SETS: Stochastic Execution Time Scheduling for Multicore Systems by Joint State Space and Monte Carlo

Nabeel Iqbal, Jörg Henkel
Karlsruhe Institute of Technology (KIT), Chair for Embedded Systems, Germany
{nabeel.iqbal, henkel} @kit.edu

Abstract— The advent of multicore platforms has renewed the interest in scheduling techniques for real-time systems. Historically, 'scheduling decisions' are implemented considering fixed task execution times, as for the case of Worst Case Execution Time (WCET). The limitations of scheduling considering WCET manifest in terms of under-utilization of resources for large application classes. In the realm of multicore systems, the notion of WCET is hardly meaningful due to the large set of factors influencing it. Within soft real-time systems, a more realistic modeling approach would be to consider tasks featuring varying execution times (i.e. stochastic). This paper addresses the problem of stochastic task execution time scheduling that is agnostic to statistical properties of the execution time. Our proposed method is orthogonal to any number of linear acyclic task graphs and their underlying architecture. The joint estimation of execution time and the associated parameters, relying on the interdependence of parallel tasks, help build a 'nonlinear Non-Gaussian state space' model. To obtain nearly Bayesian estimates, irrespective of the execution time characteristics, a recursive solution of the state space model is found by means of the Monte Carlo method. The recursive solution reduces the computational and memory overhead and adapts statistical properties of execution times at run time. Finally, the variable laxity EDF scheduler schedules the tasks considering the predicted execution times. We show that variable execution time scheduling improves the utilization of resources and ensures the quality of service.

Our proposed new solution does not require any a priori knowledge of any kind and eliminates the fundamental constraints associated with the estimation of execution times. Results clearly show the advantage of the proposed method as it achieves 76% better task utilization, 68% more task scheduling and deadline miss reduction by 53% compared to current state-of-the-art methods.

Keywords: Scheduling, stochastic execution time, parallel tasks, state space modeling, joint estimation, Monte Carlo, task utilization

I. Introduction

A paradigm shift in computing has appeared with the emergence of multicore systems, driven by an easy availability of inexpensive, high-performance processors, while at the same time offering a prospective solution for overcoming the power constraints and the thermal issues associated with the processors. The concerns of programming these platforms are urgent as no automatic solution is likely [2]. This tends to draw greater research efforts toward parallel computing methods, run-time management of parallelism and related research fields. In the realm of real-time systems, the programming of such multiprocessor systems presents a rather formidable problem. In particular, time-critical tasks must be serviced within certain pre-determined deadlines, dictated by the required quality of service. The most important attribute of real-time systems is that the correctness of such systems depends not only on the computed results but also on the time at which the results are produced (timing guarantee). Scheduling and schedulability analysis is a mechanism to provide these guarantees. The multicore systems pose new challenges while opening up new design opportunities for scheduling. Therefore, it is essential to examine the problem of scheduling in the context of multicore environment to devise efficient methods and tools for the scheduling of tasks.

In general, a task can be characterized by an integer tuple (S_t, C_t, D_t), where S_t is the start-time of the task, C_t is the execution time and D_t is the deadline. The most non-trivial entity in the context of scheduling is C_t as remaining variables can be known exactly for the given scheduling interval. C_t, by and large, is the property of the run-time and depends on a large number of factors, ranging from the task program structure to the run-time state of the underlying hardware platform. Most of the mapping and scheduling algorithms assume that task execution times or their bounds are known quantities. Because of the several factors affecting it, it is rather unfair to consider it a known quantity. The execution time is unknown before the task is mapped on the architecture, but even afterwards, the execution time remains stochastic due to application-dependent, platform-dependent, and environment-dependent factors. The variation in the amount and type of data input to the application, the micro-architecture of the processing units (which influences caching, queuing etc.), interaction with the environment, database accesses, communication links etc. introduce uncertainty in the execution time of the task. The multicore/multithreaded systems give rise to additional scenarios such as the interdependence of threads mapped on different cores (Software pipelining). In the scenario of real-time systems where timelines are essential, the advances in multicore system become part of the problem rather than the solution. Futuristic computer architectures and software have made it difficult or impossible to estimate the execution time of the task at design time [1].

In simple words, if tuple (St, Ct, Dt), is known, then scheduling boils down to the decision by classical scheduling algorithms such as Earliest Dead Line First (EDF). Because of the influence

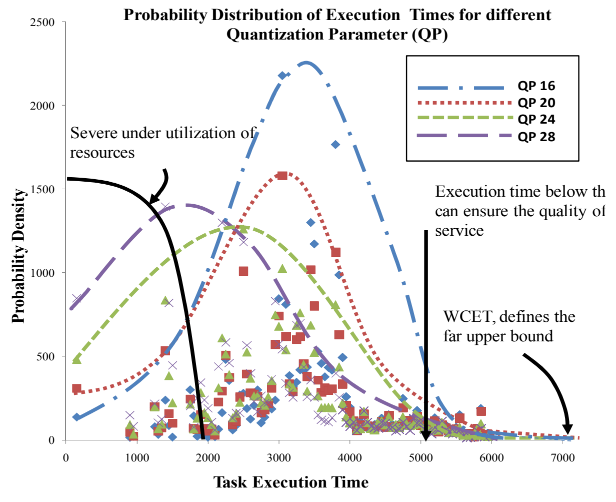


Figure 1: WCET and stochastic task execution time

of a large number of factors at run-time, it is essential to treat the problem of task execution time estimation in stochastic terms [4]. Historically, the Worst Case Execution Time (WCET) estimation has been investigated in depth by researchers. WCET defines the upper bound. It leads to severe underutilization of the resources. Moreover, in the realm of multicore, where a large number of external factors influence the execution, guaranteed calculation of WCET may not be viable. Figure 1 shows the different probability distribution functions of the task and shows that WCET lies at the far upper bound. An expensive system would be required to meet the WCET imposed deadline as it is only considering the worst-case scenario. However, during the lifespan of the system, the WCET situation will occur with a very small probability. Depending on the required quality of service and nature of application, and if some deadline misses are tolerable, then scheduling based on variable execution time of tasks can be considered to offer a cheaper solution. In such a scenario, the estimator monitors the task execution times in previous scheduling intervals and predicts the execution times for the next interval. The underlying assumption of the estimator is that, to some extent, it is possible that the future execution times can be predicted from the past observations (i.e temporal correlation of execution times). The scheduler tries to find the schedule for the predicted execution times while respecting the task deadlines. The primary focus of this work is the design of the estimator for scheduling in soft real-time systems with a stochastic treatment of execution times of parallel tasks.

A. Motivation

As discussed earlier, the stochastic behavior of execution time of task stems from several factors and is the function of run-time parameters. To make our point clear, consider the execution time of a Macro Block (MB, 16x16 pixel block) decoding task in H.264 video decoding process. Figure 1 shows the distributions of execution times of MB for 4 different Quantization Parameters (QP). A change in QP causes a change in amount of data to be decoded. For each case, all sets of conditions were kept the same except that the QP was changed at the encoding time. Figure 1 makes it clear that MB decoding time is purely stochastic in nature. It is very important to note that MB decoding task has different distributions because of different QP. This makes it clear that no a priori knowledge can be assumed for the execution time and it's statistical properties.

As software pipelining will prevail in future for the programming of multicore systems, it gives rise to a very important and new design opportunity to improve the execution time estimation of the task. Consider two tasks (T1 and T2) running in a software pipeline such that the data output of T1 drives the input of T2 and are mapped to different cores. Both tasks are working on the same data set such that the execution time of T1 will influence the execution time of T2. A theoretical study of such dependencies and derivation of bounds can be found in [5]. The detailed analysis of such dependencies on processes in the scenario of power consumption is outlined in [7]. This generates the motivation to investigate task execution time in space (spatial correlation of execution times). Figure 2 shows the execution time profile of three tasks in a software pipelined H.264 video decoder. The three tasks of the software pipeline are (i) Network Abstraction Layer (NAL) decoder, T1 (ii) entropy decoding, inverse scanning dequatization and inverse discrete cosine transform, T2 (iii) motion compensation and in-loop filtering, T3. Each of them is a part of the decoding process of a MB. They are executed one after the other. The horizontal axis represents the successive MBs, and the execution times of each task are given on the vertical left axis (scaled arbitrarily for ease of plotting and understanding). The correlation coefficients were

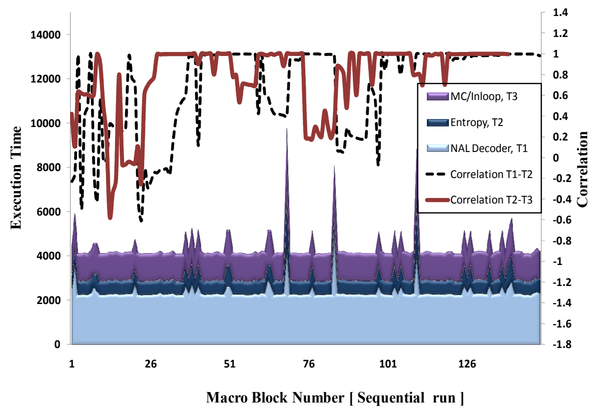


Figure 2: Correlation among execution times of tasks in software pipeline

computed (windowed correlation with two lags) and are shown on right vertical axis. The correlation axis shows that there exists a strong correlation between the execution times of the tasks in the software pipeline and most of the time hover around 1 (strong correlation). However, just like the temporal correlation, the extent of spatial correlation cannot be known a priori and must be adapted at run-time, because at some instance in time there could be a strong spatial correlation while at another instance there could be no correlation.

In lieu of the above discussion, it is clear that no a priori knowledge can be assumed for the task execution time estimation. Therefore, it is essential to derive a new general solution orthogonal to underlying architecture, task graph and the statistical properties of execution time. As a step toward this vision, this paper proposes a new framework for estimator design based on nonlinear non Gaussian joint state space modeling and also considers the spatial correlation factors among parallel tasks. To solve the nonlinear state space model recursively, we formulate the online Monte Carlo based solution. The recursive solution solves the state space model agnostic to execution time distributions and adapts the statistical properties at run time. The predicted execution times are used to schedule the tasks by variable laxity EDF scheduler. The results shows the advantage of proposed method as it achieves 76% better task utilization, 68% more task scheduling and reduction in deadline misses by 53% in comparison to state-of-the-art.

The rest of this paper is organized as follows: Section II gives an overview of the related work, Section III describes our novel methodology, followed by Section IV for experimental setup/results and Section V concludes the paper.

B. Our Novel Contributions

Within the scope of this paper, to the best of our knowledge, our novel contributions are,

- We are the first to propose the use of spatial correlation among parallel tasks to improve the execution time estimates at run-time.
- We are the first to model the parallel task execution time prediction problem in a non-linear non-Gaussian multivariate joint state space.
- We are the first to propose the solution of joint estimation of execution times and its parameters by Monte Carlo method.

II. Related Work and Motivation

A significant amount of work has been carried out for scheduling and schedulability analysis in conjunction to Worst Case Execution Time (WCET) estimation for hard real-time systems [8, 9, 10, 11]. The WCET will remain the matter of interest for a very special class of application where timeliness is the only concern and cost of the system does not matter.

Fewer publications address the analysis of applications with stochastic task execution times. Moreover, most of them consider relatively restricted application classes, and also limit their focus to the mono-processor systems. In case of a stochastic treatment, they consider a particular class of probability distribution function (e.g an exponential distribution). The task execution time estimation for independent sporadic tasks with statistical dependencies is addressed in [3]. Stochastic task execution times are considered in [13] with admittance controller for schedulability. Stochastic execution time estimation with approximations and fixed scheduling policies is discussed in [14, 15, 6]. A heuristic based solution is proposed in [16] where only the upper bounds of the missed deadline are computed.

Continuous Markov chain for parallel executing tasks is proposed in [4] for MPSoC scenario. The stationary Markov process for independent tasks is derived in [17] where task execution time can only assume the values from discrete sets. In [18], another Markovian process is considered; the analysis is performed by solving the system of linear equations and the execution time is only allowed to assume the value from finite small sets.

In most cases, WCET analysis limits the system performance, as computational resources are underutilized. The upper and lower bounds on execution time become less and less significant and irrelevant to real scheduling, as execution time tends to be more stochastic. Heuristic solutions tend to be non-optimal for the general case. The Markov Chain solutions use approximations and assume that the transition probabilities and the distributions are known. We propose a method, which does not consider any a priori knowledge and adapts at run-time and learns the statistical properties to obtain nearly optimal predictions.

III. SETS

In the scenario of variable execution time, scheduling comprises of two components: estimator and scheduler. Figure 3. elaborates the estimator/scheduler design flow. The estimator is invoked at the beginning of each scheduling interval and predicts the execution times; the scheduler then tries to schedule the tasks.

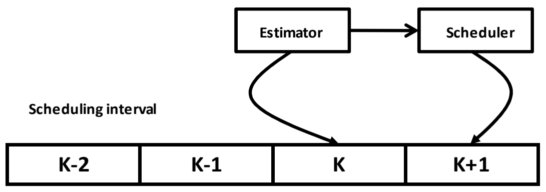


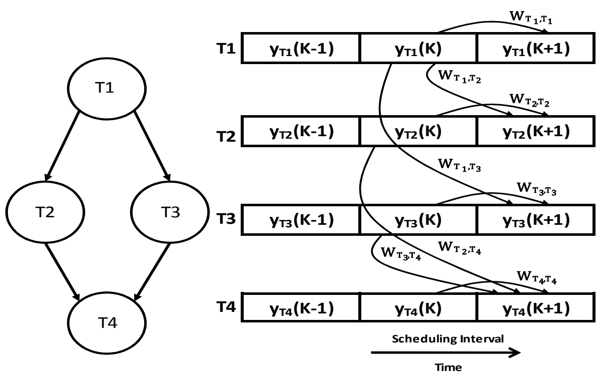
Figure 3: Estimator and scheduler flow at the beginning of each scheduling interval

In *SETS*, at the beginning of each interval, the estimator takes the difference in the predicted cycles and actual cycles utilized by individual tasks in the elapsed interval, and predicts the cycles required to complete the job in the next interval. In real world scenarios, it is extremely difficult, if not impossible, to determine and model all the parameters. Only a limited parameter vector can be used to model the execution time because of the practical considerations. Thus, the estimator is modeled as a function of temporal and spatial correlation factors based on the cycles consumed by each task in the currently elapsed interval. The unmodeled factors will cause a certain amount of error to be present in a prediction. The unmodeled factors, which affect prediction, are unknown but does show dependence on the modeled parameters due to the fact that the modeled and unmodeled parameters belongs to the same process [19]. State space model introduces state variables to compensate for the unmodeled factors. The state space model is extended to estimate the parameters in conjunction to the execution time, resulting in time-varying non-linear joint state- space. The solution of joint state space is found by the recursive formulation of Monte Carlo method, which adapts the execution time statistics at run-time and gives near to Bayesian estimates. After obtaining the predicted execution times, the variable laxity EDF scheduler is invoked to schedule the task on multicore processor. Following subsections discuss in details about each component of the *SETS*.

A. Application Model and Task Graph

To proceed forward, consider the task graph of Figure 4.a with four tasks. Task T1, is the predecessor (Producer) and the first task in the software pipeline. T1 processes the data and hands it over to task T2 and T3 (consumers) for further processing. The software pipelining of T2 and T3 as consumer of T1 constitute the parallel processing in time (forking). The task T4 has two predecessors namely T2 and T3, and act as the joining task for both, parallel processing in space (Joining). In the task graph of Figure 4.a, no task is consumer and producer at the same time for any task. It means that, we are only considering the acyclic linear task graphs. In theory, there is no restriction to model and formulate the solution for cyclic graphs but practical limitations prohibit it. This limitation stems from the formation of ill-posed first order difference equations in the model, which seriously hampers the estimation quality. To override the problem solution may incorporate the unscented transform in the solution of state space model and will be considered in the future work. The task graph of Figure 4 is comprehensive in its nature as it contains both forms of the parallelism (time and space parallelism). The task graph of Figure 4.a is considered for the understanding of the derivation of the state space model, and a general solution for any number of tasks is also developed subsequently.

B. State Space Model for Parallel Tasks



T1
T2
T3
T4
$y_{T1}(K-1)$ $y_{T1}(K)$ $y_{T1}(K+1)$
$y_{T2}(K-1)$ $y_{T2}(K)$ $y_{T2}(K+1)$
$y_{T3}(K-1)$ $y_{T3}(K)$ $y_{T3}(K+1)$
$y_{T4}(K-1)$ $y_{T4}(K)$ $y_{T4}(K+1)$
W_{T_1,T_1}
W_{T_1,T_2}
W_{T_2,T_2}
W_{T_1,T_3}
W_{T_3,T_3}
W_{T_3,T_4}
W_{T_2,T_4}
W_{T_4,T_4}
Scheduling Interval
Time

(a) (b)

Figure 4: (a) Application task graph (b) Prediction process with temporal and spatial correlation

General state space model is defined by the pair of time update and measurement update equation of the following form

$$X(k+1) = AX(k) + BU(k) \qquad (1.a)$$
$$\hat{Y}(k) = CX(k) \qquad (1.b)$$

whereas[1]

X : is the state vector, $\hat{Y}$: is the output vector
U : is the input vector, : is the system matrix
B : is the input matrix, : is the output matrix

[1] From now onwards vectors and matrices will be represented in upper case and scalars in lower case letters.

To model the execution time of tasks of Figure 4.a in a state space, we pose it as a multivariate problem in a time series. Figure 4.b depicts the prediction process in pictorial form. In Figure 4.b, $y_{T1}(k), y_{T2}(k), y_{T3}(k)$ and $y_{T4}(k)$ are the execution times in kth scheduling interval, of task T1, T2, T3 and T4 respectively. The aim is to predict the execution times in the next interval, $k+1$. To proceed forward consider task T1, it is the first task in the task graph and has no predecessor task. In the absence of predecessor task, prediction of execution time of T1 can only be carried out on the basis of temporal correlation. If $w_{T1,T1}$ is the temporal regression coefficient associated with the interval k then the predicted execution time for the interval $k+1$ considering the first order regression is defined as

$$\hat{y}_{T1}(k+1) = w_{T1,T1} y_{T1}(k) \qquad (2.a)$$

whereas $\hat{y}_{T1}(k+1)$ is the predicted execution time in the next interval. Now consider task T2 with predecessor T1, having temporal and spatial correlation with itself and with T1. To form multivariate second order regression equation for T1, we define the temporal coefficient as $w_{T2,T2}$ and spatial correlation coefficient as $w_{T1,T2}$. $w_{T1,T2}$ defines the extent of spatial correlation between T1, the predecessor, and T2, the successor. Then the second order multivariate regression equation for T2 is defined by

$$\hat{y}_{T2}(k+1) = w_{T1,T2} y_{T1}(k) + w_{T2,T2} y_{T2}(k) \qquad (2.b)$$

Similarly, second and third order multivariate regression equations for T3 and T4 respectively can be obtained as shown below, (for elaboration refer to Figure 4.b).

$$\hat{y}_{T3}(k+1) = w_{T1,T3} y_{T1}(k) + w_{T2,T3} y_{T3}(k) \qquad (2.c)$$

$$\hat{y}_{T4}(k+1) = w_{T2,T4} y_{T2}(k) + w_{T3,T4} y_{T3}(k) + w_{T4,T4} y_{T4}(k) \qquad (2.d)$$

Arranging Eq. (2.a,b,c,d) as the system of linear equations and writing in matrix form yields

$$\begin{bmatrix} \hat{y}_{T1}(k+1) \\ \hat{y}_{T2}(k+1) \\ \hat{y}_{T3}(k+1) \\ \hat{y}_{T4}(k+1) \end{bmatrix} = \begin{bmatrix} w_{T1,T1} & 0 & 0 & 0 \\ w_{T1,T2} & w_{T2,T2} & 0 & 0 \\ w_{T1,T3} & 0 & w_{T3,T3} & 0 \\ 0 & w_{T2,T4} & w_{T3,T4} & w_{T4,T4} \end{bmatrix} \begin{bmatrix} y_{T1}(k) \\ y_{T2}(k) \\ y_{T3}(k) \\ y_{T4}(k) \end{bmatrix}$$

Naturally, prediction will accompany with an error between measured and predicted execution time; it is referred to as 'innovation', and for any task Ti is defined as

$$\varepsilon_{Ti}(k) = y_{Ti}(k) - \hat{y}_{Ti}(k) \qquad (3)$$

The above given system of linear equations and the definition of error imply the direct formulation of state space model of Eq. (1.a,b) [20]. The components of the state space model for the problem of Figure 5 can be obtained by rearranging

$$\left\{ \begin{array}{c} A = \begin{bmatrix} w_{T1,T1} & 0 & 0 & 0 \\ w_{T1,T2} & w_{T2,T2} & 0 & 0 \\ w_{T1,T3} & 0 & w_{T3,T3} & 0 \\ 0 & w_{T2,T4} & w_{T3,T4} & w_{T4,T4} \end{bmatrix}, X(k) = \begin{bmatrix} y_{T1}(k) \\ y_{T2}(k) \\ y_{T3}(k) \\ y_{T4}(k) \end{bmatrix} \\ U(k) = \begin{bmatrix} \varepsilon_{T1}(k) \\ \varepsilon_{T2}(k) \\ \varepsilon_{T3}(k) \\ \varepsilon_{T4}(k) \end{bmatrix}, B = I_{4X4}, C = I_{4X4} \end{array} \right\} \qquad (4)$$

Replacing Eq. (4) into Eq. (1.a,b) yields the state space model. The system matrix A defines the model of parallel tasks and encapsulates all dependencies and associated coefficients. In the derivation of state space model, we define the regression coefficients, $w_{i,j}$ for each dependency. Let's define the regression parameter vector as

$$W = \left[w_{T1,T1} w_{T1,T2}\, w_{T2,T2}\, w_{T1,T3}\, w_{T3,T3}\, w_{T2,T4}\, w_{T3,T4}\, w_{T4,T4} \right]^T \quad (5)$$

The primary goal of the solution of state space model is to minimize prediction error and hence the objective function can be defined as

$$J_{min} = Min(\varepsilon_{Ti}(k)) \qquad (6)$$

C. Joint State Space Model

In general, for a given state space model, the parameters vector of Eq. (5) is assumed to be known. To obtain the parameters, system identification process is carried out at design time and recalibration runs are performed from time to time to adapt the vector according to the changing scenarios [19]. Model parameters obtained by calibration runs cannot adapt to rapidly changing scenarios at run-time and are always valid under certain assumptions. No assumption can be made on execution time or its characteristics and hence it is essential to derive the solution independent of the prior knowledge of system dynamics. Within the framework of state space, literature widely refers maximum likelihood method for the estimation of parameters at run-time. During the course of this work, it is observed that the computational cost and sampling errors in W due to approximations render maximum likelihood impractical. In that case, we considered the parameter estimation by Bayesian estimation. To obtain such a solution, the key idea is to estimate the model parameters like the states in a state space model. The estimation of model parameters in conjunction to the states of the model is achieved by augmenting the state vector with model parameters. The new augmented state vector by augmenting Eq. 5 in the state vector of Eq. (4) is defined as

$$X_J(k) = [X(k)\; W(k)]^T \qquad (7)$$

It is important to note that in the augmented state vector model parameters are also time-varying and are updated at the beginning of each interval, like states. As a result, the state space of Eq. (1.a,b) with parameters in state vector become time-varying and the time update and measurement equation of Eq. (1.a,b) takes the following form

$$X_J(k+1) = A_J(W(k), k) X_J(k) + B_J U_J(k) \qquad (8.a)$$

$$\hat{Y}(k) = C_J X_J(k) \qquad (8.b)$$

The components of the state space models are augmented with identity $(I_{i,i})$ and null $(O_{i,i})$ matrices to satisfy the system of equations (number of equations must be equal to the number of unknowns). The state vector of Eq. (7) and augmented components of Eq. (4) define the joint state space components and are given as

$$\left\{ \begin{array}{cc} A_J = \begin{bmatrix} A & O_{4,8} \\ O_{8,4} & I_{8,8} \end{bmatrix} & X_J(k) = \begin{bmatrix} X(k) \\ W(k) \end{bmatrix} \\ X_J(k+1) = \begin{bmatrix} X(k+1) \\ W(k+1) \end{bmatrix} & U_J(k) = \begin{bmatrix} U(k) \\ O_{8,1} \end{bmatrix} \\ B_J = \begin{bmatrix} B & O_{4,8} \\ O_{8,4} & O_{8,8} \end{bmatrix} & C_J = \begin{bmatrix} C & O_{4,8} \\ O_{8,4} & O_{8,8} \end{bmatrix} \end{array} \right\} \qquad (9)$$

Putting Eq. (9) into Eq. (8) yields the complete joint state space model

$$\begin{bmatrix} X(k+1) \\ W(k+1) \end{bmatrix} = \begin{bmatrix} A & O_{4,8} \\ O_{8,4} & I_{8,8} \end{bmatrix} \begin{bmatrix} X(k) \\ W(k) \end{bmatrix} + \begin{bmatrix} B & O_{4,8} \\ O_{8,4} & O_{8,8} \end{bmatrix} U_J(k) \quad (10.a)$$

$$\hat{Y}(k) = C_J = \begin{bmatrix} C & O_{4,8} \\ O_{8,4} & O_{8,8} \end{bmatrix} \begin{bmatrix} X(k+1) \\ W(k+1) \end{bmatrix} \qquad (10.b)$$

It is worthwhile to mention that the resultant matrices of the joint state space model are sparse and are shown in their full form for the sake of understanding and compliance with the theory. The sparse property and regularity of the model significantly reduces the computation cost at run time. Consider the measurement update equation Eq. (10.b); at first it appears that it involves the multiplication of matrix of dimension 12×12 with vector of dimension 12×1. But in actual implementation there is no multiplication involved and the first four states of vector $X_J(k)$ are copied to $\hat{Y}(k)$. Similar patterns can also be observed in time update equation e.g. the multiplication of observations and input matrix. Therefore, in actual implementation, the solution of state space requires only a handful of computations. The optimal solution of the joint state space model can be found by numerical integration, but the computational cost of numerical integration prohibits its use at run-time [21]. The recursive solution of derived state space can also be found by Extended Kalamn Filter (EKF), but it require the derivation of linear state space at design time and involves linearization step at run-time. Moreover, EKF leads to the approximation of nonlinear state space by making the piecewise linear estimation, inherent error, and can only be applicable in the case of Gaussian distributions. As elaborated in the 'motivation' sub-section, distributions of the execution time cannot be assumed a priori and hence it is required to formulate the solution independent of it.

D. Generalization

The regularity in the derived state space model of Eq. (10.a,b) allows the direct derivation of the model by algorithm for any number of tasks in a task graph. The task graph may contain the isolated task(s) and can comprise of collection of arbitrary sub graphs. Algorithm 1 outlines a systematic procedure for the derivation of joint state space model. It is important to note that the labeling of tasks should be carried out in a way such that the predecessor task must be labeled before all of its successors. However, defined labeling rule is not a constraint but has the implications for the implementation. If tasks are labeled according to the rule then the system matrix will always be a lower diagonal matrix (practical limitations for considering the acyclic linear task graphs only are discussed in Section III.A). The lower diagonal matrix means, half of the operations of major computational part are not required at all (multiplication by zeros).

```
Algorithm 1:
Event: Derive joint state space model for Ω number of tasks
Definition:
$A$: System matrix, $B$: Input matrix, $C$:Output matrix
$X$: State vector,   $U$: Input vector, $\hat{Y}(k)$: Output vector
$A_j, B_j, C_j, X_j$ and $U_j$ are the joint state space counterparts of
$A, B, C, X$ and $U$ respectively
1: label the tasks in ascending order such that predecessor task in
   software pipeline should be label before any of its successors
2: $B = C$ = identity matrix of dimension ΩxΩ,
        $X = [x_1\ x_2\ \dots x_\Omega]^T$
3: $= \varepsilon_{T1}$ ; $X = y_{T1}$
4:   for ( i=2 ; i <= Ω, i++ )
5:        Augment $\varepsilon_{Ti}$ as a last row in $U$
6:        Augment $y_{Ti}$ as a last row in $X$
7:   end
8: A = null matrix of dimension ΩxΩ, cnt = 0
9:   for ( i=1 ; i<=Ω ; i++ )              //for rows
10:      for ( k=1 ; k<=Ω ; k++ )          //for columns
12:          if ( i==k )
13:             insert $w_{Ti,Ti}$ in $A$ at ith row, ith column
14:          end
15:          if ( k < i )
16:             if ( $T_i$ Predecessor of $T_k$ )
17:                insert $w_{Ti,Tk}$ in A at ith row, kth column
18:             end
19:          end
20:       Augment $w_{Ti,Tk}$ as a last row in $W$, cnt++
20:       end
21:   end
22: $X_J = [X\ \ W]^T$ , $A_J = \begin{bmatrix} A & O_{\Omega,\text{cnt}} \\ O_{\text{cnt},\Omega} & I_{\text{cnt,cnt}} \end{bmatrix}$, $U_J = \begin{bmatrix} U \\ O_{cnt,1} \end{bmatrix}$
23: $B_J = \begin{bmatrix} B & O_{\Omega,\text{cnt}} \\ O_{\text{cnt},\Omega} & O_{\text{cnt,cnt}} \end{bmatrix}$, $C_J = \begin{bmatrix} C & O_{\Omega,\text{cnt}} \\ O_{\text{cnt},\Omega} & O_{\text{cnt,cnt}} \end{bmatrix}$
```

E. Monte Carlo Recursive Solution

To develop the solution of joint state space model independently of the distribution, we formulated the recursive solution based on online Sequential Monte Carlo method. Contrary to conventional solutions, sequential Monte Carlo method builds the underlying distributions at run-time and learns the execution time statistics at the beginning of each interval. At the start of estimation process, 's' numbers of randomly initialized samples are generated to simulate the underlying distributions. After initialization, at the beginning of the interval, it is not needed to regenerate the population of samples from scratch. Rather distribution weights are adjusted according to the error of new observation vector. Here new observation vector corresponds to the last prediction and the objective is to minimize the error and obtain the new state vector, the prediction. To achieve this goal, state space model with N states is solved to compute N weights, one appropriately defined weight for each state. The reader is referred to [12] (particularly chapter 9 for this work) for the understanding of the formulation of online Monte Carlo method.

The recursive solution by Monte Carlo comprise of two steps: (i) Correct (measurement update) and (ii) Predict (time update). The recursive solution means that only the prediction from the previous interval and current measurement are needed to compute the predictions. Contrary to batch estimation techniques, no history of previous predictions and observation is required. The correction step is carried out on the arrival of the new observation vector, as observation vector comprises of errors between previously predicted and now available execution times. After the correction step, the new statistics and dynamics of execution times till the current time are known and the prediction step is performed to compute the estimates for the next interval. Figure 5 shows the correction/prediction recursion steps computed at the beginning of each scheduling interval.

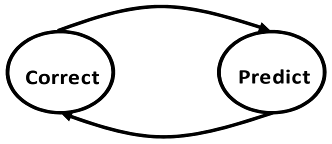


Figure 5: Correction and prediction recursion

To derive the solution for the state vector of size N under the influence of an arbitrary distribution, assume S independent identically distributed (i.i.d) variables denoted by $\beta^1, \beta^2 :: \beta^S$. This sampling follows the probability distribution function (p.d.f.) for state vector X_j as $p(X_j)$ i.e. $\beta^{1:S} \sim p(X_J)$. $p(X_J)$ is not known but can be approximated by the following function [22]

$$p(X_J) \simeq p^S(X_J) = \frac{1}{S}\sum_{i=1}^{S} \pi\beta^i(X_J)$$

whereas, π is a probability of the sample. In approximating $p(X_J)$ it is assumed that all samples β^i contribute equally in the approximation of $p(X_J)$. To generalize the approach, assign the weight factors α to the point β^i. The weight factors also satisfy the normality condition $\sum_{i=1}^{S} \alpha^i = 1$. In that case

$$p(X_J) \simeq p^S(X_J) = \sum_{i=1}^{S} \alpha^i \pi\beta^i(X_J) \qquad (11)$$

If $p(\beta^i)$ is known then the probability $p(X_J)$ can be approximated by using the discrete values of the p.d.f. $p(\beta^i) = \alpha^i$. If sampling over the p.d.f. $p(X_J)$ is unavailable, then one can use a p.d.f. $\bar{p}(X_J)$ with a similar support set, i.e. $p(X_J) = 0$ implies that $\bar{p}(X_J) = 0$. Then it holds that the expectation of the state vector is approximated by

$$E(\Phi(X_J)) = \int \Phi(X)p(X_J)dx = \int \Phi(X_J)\bar{p}(X_J)\frac{p(X_J)}{\bar{p}(X_J)}dx$$

Where $\Phi(X_J)$ is the solution of the state equation. If S samples of $\bar{p}(X_J)$ are available at points $\widetilde{\beta^1}:\widetilde{\beta^2}$ ::: $\widetilde{\beta^S}$: i.e $\bar{p}(\widetilde{\beta^i}) = \pi\beta^i(X_J)$ and the weight coefficient $\alpha^i = \frac{p(X_J)}{\bar{p}(X_J)}$ then it can be shown that

$$E(\Phi(X_j)) \simeq \sum_i^S \alpha^i \, \Phi(\widetilde{\beta^i}) \qquad (12)$$

Eq. (12) assumes that the p.d.f. $p(X_J)$ is unknown (target distribution), however the p.d.f. $\bar{p}(X_J)$ (importance law) the unknown instrumental distribution is available. Then, it is sufficient to sample on $\bar{p}(X_J)$ and find the associated weight coefficients α^i to compute $(\Phi(X_J))$.

The prediction step:

Let $Y^- = [Y(1), Y(2), \ldots Y(k-1)]$ and $Y = [Y^- \, Y(k)]$ then to compute the $p(X_J(k)|Y^-)$, according to Eq. (11) it holds that the a priori probabilities can be found as

$$p(X_J(k-1)|Y^-) = \sum_{i=1}^S \alpha_{k-1}^i \, \pi_{\beta_{k-1}^i}(X_J(k-1))$$

The posteriori probabilities can be obtained by using Bayes law

$$p(X_J(k)|Y^-) = \sum_{i=1}^S \alpha_{k-1}^i \pi_{\beta_k^i}(X_J(k)) \qquad (13)$$

With $\beta_k^i \sim p(X_J(k)|\, X_J(k-1)) = \beta_{k-1}^i$

The Eq. (13) means that the state equation of the system executes N times, starting from the N previous values of the state vectors $X_J(k-1) = \beta_{k-1}^i$

$$\hat{X}_J(k+1) = A_J X_J(k) + B_J U_J(k) + [\alpha_{k-1}^i]_{i=1}^N$$

Thus new state vector is obtained, $\hat{X}_J(k+1)$, and consequently the mean value of the state vector will be given from Eq. (13).

The correction step:

A posteriori probability density is found using Eq. (13) and now a new measurement vector $Y(k)$ is available and the objective is to compute the corrected priori probability density $p(X_J(k) \mid Y)$; from Bayes law it holds that

$$p(X_J(k)|Y) = \frac{p\left(Y(k)\middle|X_J(k)\right)p(X_J(k)|Y^-)}{\int p(Y(k)|X_J(k), Y^-)p(X_J(k)|Y^-)dx} \qquad (14)$$

Substituting Eq. (13) into Eq. (14) and after derivation it is finally obtained

$$p(X_J(k)|Y) = \sum_{i=1}^S \alpha_k^i \, \pi_{\beta_k^i} \, (X_J(k)) \qquad (15)$$

where

$$\alpha_k^i = \frac{\alpha_{k-1}^i p\left(Y(k)\middle|X_J(k)\right)}{\sum_{j=1}^S \alpha_{k-1}^j p\left(Y(k)\middle|X_j(k)\right)}$$

Eq. (15) gives the corrections for the state vector in each iteration. The given Monte Carlo solution can be viewed as hidden Markov process, with the key difference that the state variables assume values from continuous space as opposed to the discrete state space for the hidden Markov model. Though to obtain the prediction, only previous prediction and current available observations are considered, but due to the recursive solution, the prediction is the combination of all previous observations, which is a very important property of the online Monte Carlo sampling based solutions. As time proceeds, the sampling of Monte Carlo learns the system dynamics by minimizing the error. Figure 6 illustrates this fact; prediction for the interval K+1 is based on the first interval till Kth interval. This is also the reason that at the start of the process the predictions for the first few intervals are not meaningfull as the proposed solution is in learning phase (convergence).

F. Degeneration of Sample

The recursive solution described by Eq. (13) and (15) has a tendency of degeneracy of samples. After a certain number of iterations, almost all the weights α_k^i tend to 0. That means the samples lose effectiveness as time proceeds. Ideally all the weights should converge to the value $1/S$, i.e. the samples should have the same significance. Therefore, it is necessary to include the re-sampling step to block the degeneration of samples. After the completion of recursion, samples of low weight factors are removed and replaced by the duplicates of the samples with high weight factors, as suggested in [23].

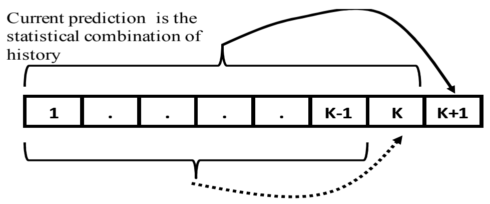

Current prediction is the statistical combination of history
1
.
.
.
.
K-1
K
K+1

Figure 6: Learning at each interval and prediction is the combination of whole history

G. Variable Laxity EDF Scheduler

To schedule the task with variable execution time the requirement is to incorporate it in the scheduling decisions. Earliest Deadline First (EDF) is an optimal scheduler for uni-processor real time systems and is also widely used for multicore systems. EDF scheduler primarily works on laxity, the measure of the spare time permitted for the task before it misses its deadline. According to tuple defined in Section 1 (start time, S_t, execution time, C_t and deadline, D_t), laxity at the start time S_t for each task can be computed by: laxity = (D_t - C_t). In our case, C_t is variable and is obtained at the beginning of each scheduling interval. Because of variable laxity and multicore scenario, we used the variable laxity based EDF scheduler for multicore platforms proposed in [24].

IV. RESULTS

We carefully selected the H.264 video decoder application against generic task graphs or simple applications. The control dominant and computation-intensive properties of H.264 decoder fit well for the benchmarking and evaluation of all points raised in the proposed method. We have used our in-house developed H.264 decoder, which offers time and space parallelization features simultaneously. The space parallelization is achieved by enabling the decoding of multiple slices within a single frame and time parallelization is achieved by dividing the MB decoding process into the software pipeline. Figure 7 shows the task graph of decoder and Table 1 defines the functionality of each task. It is evident from Figure 7 that the decoder is able to decode two

Table I: Tasks and their functionality

Task	Functionality
T1	Network abstraction layer decoder
T2	Slice 1: Entropy decoder, Inverse Scaling/D-Quantizatiion and Inverse discrete cosine transform
T3	Slice 2: Entropy decoder, Inverse Scaling/D-Quantizatiion and Inverse discrete cosine transform
T4	Slice 1: Motion compensation and inloop filter
T5	Slice 2: Motion compensation and inloop filter
T6	Slice to frame merger (logical task)

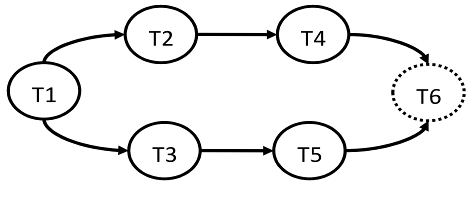


Figure 7: Slice parallel and pipelined parallel H.264 decoder

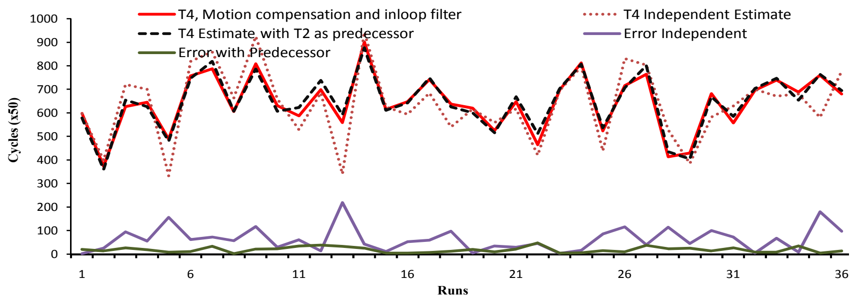


Figure 8: Prediction with and without predecessor

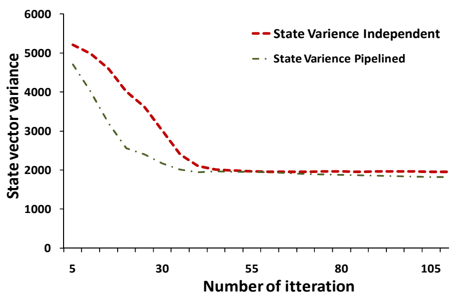


Figure 9: Convergence

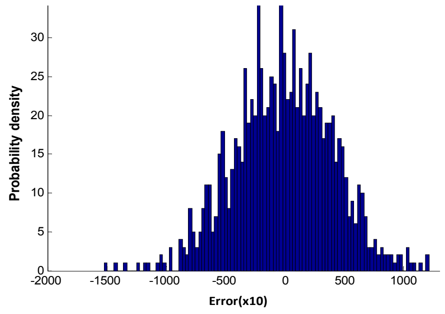


Figure 10: Probability density function of error in the prediction of T1

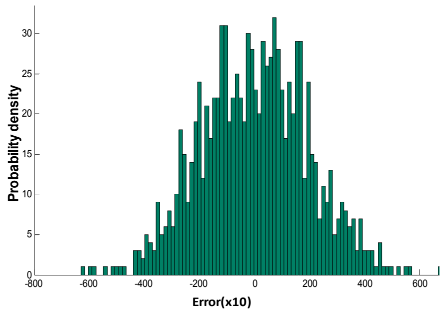


Figure 11: Probability density function of error in the prediction of T4

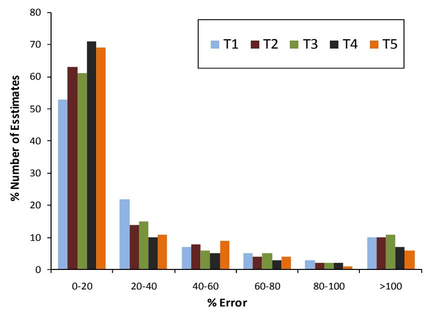


Figure 12: Prediction error

slices simultaneously, the numbers of slices in a frame are decided at the video encoding time. Multiple test sequences of diverse characteristics are used in evaluation and are derived from ITU-T test vectors, namely BBC, Rushhour and Container with 4CIF resolution. First 50 frames of each data set are encoded by JM13 H.264 reference encoder for different QPs (16, 20, 24, 28) to vary the data characteristics and the execution time at decoding. Two slices per frame are encoded to enable the space parallelization at decoding. Formation of slices is static and the frame is divided equally between the slices by partitioning the frame into upper and lower parts.

The experiments are conducted on quad core platform. The evaluation platform is based on Intel® Quadcore (Q9100) processor with 4GB of memory, 1.333 GHz bus interconnects and Windows® XP operating system. Threads are statically mapped on the cores and run time thread migration is disabled using thread affinity. Intel® VTune performance analyzer is enabled to gather the data from hardware sampling units to obtain precise cycle count for each task. The scheduling evaluation is done on Cheddar [25], a real-time scheduling tool.

We first evaluate the impact of the inclusion of spatial correlation in the estimation model. Figure 8 shows the estimation of the execution time of T4 (motion compensation and in-loop filtering for slice 1) in two different scenarios. The prediction of T4, as independent task and also as the consumer of T2. Results of Figure 8 clearly show that the predicted values lie closer to actual when we take into account the execution time of predecessor task, T2. The absolute error plots reveal that the inclusion of the execution time of the predecessor not only reduces the error but it stays below the error when the estimation is performed as an independent task. It is apparent that the inclusion of spatial correlation in the execution time model significantly improves the quality of the prediction. Figure 9 shows the state variance, an indirect measure of convergence, of estimator from the beginning of the estimation process for task T4. Convergence curves make it clear that the pipelined estimate converges rapidly than independent estimate. Moreover, state covariance of the pipelined estimate remains lower after the convergence and yields stable and better prediction. Figure 10 and 11 shows the probability distributions of error in the prediction of T1 and T4 respectively. In case of T1, the error distribution is widely distributed while for T4, error distribution has limited range. Properties of distributions also suggest that significant improvement has been achieved by considering the spatial correlation among tasks. Figure 12 shows the prediction error for five tasks of the application. In Figure 12, percentage residual error is computed for the comparison purposes and the estimates are clustered into the bins with width of 20% of error. The prediction error greater than 100% is combined into a single bin instead of showing the lengthy decaying tail of bins. The analysis of prediction process and task execution time characteristics reveals that the abrupt change in the task phase leads to the large prediction error, sometime this error is greater than the 100%. Results clearly show that the prediction error by *SETS* largely falls within 20% of the error. The computational overhead of the proposed task execution time prediction process for the experimental setup is 7.8% (on average). This overhead is relatively very small compared to the benefits obtained in term of task utilization. Furthermore, computational overhead also depends on the granularity of the tasks. In the experimental application of this paper tasks are of fine granularity and this overhead diminishes for the coarse grained tasks. Without any kind of a priori knowledge about the tasks or computing platform the quality of prediction can considered as sufficient for schedulability analysis.

To analyze the scheduling we used the actual execution time and predicted execution time traces and simulated the scheduling process on Cheddar. On Cheddar, 4 cores are assumed and variable laxity EDF scheduling scheme proposed in [24] is implemented. We experimented with non-preemptive uniform priority attributes. We measured the results in terms of task utilization (=execution time/deadline), task schedulability (number of tasks that can be scheduled) and deadline misses. We compared our results with state-of-the-art proposed in [6] as it also considered the stochastic execution time behavior of the tasks. The work done in [6] like this paper argues that the WCET seriously underutilizes the resources and an average case must be considered. Moreover, for scheduling, [6] also considers the EDF policy and establishes the fairness of the comparison. The prime difference in [6] and SETS is that [6] derives the prior tardiness bound for global EDF. However, the paper only considers the upper bounded deviation from the mean execution time of the task. The absence of lower bound results in the under-utilization of task when execution time falls below the mean execution time. The simulation on Cheddar shows that the *SETS* improves the task utilization to the factor of 0.37, an improvement of 87%. Figure 13 summarizes the results of the task utilization, task schedulability and the deadline misses. The results clearly show the advantage of *SETS* over *Mills* [6]. The improvement in task utilization achieved by *SETS* is 76% better than the *Mills*. The huge improvement in the task utilization is due to the lack of consideration of the execution time below the mean value. The *SETS* does not define upper or lower bounds on execution times, computes it at run time, and hence gives the better utilization. The same trend can be observed for the task schedulability. However, the percentage improvement of the *SETS* over the *Mills* is 68% because of some large over-prediction of execution times by the *SETS*. The large over-predictions benefit *SETS* in terms of the deadline misses. The task schedulability and deadline misses can be considered as a tradeoff. Lastly, the task scheduled by the *SETS* framework are less susceptible to deadline misses. The Figure 13 shows that average deadline missed by the *SETS* are 53% less compared to *Mills*.

The results and their analysis make it clear that the spatial correlation has profound positive impact on the quality of prediction. Moreover, run time estimate of execution time alleviates the constraints of fixed time scenario and thus significantly improves the effective utilization of system resources. Therefore, *SETS* is the applicable solution for the scheduling of soft real-time systems.

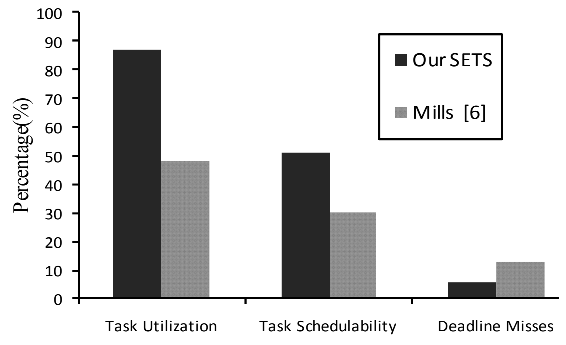


Figure 13: Comparison of SETS with Mills for different metrics

V. CONCLUSION

We propose scheduling techniques for soft real-time systems exhibiting stochastic task execution times. Our solution is estimating their execution times using a joint state space model. The solution of the proposed model is found by a Monte Carlo sampling based recursive technique to estimate the execution time of the task independent of distributions. Moreover, state space modeling opens the opportunity to take benefit of the large body of work that already exists for solving state space. Within this work, we formulated an online Monte Carlo method for the solution of state space. The recursive solution significantly reduces the memory and computational requirements compared to other sophisticated methods (i.e. Markov Chains) obtaining nearly a Bayesian estimate. We have shown that the inclusion of spatial correlations among execution time of tasks significantly improves the prediction quality. The proposed method may not scale well in case of thousand of tasks in a task graph. A workaround can be found by investigating the distributed estimation by combining the most relevant tasks in a single cluster. Furthermore, the computational cost of Monte Carlo method can be reduced by generating the proposal distribution by RANdom Sample Consensus (RANSAC) for run time distributions, and will be considered in future work. The scheduling by variable laxity EDF on multicore systems for variable execution times shows a 76% improved task utilization.

VI. REFERENCES

[1] Edward A. Lee, “Building unreliable system out of reliable components: A real time story”, http://www.eecs.berkeley.edu/Pubs/

[2] Shekhar Borkar, Norman P. Jouppi and Per Stenström, “Microprocessors in the Era of Terascale Integration”, Design automation and test in Europe, pp. 237-242, 2007.

[3] A. Burns, G. Bernat, and I. Broster. “A probabilistic framework for schedulability analysis”. EMSOFT, pp. 1-15, 2003

[4] Sorin Manolache, Petru Eles, Zebo Peng. “Schedulabil-ity analysis of multiprocessor real-time applications with stochastic task execution times”, ICCAD, pp. 699-706, 2002.

[5] Val Donaldson and Jeanne Ferrante, “Determining Asynchronous Acyclic Pipeline Execution Times”, IPPS, pp 568-72,1996.

[6] Alex F. Mill and J. Anderson, “A stochastic framework for multiproces-sor soft real time scheduling“, IEEE RTAS 2010.

[7] S. Yaldiz, A. Demir and S. Tasiran, “Stochastic modeling and optimiza-tion for energy management in multicore systems: A video decoding case study”, IEEE trans. on computer aided design, pp. 1264-1277, 2008

[8] E. Bini, G. Butazzo, and etl. “A hyperbolic bound for the rate monotonic algorithm”. In Proceedings of the 13th Euromicro Conference on Real-Time Systems, pp. 59–66, 2001.

[9] J. C. Palencia Gutierrez and M. Gonzalez Harbour. “Schedulability analysis for tasks with static and dynamic offsets”. In Proceedings of the 19th IEEE RTS Symposium, pp. 26–37, 1998.

[10] R.Wilhelm and etl., “The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans. Embedded Comput. Syst. 7(3): 2008.

[11] Guillem Bernat, Antoine Colin and Stefan M.Petters, “WCET analysis for probabilistic hard real time systems”. IEEE Real-Time Systems Symposium, pp. 279-288, 2002.

[12] A.Doucet, N.Freitas, N.Gordon, A. Smith, “Sequential Monte Carlo Methods in Practice”, Springer 2001.

[13] L. Abeni and G. Butazzo. “QoS Guarantees using probabilistic deadlines”. IEEE ECRTS, pp. 242-249, 1999.

[14] J. Kleinberg, Y. Rabani, and E. Tardos. “Allocating bandwidth for bursty connections” SIAM Journal on Computing, 30(1), pp. 191–217, 2000.

[15] A. Goel and P. Indyk. “Stochastic load balancing and related problems”. In IEEE Symposium on Foundations of Computer Science, pp. 579–586, 1999.

[16] S. Hua, G. Qu, and S. Bhattacharyya. “Exploring the probabilistic design space of multimedia systems”. In Proceedings of the 14th IEEE InternationalWorkshop on Rapid Systems Prototyping, 2003

[17] J. Luis et al. “Stochastic analysis of periodic real-time systems”. In Proceedings of the 23rd Real-Time Systems Symposium, pp. 289, 2002.

[18] D. Guo, X. Wang and R. Chen, “New sequential Monte Carlo methods for nonlinear dynamic systems”. Journal of Statistical and computing, pp. 135-147, 2005.

[19] M. A. Iverson, F.O. zguner, and L.C. Potter, “Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment”, IEEE Transaction on Computers vol. 48, pp 1374-1379, 1999.

[20] J.J.F. Commandeur and S. Koopman. “Time series analysis using state space models”, Oxford University Press, USA, 2007.

[21] Simon Haykin and etl, “Kalman filtering and neural networks”, Wiley-Interscience, 2001.

[22] Tommy Oberg, “Modulation, Detection, and Coding”, John Wiley & Sons, Inc., New York, 2001.

[23] K. Pradheep Kumar and A. P. Shanthi , “Application of non-uniform laxity to EDF for aperiodic tasks to improve task utilization on multicore platforms”, CoRR 2009.

[24] Cheddar, http://beru.univ-brest.fr/~singhoff/cheddar/