Scheduling of Soft Real-Time Systems for Context-Aware Applications

Jennifer L. Wong, Weiping Liao, Fei Li, Lei He, Miodrag Potkonjak

Scheduling of Soft Real-time Systems for Context-aware Applications

Jennifer L. Wong[‡], Weiping Liao[†], Fei Li[†], Lei He[†], Miodrag Potkonjak[‡]
[‡] Computer Science Department, University of California, Los Angeles
[†] Electrical Engineering Department, University of California, Los Angeles

Abstract

Context-aware applications pose new challenges, including a need for new computational models, uncertainty management, and efficient optimization under uncertainty. Uncertainty can arise at two levels: multiple and single tasks. When a mobile user changes environments, the context changes resulting in the possibility of the user requesting tasks which are specific for the new environment. However, as the user moves these requested tasks may no longer be context relevant. Additionally, the runtime of each task is often highly dependent on the input data.

We introduce a hierarchical multi-resolution statistical task model that captures relevant aspects at the task and intertask levels, and captures not only uncertainty, but also introduces the notion of utility for the user. We have developed a system of non-parametric statistical techniques for modeling the runtime of a specific task. This model is a framework where we define problems of design and optimization of statistical soft real-time systems (SSRTS). The main algorithmic novelty is a cumulative potential-based task scheduling heuristic for maximizing utility. The heuristic conducts global optimization and induces low runtime overhead. We demonstrate the effectiveness of the scheduling heuristic using a Trimaran-based evaluation platform.

1. Introduction

Application domains often imply a need for unique computational models and unique optimization goals. Recently, a new domain of context-sensitive mobile wireless applications emerged. Context can be defined as the set of environmental states and settings that either determine an application's behavior or in which an application event occurs and is interesting to the user [1]. Context-sensitive applications include mobile applications, sensor networks, pervasive and ubiquitous computing, Internet browsing, video games, and virtual reality. In all these applications, the next set of computational tasks which might be executed statistically depend on the current context of the user and the current executed tasks. Additionally, there is intrinsic uncertainty about which context direction the user will follow.

We have developed a new computational model, statistical soft real-time systems (SSRTS). We represent the workload of a user as a system of tasks with execution dependencies. At each moment of time, the scheduler has information about the tasks in the current context of the user and statistical information about the likelihood of future pending tasks to be executed by the user. For all tasks, we have statistical information about the likelihood that they will be completed in a given amount of time on a given processor. Therefore, SSRTS considers uncertainty at two levels: the execution time for a single task and the user invoked subset of tasks. What is unique for SSRTS is that uncertainties are addressed using statistical techniques.

The SSRTS model is related to both hard and soft real time systems. The main difference from hard real-time systems and the SSRTS model is that the only a subset of the tasks needs to be executed and therefore their exists uncertainty. In comparison to classical soft real time systems, SSRTS differs in the presence of statistical information about tasks and requirements which one has to complete the task in order to achieve a measure of benefit. Note that one can interpret traditional soft real-time systems (SRTS) as a special case of SSRTS, by partitioning each task in the SRTS into a series of smaller tasks.

2. Related Work

In this section, we briefly survey the related work in several directly related fields. The pervasive computing community and many others have contributed a number of notable efforts in this area, including [13]. Models of computation have been recognized as the first building block for the development of formally sound systems [3]. Shin et al. discuss the task-level computational models for real-time applications [11]. Scheduling of hard, soft, and a combination of hard and soft real-time systems has been also addressed in CAD literature [10]. While hard real-time systems [9] ask for an absolute guarantee that all deadlines are satisfied, there is no such requirement for soft real-time systems. Soft real-time systems has been mainly motivated by applications from two domains: databases [5], and multimedia [4]. An important characteristic of real-time systems is the timing constraints of each task. Determining an esti-

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE

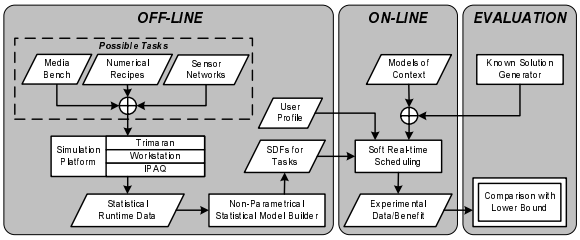

Figure 1. Global Flow for Scheduling SSRTS.

mate of the worst case execution time (WCET) for tasks is often crucial for making judgments concerning timing properties in real-time systems [7]. We use non-parametric statistical techniques for the development of the task model due to their application to data which has arbitrary distributions and without any assumptions on the data [12].

3. Preliminaries and Global Flow

A **task** is a single program which can be invoked and executed. For example, a task may be a single execution of a jpeg decoder, audio decompression, or audio player. Each task has a set of attributes including its name, type, utility (benefit), arrival time, deadline, and a statistical distribution function that characterizes its execution time. The **utility** of the task is the benefit gained by the user for complete execution of the task. The utility or benefit is proportional to the value of the programs that are executed within time intervals specified by the user. The point in time at which the task must be completed in order to receive benefit for the task is the **deadline**. The utility and the deadline can be statistically determined based on the users previous behavior i.e. according to their profile. The **statistical distribution function (SDF)** for execution time provides information about the statistical likelihood of specific runtimes for a task. The runtime of the task is plotted versus the likelihood of the task completing execution at each runtime. Between two tasks there can be two types of relationships: dependency and weighted likelihood. If the results of one task is needed for the execution of another task, we specify this relationship as a **dependency**. A **weighted likelihood** relationship reflects the user's likelihood to request the execution of a task after the completion of another task.

The system, shown in Figure 1 consists of three stages: off-line, on-line, and evaluation. The goal of the off-line stage is to statistically characterize each of the tasks in terms of their runtime and to determine the user profiles. In the on-line stage, the information learned in the off-line stage is used to guide the real-time scheduler. Lastly, the evaluation stage evaluates and validates the performance of the scheduler against lower bound estimates. The off-line stage starts with an analysis of the possible tasks that the user can invoke. Analysis of each of the tasks on different simulation platforms is performed in order to gather statistical runtime data for each of the tasks. Non-parametric statistical techniques are use to model the runtime of each task. The resulting output is the statistical distribution function (SDF) of execution times for each task. We discuss the technical details concerning this process in Section 4. Additionally, in this stage, information from the user concerning the user's task preferences is gathered.

Once the user profile and the task profiles have been established, the user can start to use the system. As the user changes contexts (e.g. moves from a particular place to a new place), different tasks can be invoked. The possible tasks in the user's context are contained in an instance of the task model which we discuss in Section 4, i.e. window of consideration. It is often beneficial to invoke a subset of pending tasks in order to maximize the overall benefit. The real-time scheduler schedules the possible tasks with the goal of providing the user with the maximal amount of benefit. When the user invokes a task, one of two situations may occur. In the first situation, the scheduler had pre-fetched the invoked task and the data is available. In this case, the scheduler has provided benefit to the user. However, if the scheduler had not pre-fetched the task, the task may still be finished by the tasks deadline. If so, and the task has exceeded its deadline the task finishes execution prior to the deadline placed according to the user's profile, benefit is gained. Otherwise, the task is dropped from consideration and the user gains no benefit. The purpose of the final stage, evaluation, is to determine the effectiveness of the modeling software and scheduling heuristic. Evaluation is done through a comparison with lower bound estimates.

4. Statistical Model of Task

The purpose of the task model is to provide a compact and accurate representation of soft real-time tasks with respect to their execution time. The task model has three hierarchical layers: window of consideration, intertask, and intratask. At the top is the window of consideration, which determines the current prospective of the model. The middle layer, intertask, encompasses the interaction between different tasks. Finally, the lowest level models each task in detail. Figure 2 illustrates these layers.

The intratask layer defines a single task according to its process requirements and statistical information. Each task is represented by four components: name or type of task, utility, deadline, and a statistical distribution function (SDF) for the task's runtime. The utility of the task measures the benefit that is gained by the application or user if the task is completed. The utility can be defined by the user in terms of application preferences, or can be determined based on the profile of the user. The profile can be statistical information gathered from the user's hard real-time requests or feedback from the user. The deadline of the task can be represented as absolute time or relative to the arrival time of

the task. The deadline is based on the requirements placed by the user on the latency of the task. There is intrinsic uncertainty associated with the runtime of the task. In order to represent this uncertainty, a statistical distribution function (SDF) for the runtime of the task is used. The statistics for creation of the SDF can be gathered through off-line evaluation of the task under various conditions, as shown in the off-line stage of Figure 1.

In order to create a runtime model, in the form of an SDF, for each task, we use the non-parametrical statistical resubstitution technique and a greedy provably optimal algorithm. Before we discuss the iterative creation process for an SDF, we first introduce the notion of a cumulative statistical distribution function (CSDF) and a measure for differences between CSDFs. The CSDF is the likelihood that the runtime of task is less than or equal to the current time unit. The cumulative likelihood at time unit x of the CSDF as the $\sum_{i=0}^{x} SDF(i)$ where $SDF(i)$ is the value of the SDF at time i. The difference between two CDFs can be calculated by comparing the difference between the likelihoods at each time unit. We define the difference between a CSDF S and k other CSDFs as $\sum_{i=0}^{m}\sum_{j=0}^{k} |S(i) - CSDF_j(i)|$ where m is the length of the longest CSDF. This definition naturally captures the impact of differences on applications.

When creating a SDF to capture the statistical information about runtime of a task, the goal is to create an SDF which "best" represents the likelihood of the task's runtime. We define "best" as the SDF which has the smallest CSDF difference between itself and all other SDFs. The process of creating a representative SDF begins with the runtime data for each task which is collected from the simulation platforms. The initial collected data should be a representative set. Resubstitution is applied to the data for each task to form a large number of subset SDFs, at least a thousand subsets to be statistically sound. Each of the subset SDFs are then converted into CSDFs. A representative CSDF is first created that will later be converted to a representative SDF for the task. The representative CSDF is created by considering the values of each subset CSDF for each time unit. If there are an odd number of CSDF subsets, the median value is selected as the representative value for that time slot. In the case of an even number, the midpoint value between the two median defining values is selected. The selected values for each time slot are optimal. This is due to the fact that if any value selected which is between the CSDF value to the right or the left of the median value, then more than half of the other CSDF values will be at an increased distance from the representative value. Therefore, no other CSDF assignments for each value will result in a smaller difference.

The main purpose of the intertask layer is to represent the relationships between tasks. We introduce two types of directed relationships between tasks: dependencies and likelihood. Dependency relationships imply that data from the initial task is needed in order for the second task to perform its function. For example, if the user requested a sound file to be played, the file must first be decompressed, then it can be played. In Figure 2 we represent dependency edges using solid arrows. The weighted likelihood edges, drawn as dotted arrows, represent the likelihood of the second task to be requested by the user after the execution of the initial task. Note that the resultant intertask model forms a directed acyclic graph (DAG).

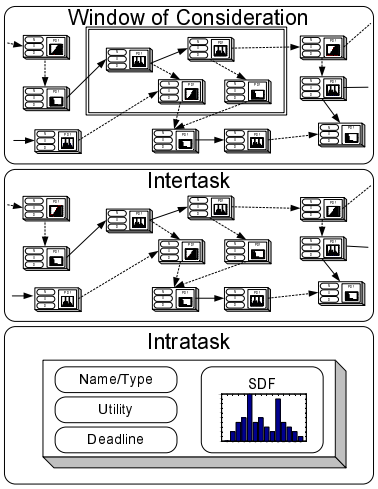

Figure 2. Task Model hierarchy: Window of Consideration, Intertask, Intratask levels.

The highest layer in the hierarchy is the window of consideration. The window of consideration determines the portion of the model which is currently considered according to the user's context. The size of this window is dependent on the amount of available memory, the processing utility, and the requirements of the application.

5. Problem Formulation

In this section, we introduce the benefit maximization under uncertainty scheduling optimization problem. We also establish the computational complexity of the problem. For a context-sensitive computing application, the user has the opportunity to execute a variety of tasks dependent upon the users current context. Each task has uncertainty in terms of its execution time. It is also uncertain which of the tasks will be eventually request for execution. The goal is to pre-fetch, schedule, and execute tasks to achieve the highest overall utility.

A single directed acyclic graph, such as the subgraph defined by the window of consideration presented in the task model, is composed of task vertices and dependencies (partial order) and weighted likelihood edges. Each task in the graph corresponds to a single task to possibly be scheduled,

and is composed of the four components as shown in Figure 2: name or type of task, utility of the task, deadline, and SDF. The goal is to maximize the total benefit achieved by scheduling tasks and completing each of these tasks before their deadline under the condition that benefit is only received if the task was requested (subset of tasks S).

We proved that the Maximum Benefit under Uncertainty Scheduling problem is computationally intractable since it is a special case of the Sequencing with Release Times and Deadlines Problem [2] which is NP-complete. This special case occurs when the benefit for each task is 1, all tasks have a probability of execution equal to 1, the SDF of each task is replaced with a known static execution time, and all dependency and likelihood edges are removed.

6. Scheduling

The goal of the scheduling approach is to determine which tasks from the current window of consideration, or current context of the user, should be scheduled based on the user's likelihood to request the task to be executed and the benefit the user receives if the task is completed. The intuition behind the heuristic is to select tasks according to the following three criteria: (i) high likelihood of execution (likelihood edges), (ii) high user benefit, and (iii) SDF which show high likelihood to finish the task within its deadline. We assume that at periodic intervals of time, the current context of the user is provided to the algorithm, i.e. the window of consideration is refreshed. In our experimentation, the window of consideration is always two period lengths. The second assumption is that only single dependencies exist between two tasks, meaning a task can only be dependent on the output of one task. Note that this condition is often true in context-sensitive applications and in other situations one can reduce more complex situations to this one by introducing dummy tasks with runtime and benefit equal to zero. Using the three stated criteria, we have developed the following heuristic approach. For each period, we consider all tasks which have arrived and have not exceeded their deadline. All tasks which have dependency edges are merged into a single representative task, which we discuss later in this section. For each task, we consider its maximum potential and average potential according to the above mentioned three criteria. According to these two measures, we select the order of tasks to be executed in this period. If the task which is being executed exceeds it's deadline, the task is dropped and the next task executed. A special condition exists if the task is part of the dependency tree. In this case, if the task has exceeded its deadline, it should not necessarily be dropped due to the fact that its output maybe necessary to execute other task(s). In this case, we calculate the tasks maximum potential to finish within k additional units of time using the task's SDF. At the end of the period, each task which has not exceeded its deadline is kept to consider in the next period.

Algorithm:
1. Let $b(t)$ be the benefit value for finishing task t;
2. Let $SDF(t)$ be the probability distribution function for task t;
3. Let $D(t)$ be the deadline for task t;
4. Let $p(t)$ be the probability of executing task t, i.e. sum of all incoming probabilities;
5. Let R be the set of tasks that arrived before the current period;
6. Let $MaxB$ be the maximum benefit density for task t;
7. Let $AvgB$ be the average benefit density for task t; but have not been finished;
8. Let $W(t)$ be the weight of task t in our scheduling;
9. $R = \phi$;
10. Foreach period P_i\{
11. Let $S = R \cup T_i$;
12. Substitue the tasks in one dependency tree with a single task by combining SDF of tasks in the dependency tree;
13. Foreach task $t \in S$\{
14. Calculate $MaxB(t, SDF(t), D(t), p(t))$;
15. Calculate $AvgB(t, SDF(t), D(t), p(t))$;
16. Calculate $MLEB(t, SDF(t), D(t), p(t))$;
17. if(ave runtime/period length $> .5$)
18. $W(t) = MaxB + MLEB$;
19. else
20. $W(t) = AveB + MLEB$; \}
21. Sort the tasks in S in non-increasing order by $W(t)$;
22. Schedule the tasks in S according to $W(t)$ until period P_i is finished;
23. Drop task t in S only if it has passed its deadline $D(t)$ and it is not a combined task from a dependency tree;
24. Truncate and re-scale the $SDF(t)$ if task t in S is executed but not finished;
25. $R = S$; $S = \phi$; \}

Figure 3. Pseudo-code for scheduling.

The pseudo-code for the approach is presented in Figure 3. Lines 1-9 define the notation used to represent the task model. The processing for each period begins on line 10. Essentially, at the beginning of period i, T_i tasks arrive and are considered in conjunction with the tasks from the previous periods which have not passed their deadline. If there are any dependencies between the tasks, we combine then into a single tasks. Since only single dependencies exist, either a dependency chain or tree can be constructed. In the case of a tree, we simplify the situation, in order to make the algorithm efficient by converting the tree into a chain. It is done by performing depth-first search on the tree and placing the tasks in a chain according to order.

In order to represent the chain of dependency tasks as a single "combination" task, we must combine the SDFs to form a representative SDF. The method used to construct the combined SPDF is, for any given amount of time, we try all possible time allocation schemes to individual tasks in the dependency chain and use the best finishing probability, ie. achievable benefit, as the value for the combined SDF at that given time. For any given amount of time, there is an exponential number of possible allocation schemes for a dependency chain. We use a heuristic algorithm to create the combined SDF. First, for each task's SDF, we allocate time to that task such that the task may achieve the largest finishing probability in its SDF. Next, we sum all the best allocated times for all the tasks in the chain resulting in the total

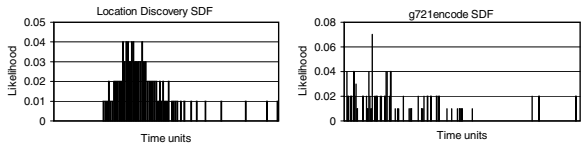

Figure 4. SDFs of Benchmarks: location discovery and g721encode tasks.

allocated time A for the dependency chain. To construct the combined SDF, we use a dynamic programming method. Suppose the given time for the combined SDF is $A-1$, we then examine the sensitivity of each task's PDF, i.e., if the allocated time to that task is reduced by one, how much benefit will be lost. We choose the task with the lowest sensitivity and reduce the time allocated to that task by one. This is the value of the combined SDF for $A-1$. Through this method, we can re-construct the combined SDF (combined benefit delivery function) by computing in both directions from A.

Once all tasks have been represented as independent tasks, we determine the suitability of each task for scheduling. The suitability is the task's potential according to the three criteria. Measures can be defined to summarize the potential benefit of a task. For example, we can define two measures: MaxB and AveB. Let $l(x)$ be the likelihood of the SDF at time unit x. We define MaxB as the maximum($\frac{(f(x)\cdot b(t)\cdot p(t))}{x}$), over all x in a domain where $l(x)$ is defined. This measure essentially indicates the maximal potential benefit for a task normalized with respect to consumed time. The higher the MaxB, the more benefit this task delivers in the same amount of execution time in the best case scenario. The average expected benefit for a task with a given SDF is denoted by AvgB and defined as $\frac{\Sigma(f(x)\cdot b(t)\cdot p(t))}{length(SDF)}$. Obviously, neither the maximum nor the average metric alone is enough to guide the scheduling approach, because both abstract only some aspects of each SDF. Following, the maximum likelihood principle, for summarizing the SDF information, we use most likely expected benefit, MLEB, defined as $\sum \frac{(f(x)\cdot b(t)\cdot p(t))}{x}$. In order to prevent often selection of tasks with benefit variance too high or too low, we combine the MLEB with two other measures. If the ratio of the average runtime to the available time is low, we use MLEB and AveB and if high, we use MLEB and MaxB. The weight factor for MLEB is proportional to the square of the inverse of the ratio. The resulting weight is assigned to each task, and the tasks are sorted according to the highest weight value (line 21). Each task is scheduled in this order until the end of the period. If at any point in time a task exceeds it's deadline, the task is dropped, unless it is part of a dependency tree. For any task which has begun execution, and has not finished or passed its deadline, we shorten the task's SDF and rescale the SDF according to the amount of processing already completed. The tasks which still remain as possible executable tasks, are passed on to the next period, and the process is repeated.

7. Experimental Results

In this section, we present experimental evaluation of the developed techniques and algorithms. We have two goals. First, to demonstrate the relevancy and accuracy of the SDF in the task model. The second is to demonstrate the effectiveness of the benefit under optimization scheduling heuristic. In order to accomplish these goals, we compare the results with a lower bound which is also presented.

In order to evaluate the effectiveness of our approach, we introduce a lower bound technique. The approach is based on a relaxation of the benefits under uncertainty problem which replaces SDFs with static execution times, dependencies between tasks are removed, and preemption is allowed. For each fraction of a task that is executed, partial benefit is included into the overall benefit. The lower bound algorithm is applied on a period-by-period basis. For each time period, the algorithm runs the task with the highest benefit to runtime ratio, and preempts the task when a task with a higher ratio arrives or the task's deadline is reached.

In order to establish the effectiveness of both the SDF and the new approach for scheduling, we selected a variety of multimedia, statistical tasks, and sensor network tasks. The multimedia tasks (e.g. g721encode, gsmencode) were taken from the MediaBench test suite [6]. For statistical and computation tasks, we used programs (xfit, ksone, twofft) from the Numerical Recipes software [8]. The sensor network tasks, location discovery and exposure were used. SDFs from two benchmarks are shown in Figure 4.

In order to demonstrate the relevance and effectiveness of the SDFs as a measure of runtime over the average or worst case expected runtime in the model, we experimented using the heuristic algorithm. Fifty different real-time instances were created using the task model with the SDF, with the SDF replaced by average runtime, and with the SDF replaced by worst case runtime. Additionally, the instances varied in period length and amount of dependencies between tasks. Each of the instances were tested using the lower bound and heuristic approach. Table 1 provides the statistics for all of the examples. The use of the SDF in the task model proved to be more effective in all cases. The accuracy of the SDF to represent the expected runtime of a task was determined using the resubstitution and confidence calculation procedure presented in Section 4. For each of the SDFs created for the benchmark tasks, we validated using a 90% confidence interval of the SDF values at four likelihood intervals, 90%, 60%, 30%, and 10%. For a subset of the instances we present the true maximum trial runtime (in cycles) for each benchmark and the true confi-

	SDF	Ave.	Worst Case
MinB	33%	5%	9%
MaxB	71%	57%	55%
AveB	53%	39%	37%
Var	0.010	0.025	0.0242

	Max Value	10%	30%	60%	90%
g721encode Normalized	2.63E7	$(3.6\text{-}11.4)\cdot10^5$ 0.01-0.04	$(2.4\text{-}3.3)\cdot10^6$ 0.09-0.13	$(4.5\text{-}6.2)\cdot10^6$ 0.17-0.24	$(1.0\text{-}1.5)\cdot10^7$ 0.40-0.59
gsmencode Normalized	3.81E8	$(.90\text{-}1.4)\cdot10^6$ 0.02-0.04	$(2.2\text{-}2.9)\cdot10^7$ 0.06-0.08	$(3.8\text{-}5.1)\cdot10^7$ 0.10-0.14	$(1.5\text{-}1.9)\cdot10^8$ 0.40-0.51
xfit Normalized	2300	$(1\text{-}1)\cdot10^3$ 0.0478-0.0478	$(1\text{-}1)\cdot10^3$ 0.0478-0.0478	$(1\text{-}1)\cdot10^3$ 0.0478-0.0478	$(1\text{-}1)\cdot10^3$ 0.0478-0.0478
Loc. Discovery Normalized	46200	$(1.1\text{-}1.3)\cdot10^4$ 0.025-0.3	$(1.3\text{-}1.7)\cdot10^4$ 0.3-0.37	$(1.7\text{-}2.0)\cdot10^4$ 0.37-0.46	$(2.2\text{-}2.9)\cdot10^4$ 0.48-0.65

Table 1. (Left) Normalized benefit statistics for different runtime mechanisms used in task model. (Right) Confidence Intervals for SDFs of benchmarks.

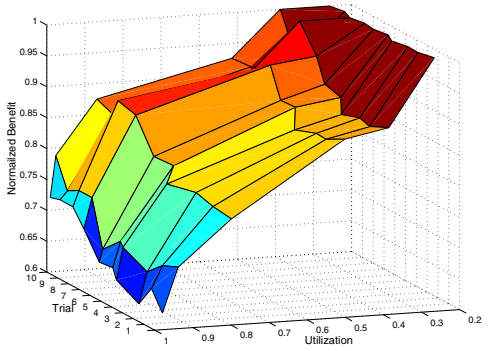

Figure 5. Algorithm benefit normalized vs. benefit of lower bound for varying utility.

dence interval for each interval in Table 1. Additionally, we present the normalized confidence range.

We now present analysis of our heuristic scheduling approach. Comparison is performed against the lower bound. Additionally, we test the performance of the approach with varying processing utilization. The purpose of the utilization is to determine the effectiveness of the approach with respect to systems with varying resource limitations, such as laptops or portable devices such as PDAs. Figures 5 presents the performance results for twenty different scheduling instances, shown on the y-axis. The benefit achieved by the heuristic is normalized against the lower bound benefit (z-axis). The third dimension is the processor utilization factor. In Figure 5 the period length is 300 units. Similar results were obtained for a period of 200 units. In both cases with 100% utilization of the processor, we are able to achieve between 50-70% of the lower bound. With between 30-40% utilization, we can process all tasks.

8. Conclusion

We have developed a multi-resolution statistical soft real-time task model of computations that addresses the needs of context aware applications. We also have implemented a system of non-parametric statistical techniques for modeling the runtime of a specific task. Furthermore, we have proposed and tested a new cumulative potential-based task scheduling heuristic for maximizing utility.

References

[1] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[2] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[3] A. Jantsch. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2003.

[4] M. B. Jones, D. Rosu, and M. Rosu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Symposium on Operating Systems Principles*, pages 198–211, 1997.

[5] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. *IEEE Transactions on Parallel and Distributed Systems*, 8(12):1268–1274, 1997.

[6] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[7] Y. T. Li, S. Malik, and A. Wolfe. Efficient microarchitecture modeling and path analysis for real-time software. In *IEEE Real-Time Systems Symposium*, pages 298–307, 1995.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C : the art of scientific computing*. Cambridge University Press, 1997.

[9] K. Ramamritham and J. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, 1994.

[10] K. Richter and et al. Model composition for scheduling analysis in platform design. In *Design Automation Conference*, pages 287–292, 2002.

[11] K. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. In *Proceedings of the IEEE*, volume 82, pages 6–24, 1994.

[12] R. Thisted. *Elements of statistical computing*. Chapman and Hall, 1988.

[13] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.