A Performability Model for Soft Real-Time Systems

Krishna M. Kavi
Hee Yong Youn
and
Behrooz Shirazi,

University of Texas at Arlington

Abstract

In this paper we present a simple model that combines the failure to meet deadlines with hardware/software failures in computing the reliability of a real-time system. We define the performability as the probability of meeting deadlines by real-time tasks in the presence of hardware and software failures. Deadline driven schedules rely on worst-case task execution time. This may be necessary for hard real-time systems where missed deadlines can be very costly. For soft real-time systems (where a missed deadline is not catastrophic), using worst case task execution times leads to very inefficient use of processing resources. This is particularly true when worst case execution occurs very infrequently. Our performability analysis permits task schedules to slide (that is, require more time than predicted). The amount of slack allowed by the task deadlines can be varied to achieve a desired performability. We are developing a tool that empirically calculates the performability for a task system with specified task profiles and reliability system components.

Key Words: Real-Time Deadlines, Task Execution Profiles, Reliability, Software Fault-tolerance, Hardware fault-tolerance.

1. Introduction

Real-time systems are characterized as those where *the correctness of applications depends not only on the correctness of the logical computation being performed, but also on the time at which the results are produced*. The implication is that in real-time systems, failures should include hardware failures, software failures, communication failures, and failure to meet deadlines. In this paper, we propose a simple model to compute the system reliability that incorporates the inability to meet deadlines. The model is primarily motivated by our feeling that in many real-time applications the reliability goals are not clearly defined or understood. Consider for example, the following statements.

1. The survivability of the space-craft computing system should be at least 10 years.
2. The system should have a failure rate of 10^{-9} per hour during a 10 hour mission.

In the first statement, it is not clear if the system should survive the stated 10 years with all the capabilities it started with, or only some critical capabilities should survive. In the second statement, it is not clear if the system failures include missed deadlines.

Some real-time systems (often known as hard real-time systems) do not tolerate any missed deadlines. This requires that the task scheduling be based on worst-case (and accurate) task execution time estimates. This leads to inefficient utilization of resources, particularly when tasks normally require much less time to complete execution than the worst case estimates, and the worst case behavior is very rare. In soft real-time systems, where some missed deadlines can be tolerated, it is not necessary to use

worst case execution time estimates. The probability that all tasks in the system complete by a specified deadline can be increased by allowing a small amount of slack times (i.e., temporal redundancy [Siewiorek 92]). The slack time is to tolerate the failure of some tasks completing within their assigned execution times. Thus trade-off between the guarantee that all tasks meet their deadlines and the cost of such guarantees (in terms of excessive amount of processing capabilities) can be made.

Moreover, since any real-time system must also account for the failure of hardware, software and communication failures, the task deadlines must account for the possibility of reexecuting some tasks (as in active redundancy techniques [Siewiorek 92], and recovery block approaches [Kim 89]), task migrations, and delays in starting some tasks. Our model permits the inclusion of hardware, software and communication failures.

2. Performability Model

The term performability was first introduced by Meyer [Meyer 79] to combine performance and reliability analyses in fault-tolerance systems. Informally, performability can be defined as the probability that a system performs at different levels of "accomplishment". Markov processes have been used to estimate the probability that a system is in one of several "capacity" states in [Gay 79]. Chou and Abraham [Chou 80] provided an availability model for gracefully degraded systems with critically shared resources. However, these models do not include the failure to meet deadline in their computations.

In this paper we define performability of real-time systems as the probability that the set of tasks comprising the real-time system complete their execution successfully by the deadline defined with the system[1]. The failure of a task to successfully complete may be because of the following reasons.

a). For a particular instance the input data required longer execution time.
b). Due to failures, the task has to be re-executed, migrated and restarted.
c). The task is delayed (hence missed the deadline) due to the failure of a preceding task to successfully complete.

To introduce our model, let us first ignore hardware, software and communication failures, and assume a single processing system, where all the tasks comprising the real-time system must be executed in sequence. Our model will be extended to permit multiple processing elements and hardware/software failures. In this paper we will not deal with the algorithms for scheduling tasks on the available processors, or with the techniques to tolerate hardware and software failures (e.g., how tasks are migrated on processor failures, how remaining task are rescheduled on available processors, if tasks are retried when (software) failures are detected). The reader is referred to [Stankovic 88] for scheduling algorithms for real-time systems. We assume that the given task schedule is optimal and designed to meet the deadlines under normal circumstances based on estimated execution times.

Let us assume that a mission consists of a set of tasks $J= \{J_i \mid i = 1 \text{ to } n\}$. Each task is required to finish by a deadline D_i (the mission deadline is $D= D_n$, where J_n is the final task in the system). We assume that tasks take different amounts of time to complete their execution for different input data, and the actual execution time for a specific run is described by a probability distribution E_i (Figure. 1)[2].

[1]It should be noted that our definition permits the inclusion of accomplishment levels and graceful degredations.

[2]Researchers are actively developing methods for estimating execution time of tasks and designing languages that permit predictable execution times for tasks. For example, task execution times can be estimated from instruction counts, and maximum number of loop iterations. See for example [Stoyenko 92] for a survey of real-time languages. These approaches are within the scope of our model. In addition, our model permits imperfect execution time estimates. Figure 1 can be viewed as an empirical derivation of task execution times based on actual executions. Or it can be derived based on instruction counts, and various number of loop iteration counts.

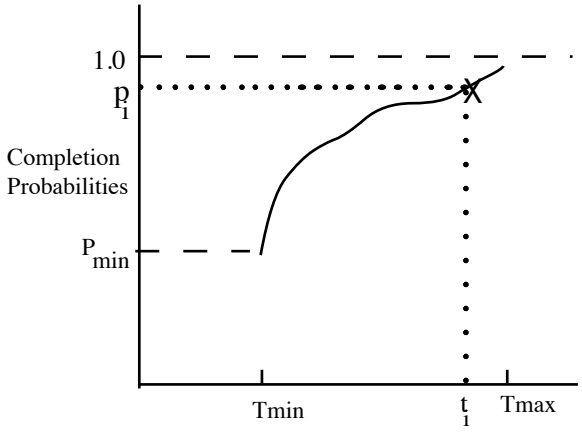

Figure . 1. Task Execution Profile

Such a task profile can be used in arriving at an execution time estimate. In hard real-time systems, one may use T_{max} as the execution time estimates. In less stringent cases, one may use the p percentile execution time (i.e., the task will complete execution with probability p). Let us denote this estimated execution time for task J_i as t_i time units. Typically, these execution time estimates are used in scheduling the tasks on available processors. Task deadlines D_i along with task execution time estimates t_i are used by the scheduling algorithm (e.g., least-laxity-first, earliest deadline first) [Stankovic 88].

Let P_{D_i} be the probability that task J_i will complete successfully during its deadline D_i. Then, the probability that the system (i.e., all tasks) will successfully complete by the system deadline D is defined as the performability of the system

$$P = \Pi P_{D_i}$$

The probability P_{D_i}[3] of a task is based on meeting deadlines. A deadline may be missed either because the task execution time exceeds the deadline, or because of a failure (either hardware, software or communications). The factor due to execution time can be obtained from the distribution E_i. Traditional derivations for modeling hardware and software failures can be used to compute their contributions to the tasks' failure to meet deadlines.

[3]Note that p_i is the probability that task J_i completes within the estimated execution time t_i, while P_{D_i} is the probability that the task completes within the deadline D_i that may permit retries, as described in section 2.1.

2.1. Significance of Active and Passive Redundancies.

A fault-tolerant computing system can be defined as a system that executes its responsibilities correctly even in the presence of defects [Siewiorek 92]. Redundancy is the primary method of achieving fault-tolerance. Redundancy can be active or passive. Triple Modular Redundancy is an example of passive redundancy, since the faults are passively masked by the majority voter. Dual redundant systems are examples of active redundancy, since faults must be detected and a corrective action must be taken. The corrective action may be in the form of reexecuting the task on a new pair of processing units. Passive redundancy (e.g., N-version programming), where the replicated units are loosely synchronized, can aid in improving the probability of meeting deadlines, since it is only necessary for a majority of the tasks to complete within the deadline (and the remaining tasks can be aborted). One such system implementation is MAX/COSMOS designed at JPL [COSMOS 93]. This, however, presents difficulties in implementation because of the lack of synchronization among replicated tasks.

Temporal redundancy where the system is permitted with extra time for repeating faulty tasks is another way of tolerating failures. However, even if the task executes successfully the second time, if the second execution is not completed by the deadline, it should be treated as a failure. A simple Bernoulli like expressions can be used to estimate the number of times a task should be retried (and hence the extra time needed for the retires). Let $q_i = (1-p_i)$ be the probability that task J_i fails to complete within the execution estimate t_i, then the probability that the task will meet the deadline, based on retries is given by

$$P_i = \sum_{j=0}^{k-1} (q_i)^j * (p_i) \text{ where } k * t_i \leq D_i$$

In other words, if the deadline permits k-1 retries, the reliability of the task is equal to the probability that it completes successfully in 1, 2, .., k attempts. Such an

evaluation is useful in deciding between redundancy and retries depending on the deadlines.

2.1.1. An Example:

Consider the following task graph (Figure 2).
The estimated execution times (i.e., t_i) are shown. With one processor, the estimated total time for the 5 tasks is 110. Consider the case when the system of jobs is allowed to complete in 120 time units (that is, 10 time units more than the combined estimated times). This specification permits one retry of tasks J_3 or J_4. The possible schedules are:

No failures; all tasks execute within their assigned times t_i .

$J_1\ J_2\ J_3\ J_4\ J_5$ Time to complete = 110

J_5 has failed and reexecuted.

$J_1\ J_2\ J_3\ J_4\ J_5\ J_5$ Time to complete = 120

J_3 has failed and reexecuted.

$J_1\ J_2\ J_3\ J_3\ J_4\ J_5$ Time to complete = 120

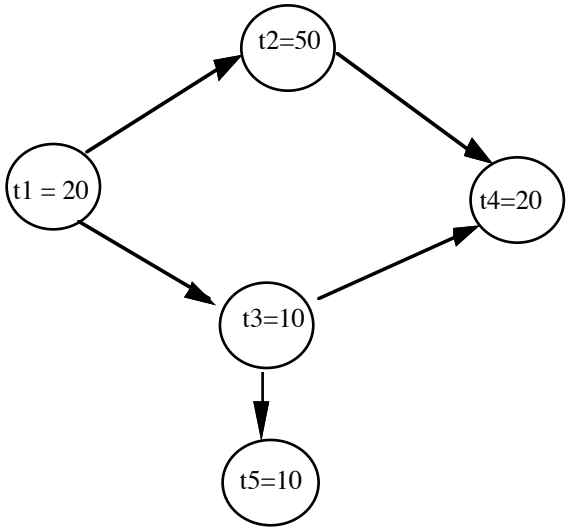

Figure 2

The performability (probability that the mission will be accomplished in 120 time units) is given by $p_1 * p_2 * [p_3 + (1-p_3)*p_3] * p_4 * [p_5 + (1-p_5)*p_5]$

Consider the same example where the deadline is extended to 130 time units. This would permit one retry each tasks J_1 and J_4 and up to 2 retries of J_3 or J_5 (or a combinations). The only task that can't be retried is the task J_2. Based on the new deadline of 130 time unit the system reliability can be computed using the following set of equations.

No failures (hence no retries):
$p_1 * p_2 * p_3 * p_4 * p_5$
One retry of task J_1 or J_4 .

$$[(1-p_1)*p_1]* p_2 * p_3 * p_4 * p_5$$
$$p_1 * p_2 * p_3 * [(1-p_4)*p_4]* p_5$$

2 retries J_3 or J_5 or 1 retry of J_3 and one retry of J_5

$$p_1 * p_2 * [(1-p_3) + (1-p_3)^2]* p_3 * p_4 * p_5$$
$$p_1 * p_2 * p_3 * p_4 [(1-p_5) + (1-p_5)^2]* P_5$$
$$p_1 * p_2 * [(1-p_3)]* p_3 * p_4 * [(1-p_5)] * p_5$$

Table 1 show how the performability is increased by allowing slack (or temporal redundancy) in the deadline. In these computations we assumed identical completion probabilities with all tasks (viz., p_i), and these probabilities are listed in the first column. Efficiency (Eff) is a measure of temporal redundancy due to the slack in the deadline.

2.1.2. Two Processors.

Consider the same example with two processors. Since task J_1 must be executed first, during its execution, we either let one processor idle or use redundant execution of task to achieve higher reliability. Then we can use one processor to execute J_2 while the other processor executes J_3 and J_5 . Since J_2 takes 50 time units, we can permit failure of J_3 and J_5 with retries (either 3 retries of J_3, 3 retries of J_5, 1 retry of J_3 and 2 retries of J_5, or 2 retries of J_3 and 1 retry of J_5). Finally, we can use redundant execution of J_4. Note that the redundant execution is either to tolerate hardware/software failures, or to overcome execution unpredictability due to the presence of pipelines, cache memories, etc. In other words, we hope that one of the two redundant executions will meet the deadline.

It takes a total of 90 time units to complete the tasks on 2 processors. Let us assume that this is also the mission deadline (no temporal redundancy is permitted). Table 1 shows the performability values for one processor case (as indicated earlier) and two processor case. The efficiency in two processor case is computed based on the idle times experienced by either of the processors.

The table shows that one processor with a small temporal redundancy of 20 time unites (i.e., deadline of 130 time units) achieves better performability than that can be achieved with two processors. As the tasks become more predictable, (i.e., the probability that a task completes execution within its estimated time approaches 1.0), the differences in the system reliability between the approaches is not significant. However, the use of two processors may have other benefits. For one thing, the mission time is reduced to 90 time units and the physical redundancy (whenever possible) can tolerate hardware and software failures.

Probability Of Task Completion p_i	No Redundancy Deadline 110 Eff = 1.0	One Processor with Slack Time (Temporal Redundancy)		Two Processors
		Deadline = 120 Eff = 0.917	Deadline = 130 Eff = 0.846	Deadline = 90 Eff = 0.611
0.9	0.5905	0.7145	0.8444	0.7346
0.95	0.7738	0.8531	0.9343	0.8958
0.99	0.9510	0.9701	0.9893	0.9704

Table 1.

2.2. Significance of Scheduling and Multi-tasking.

If tasks have completely deterministic and predictable behavior, it may be possible to obtain a static schedule that guarantees deadline in a multiprocessor system. Scheduling may or may not permit task preemption. Failures in processing resources may require alternate task schedules and task migration. When task behavior is not completely predictable, we may allow slack time in deadlines to absorb delays in task execution. In a multiprocessing and multitasking systems, since a task cannot start until all its predecessors (in terms of dependencies) have completed, the ability of a task to meet its deadline depends on that of its predecessor tasks.

This can be modeled by the following formulation.

P_{D_i} (= prob. that task J_i will meet its deadline D_i) =

(Probability that task J_i completes successfully on one try) *
(Prob. that predecessor tasks complete by $D_i - t_i$)
+ (Prob. that task J_i completes in two tries) *

(Prob. that predecessor tasks complete by D_i - $2 * t_i$)
+ .....

The above computation is applied recursively to all tasks. We may need to include communication failures and time-out protocols. When preemptive scheduling techniques are used, the performability computation needs to represent how task preemption leads to delays in task completion.

2.3. Continuous Time Model.

In the analysis so far, we have used discrete probabilities for task completion, and assumed that when a task fails to meet its deadline, the task must be re-executed. In most cases, a task may exceed its allocated time only by a small amount. In many applications, it is acceptable to let the task complete its execution (and exceed the allocated time), hoping that the slide in the schedules will be allowed by the slack in the system deadline.

To model a task slipping its deadline, we need to use a continuous probability distribution E_i for task execution profile (see Figure 1). If a task has a minimum and maximum execution times specified, the distribution will be scaled appropriately. We have observed that for a majority of real-time tasks in general, and for spacecraft tasks in particular, the task execution profile (Figure 1) can be approximated by a truncated exponential. Our model, however, is not dependent on the distribution; only, the analysis complexity is dependent on the distribution.

2.3.1. The Distribution E_i:

Typically, a task J_i is specified with a minimum and maximum execution times, $Tmin_i$ and $Tmax_i$. Depending on the behavior of the task, the actual execution time, for a given execution of task J_i can be viewed as an outcome from a probability distribution ranging between the T_{min_i} and T_{max_i}. Figure 1 shows such a behavior.

Consider two tasks J_1 and J_2 where J_1 must be completed before J_2 can be started. Assume that the deadlines for the two tasks to complete are D_1 and D_2. We can allow J_1 to execute only until D_2 - T_{min_2} ; otherwise J_2 cannot complete by D_2. The system performability can be obtained from the following convolution.

$$\int_{Tmin_1}^{\tau} E_1(\tau)d\tau \int_{\tau+Tmin_2}^{D} E_2(t)\,dt$$

$$\text{Where } \tau \leq D\text{-}Tmin_2 \text{ and}$$

$$\tau \leq Tmax_1$$

The above formulation can be extended to any sequence of tasks. When independent tasks execute in parallel (on a multiprocessing system), the above convolution becomes a simple product.

2.3.2. An Example:

Consider the same task graph (Figure 2) used in the previous example. To simplify the illustration, we will assume a uniform distribution for each task completion probabilities. Let us start with the following values. For each task, the probability that the task completes by their assigned times $t_i = p_i$. Let us use following values for T_{min_i} and T_{max_i} .

Task	T_{min_i}	T_{max_i}	t_i
J_1	15	25	20
J_2	45	55	50
J_3	5	15	10
J_4	15	25	20
J_5	5	15	10
Deadline	85	135	110

If p_i for each task (based on t_i) is 0.9, the probability that all tasks will complete by the deadline of 110

time units is given by 0.5905. On the other hand, if we extend the deadline to 120 units, and let tasks execute to completion (even if they exceed their assigned times, t_i), the probability of meeting the deadline is 0.729. This is computed as follows. All tasks are allocated at least the estimated execution times (i.e., t_i). The remaining slack time of 10 time units is given to tasks J_1 and J_2 (5 time units each). Thus these tasks are guaranteed to complete, since we allocate them their T_{max_i}. We call this a greedy method.

Performability =

$P_1(=1.0) * P_2(=1.0) * P_3 * P_4 * P_5$

$= P_3 * P_4 * P_5$

Alternatively, the 10 units of slack time can be distributed evenly among the tasks (i.e., each task can be allocated an additional 2 time units). We call this a fair-method. The following table shows the results obtained.

It should be noted that if the deadline is extended to 135 time units, then all tasks are guaranteed to complete, giving a probability of 1.0 for meeting the deadline (135 time units permits worst case execution for each task).

Probability Of Task Completion p_i	No Redundancy Deadline 110 Eff = 1.0	Greedy Method		Fair Method	
		Deadline = 120 Eff = 0.917	Deadline = 130 Eff = 0.846	Deadline = 120 Eff = 0.917	Deadline = 130 Eff = 0.846
0.9	0.5905	0.729	0.9	0.7339	0.9039
0.95	0.7738	0.8575	0.95	0.8587	0.951
0.99	0.9510	0.9703	0.99	0.9704	0.99004

Table 2

2.4. Significance of Failures.

In above examples, we have not incorporated any failures (hardware, software or communication failures). Our method permits the incorporation of failure distributions. Consider the following task profile shown in Figure 3 which is similar to Figure 1. The shaded area shows the significance of failures. The difference between the two task profile curves (with and without failures) widens near T_{max_i}, since as a task takes longer, the probability of failures increases. The probability that a task meets its deadline can now be obtained from the failure profile of Figure 3.

In a multiple processor system, hardware and software failures may require migration of tasks. The actual mechanism used to tolerate failures is beyond the scope of our model. For example, in ATAMM [Som 93], several alternate schedules (known as operating points) are precomputed for various system configurations (such as, different number of processors to account for processor failures that reduce the number of processors and repairs that

increase the number of processors). The tool is designed for periodic tasks, and a new operating point (hence a new schedule) is selected when the number of available processors changes. We plan to interface with ATAMM tool so that the performability for different number of processors and different amounts of slack in deadlines can be computed.

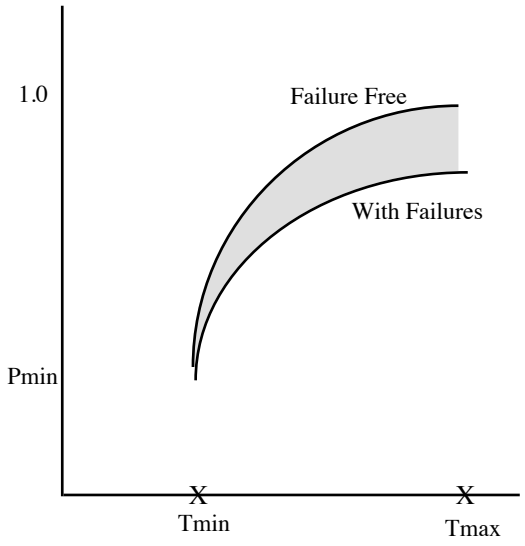

Figure 3

2.5. Application to Periodic Real-Time Tasks.

In many of real-time applications (e.g., spacecraft) tasks are repeated based on certain periodicity. In such systems, the task schedules are based on cycles, frames and subframes. A cycle consists of a fixed number frames. Each frame consists of a variable number of subframes. A subframe refers to the execution of a task. There may be different sets of tasks scheduled in different frames, but the set of tasks scheduled during a cycle is repeated. A task may be scheduled more than once in a cycle, but never more than once in a frame. If a task misses its deadline, one of the following actions are possible.

a). Preempt the task, and hope that the task will execute correctly on its next schedule. Dependent tasks will use data output from a previous execution. Here the trade-off is between accuracy and timeliness of the mission.

b). Permit the task to continue, and hope that the set of tasks constituting a mission will still meet the deadline. This is often possible, since the mission contains some low priority tasks which can be delayed (or not scheduled).

Our approach permits both techniques for dealing with missed deadlines. Since the mission consists of a fixed number of cycles, the performability of the mission should be based on the overall mission time.

2.6. A Performability Prediction Tool.

It is possible to apply traditional stochastic processes to obtain an analytical solution to the performability of a soft real-time system that incorporates hardware/software failures and the failure to meet deadline. We, however, feel that it will be useful to a real-time systems designer to provide a tool that obtains empirical data by simulating the various task execution times, and injects hardware or software failures. Such a tool can be used to determine the temporal redundancy needed to achieve a desired performability. We have completed a simple version of such a tool. At present we permit uniprocessor systems. We are in the process of extending the tool to multiprocessor systems (both shared memory and message-passing).

When the tool is complete, we envision the following environment. The input consists of a task graph (depicting task dependencies), an execution profile for each task, target architecture (i.e., number of processors, interconnection, shared vs distributed memory), schedule of tasks on the target architecture (described in the form of a Gant chart), mission deadlines, communication delays (if any), failure distributions (for hardware, software and communication subsystems). The user can utilize graphical interfaces (e.g., X or Motif) to specify the input and/or choose from a library of pre-determined configurations. Figure 4 shows a typical window. The user can select from a variety of fault-tolerance mechanisms and scheduling methods to deal with failures. The user can select from a variety of probability distributions for the task profiles and failures.

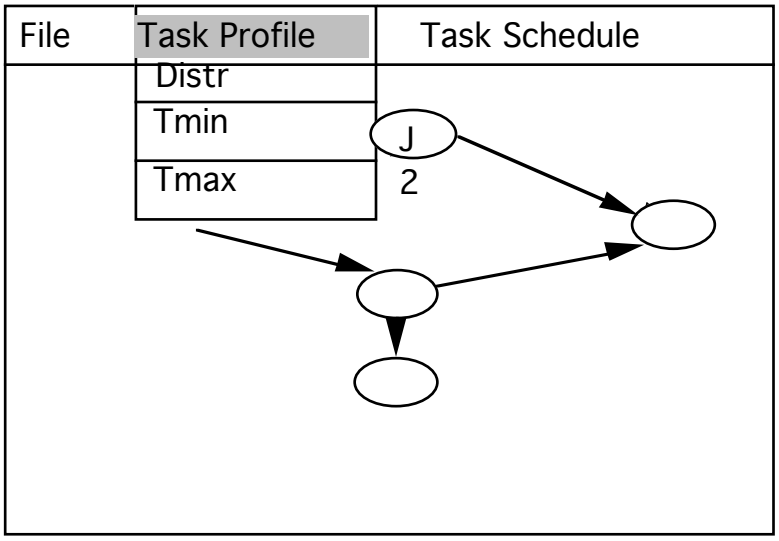

Figure 4.

The system will generate performability predictions empirically. Using event driven simulations, the performability predictions for various amounts of slack times (temporal redundancy) will be displayed. At user option, other statistics such as processor utilization will be provided. The output can be displayed as bar charts, graphs or tables.

At present, the tool has no graphical interface. The input is fed from another tool that calculates task schedules based on a variety of heuristics ([Shirazi 93]). The task graph (and the task schedules) are described as an extended adjacency list. The outputs display the performability of the system for various amounts of slack time in the deadline.

3. Summary and Conclusions

In this paper we have described a simple model that can be used to evaluate the probability that a real-time system meets its deadline in the presence of failures and imperfect execution time estimates (viz., performability). Our model can be used to estimate the time-redundancy (slack time) needed in the overall mission deadline in order to achieve a desired performability. The analysis can also be used to evaluate the trade-off between physical redundancy (such as Triple-Modular-Redundancy and N-Version Programming) and active redundancy (such as roll-back and recovery, re-try) - that is, a trade-off between performance and reliability. As failures occur, the ability of the system to support the critical tasks (and meet deadlines) can also be evaluated (viz., graceful degradation) from our model.

We are currently developing a user-friendly tool that uses even-driven simulation to empirically derive the performability of real-time systems consisting of concurrent tasks executing on multiple processors. When fully developed the tool will permit all of the analyses mentioned in the previous paragraph. Our tool relies on users to obtain optimal schedules, and estimates of task execution times (i.e., task execution profiles).

4. References

[Chou 80] T.C.K. Chou and J.A. Abraham. "Performance/Availability model of shared resource mulitprocessors", IEEE Transactions on Reliability, R-29, April, pp. 70-74.

[COSMOS 93] B.F. Lewis et. al. "COSMOS Multicomputer operating system and development environment: Functional Specification", NASA-JPL Memo.

[Gay 79] F.A. Gay and M.L. Ketelsen. "Performance evaluation for gracefully degrading systems", Digest of 9th International Symposium on Fault-Tolerant Computing, Madison, WI, pp. 51-58.

[Kim 89] K.H. Kim and S.M. Yang. "Performance impacts of lookahead execution in the

conversation scheme", IEEE Transactions on Computers, Aug. 1989, pp 1188-1202.

[Meyer 79] J.F. Meyer, D.G. Furchgott and L.T. Wu. "Performability evaluation of the SIFT Computer", IEEE Transactions on Computers, C-29, pp. 501-509.

[Shirazi 93] B. Shirazi, K.M. Kavi, A.R. Hurson and P. Biswas. "PARSA: A parallel program scheduling and assessment environment", Proc of 1993 International Conference on Parallel Processing, August 16-20, 1993

[Som 93] S. Som, R.R. Mielke and J.W. Stoughton. "Prediction of performance and processor requirements in real-time dataflow architectures", to appear in IEEE Transactions on Parallel and Distributed Systems.

[Siewiorek 92] D.P. Siewiorek and R.S. Swarz. Reliable Computer Systems: Design and Evaluation, 2nd Edition, Digital Press, Bedford, MA.

[Stankovic 88] J.A. Stankovic and K. Ramamritham. Tutorial on Hard Real-Time Systems, IEEE CS Press.

[Stoyenko 92] A.D. Stoyenko. "The Evolution and State-of-the-Art of Real-Time Languages", The Journal of Systems and Software, April , pp. 61-84

[Trivedi 82] K.S. Trivedi. Probability & Statistics with Reliability, Queuing, and Computer Science Applications, Prentice-Hall, Englewood Cliffs, NJ.