison of frame latencies between the baseline model and the coupled model with *low rate* and *high noise*.

Because of the high levels of noise in these two experiments, the improvements in end-to-end frame latency are not as prominent as in previous experiments. Under these input conditions, the Bayesian model is more effective than the Markovian model. The frequency of the input spikes limits the accuracy of the Markovian predictions. The increased noise also results in an greater number of lower-level resource reallocations. These additional reallocations ensure that the Markov statistics are initialized more frequently, thereby limiting the model's effectiveness as a predictor. However, the Markovian model does serve an important purpose in these experiments. As the embedded smart filter in the Bayesian model attempts to filter out the input noise, the Markovian model acts as a backup to ensure that it does not filter any significant load changes. Any sustained performance levels violating the real-time specifications will still be corrected by the Markovian model.

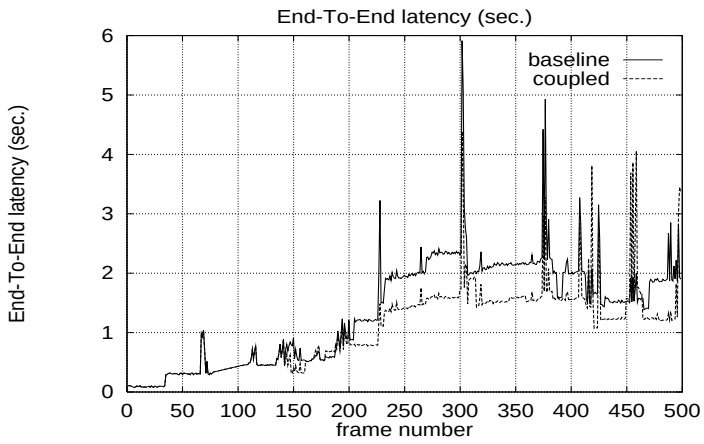

Figure 15: A comparison of frame latencies between the baseline model and the coupled model with *high rate* and *high noise*.

While these figures demonstrate improved latency performance for the coupled decision model, additional benefits are not apparent from the graphs. The filtering properties of the Bayesian model along with the Markovian backup allow the coupled model to achieve better latency performance in a fraction of the cost evaluations and resource reallocations required by the baseline model.

Our experiments demonstrate the effectiveness of the coupled model across a range of input conditions. The total number of resource reallocations and the number of false detections are both significantly reduced. These reductions are accomplished while maintaining improved latency performance. The reduction in both detection and enactment overhead allow more processing cycles for useful work without sacrificing any latency performance. This is significant in that these experiments are not based on a fixed number of processor cycles distributed between useful computation and allocation. In a practical implementation with real applications, we expect that the latency improvements using a fixed number of processors will be greater since the cycles saved by effective detection and prediction will directly reduce the latency.

5 Conclusions and Future Research

Clearly, there is a need for efficient adaptive resource mechanisms to be used with data-dependent, real-time applications. The mechanisms must be responsive to change and yet accurate in their remapping requests. These quality requirements place a great deal of pressure on the remapping decision model. Current implementations of simple decision models might not be able to meet increasingly stringent real-time requirements. This paper proposes an improved decision process to provide increasingly accurate resource mappings while maintaining low decision latency and overhead.

Experiments using a synthetic workload generator and the statically defined model parameters yielded promising results in multiple categories. An overall reduction in the percentage of unsuccessful invocations of the cost evaluator and number of unnecessary resource reallocations was realized with the Bayesian decision model. This allows more cycles for useful computation and can mask the use of the more complex Markovian decision process. Experiments with frame latency showed similar or improved performance compared with the simple decision model for a significantly lower number of remappings.

By coupling the reactive Bayesian model with the predictive Markovian model, we create a multi-level decision model capable of improving the performance of adaptive resource

managers under a variety of input conditions. Under average input conditions, both models contribute to decrease the end-to-end latency of input frames and reduce the decision and enactment overhead. Toward the extremes, the Bayesian model proves more applicable to high noise environments and the Markovian model better suited to low noise environments. In these situations, the less suited model provides good backup support for the more effective model. Under low noise conditions, the Bayesian level keeps track with the baseline model while the Markovian level pushed the system toward more acceptable performance states. Under high noise conditions, the Bayesian level filters a much larger percentage of the input spikes while the Markovian level ensured performance did not fall below the real-time specifications. Over a wide range of input streams, the coupled model is shown to maintain or improve the latency performance while decreasing the number of false detections and unnecessary resource reallocations.

Future work in the context of this system will include methods for dynamically varying the Bayesian and Markovian thresholds in response to the current task-level resource allocation. We also plan to implement mechanisms allowing the Markovian model to suggest appropriate resource allocations for the steady-state behaviors it currently predicts. In addition, we are currently working on an 3-D tracking system that will allow us to test these decision models in the framework of an actual application.

6 Acknowledgment

The authors would like to thank Daniela Ivan Roşu for her work developing the Adaptive Resource Management framework we used to test our models, and her invaluable help in the use and application of this framework.

References

[1] R. Jha, M. Muhammad, S. Yalamanchili, D. Ivan-Rosu, C. de Castro, "Adaptive Resource Allocation for Embedded Parallel Applications," *Proceedings of the Third International Conference on High Performance Computing Systems*, 1996.

[2] D. Nicol and P. Reynolds Jr., "Optimal Dynamic Remapping of Data Parallel Computations", *IEEE Transactions on Computers*, February 1990.

[3] D. Nicol and J. Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands", *IEEE Transactions on Computers*, September 1988.

[4] D. Ivan-Rosu, K. Schwan, S. Yalamanchili, R. Jha, "On Adaptive Resource Allocation for Complex Real-Time Application," *Proceedings of the Real Time Systems Symposium*, December 1997.

[5] B. Mukherjee and K. Schwan, "Improving Performance by Use of Adaptive Objects: Experimentation with a Configurable Multiprocessor Thread Package", *Proceedings of Second International Symposium on High Performance Distributed Computing*, July 1993.

[6] B. Schroeder, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter, "From Interactive Applications to Distribute Laboratories", to appear in *IEEE Concurrency*, 1997.

[7] D. Paul, "Decision Models for On-line Adaptive Resource Management", a Masters Thesis to be presented to the faculty of the Georgia Institute of Technology in November, 1997.

[8] Y. C. Chang and K. G. Shin, "Optimal Load Sharing in Distributed Real-Time Systems," *Journal of Parallel and Distributed Computing*, pp. 38-50, 1993.

[9] D. L. Eager, E. D. Lazowska, and J. Zahorajan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*, May 1986.

[10] S.H. Bokhari. Partitioning problems in parallel, pipelined and distributed computing. *IEEE Transactions on Computers*, Jan. 1988.

[11] Honeywell, Inc. C3I parallel benchmark suite. Technical Information Report, Aug. 1996.

[12] J. Huang and W.P.-J., "On supporting mission-critical multimedia applications," *Proceedings of the 3rd IEEE International Conference on Multimedia Computing and Systems*, 1996.

[13] C. McCann and J. Zahorjan, "Processor allocation policies for message-passing parallel computers," *Proceedings of ACM Sigmetrics*, 1994.

[14] K-H. Park and L.W. Dowdy, "Dynamic partitioning of multiprocessor systems," *International Journal of Parallel Programming*, No. 2, 1989.

[15] K. Ramamritham and J.A. Stankovic, "Dynamic task scheduling in hard real-time distributed systems'," *IEEE Software*, Vol. 1, No. 3, pp. 65-75, July1984.

[16] D.I. Rosu and K. Schwan, "Improving protocol performance by dynamic control of communication resources," *Proceedings of the 2nd IEEE ICECCS*, 1996.

[17] H. Rotithor and S. Pyo, "Decentralized decision making in adaptive task sharing," *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing*, 1990.

[18] T.Bihari and K. Schwan, "A comparison of four adaptation algorithms for increasing the reliability of real-time software," *Proceedings of the Real-Time Systems Symposium*, 1988.

[19] T.Bihari and K. Schwan, "Dynamic adaptation of real-time software," *ACM Transactions on Computer Systems*, May 1991.

[20] A. Tucker and A. Gupta, "Process control and scheduling issues for multiprogrammed shared-memory multiprocessors," *12th ACM Symposium on Operating Systems Principles*, 1989.

[21] H. Zhou, K. Schwan, and I. Akyildiz, "Performance effects of information sharing in a distributed multiprocessor real-time scheduler," *Proceedings of the Real-Time Systems Symposium*, 1992.