APPLYING STATISTICAL DESIGN TO CONTROL THE RISK OF OVER-DESIGN WITH STOCHASTIC SIMULATION

Yi Wu[1], Peng Zhou[1*], Jian Lin[1,2] ,Wanhua Qiu[1]

[1]Dept. of Economics and Management, Beijing University of Aeronautics & Astronautics, Beijing

*[*1]Email:*shengniao@gmail.com

[2]Institute of Education, Tsinghua University, Beijing

ABSTRACT

By comparing a hard real-time system and a soft real-time system, this article elicits the risk of over-design in soft real-time system designing. To deal with this risk, a novel concept of statistical design is proposed. The statistical design is the process accurately accounting for and mitigating the effects of variation in part geometry and other environmental conditions, while at the same time optimizing a target performance factor. However, statistical design can be a very difficult and complex task when using classical mathematical methods. Thus, a simulation methodology to optimize the design is proposed in order to bridge the gap between real-time analysis and optimization for robust and reliable system design.

Keywords: Percolated stochastic, Simulation optimization, Statistical design

1 INTRODUCTION

Soft Real-time embedded systems, such as battery-operated PDAs etc., have become increasingly popular in our life. Unlike hard real-time counterparts, soft real-time applications are only expected to guarantee "expected delay" over input data space. However, timing is traditionally treated as a hard constraint throughout the system design process. As a result, applications are often pessimistically analyzed for worst case scenarios, and the slowest responses determine their performance. Although this is a necessity for hard real-time applications, soft real-time counterparts can occasionally take longer than the deadline to finish some tasks. They are often expected to guarantee an expected delay (or latency) rather than a worst case execution time over the input data space. For example, embedded multimedia systems require the processing of signal, image, and video data streams in a timely fashion to the end user's satisfaction. Such applications are often characterized by repetitive processing on periodically arriving inputs, such as voice samples or video frames, and the tolerance to occasional deadline (determined by the throughput requirement of the input data streams) misses without being noticed by human visual and auditory systems. In packet audio applications, loss rates of one to ten percent can be tolerated.

As soft real-time systems can tolerate some violations of timing constraints, those methods which guarantee no deadline missing by considering worst-case execution time (WCET) of each task will often lead to over-designed systems that deliver higher performance than necessary at the cost of expensive hardware, higher energy consumption, and other system resources.

There are plenty of studies on the estimation of soft real-time system's probabilistic performance when the application's computation time can be varied. However, most of their goals are to improve the system's performance or to provide probabilistic performance guarantees. Our recent work discusses a percolated stochastic simulation approach to optimize a system's design by taking advantage of a soft real-time application's tolerance to deadline misses. To the best of our knowledge, there is no reported effort on systematically incorporating an application's performance requirements, uncertainties in execution time,

and tolerance for reasonable execution failures to guide rapid and economic optimization of real-time embedded systems.

In this paper, we study the problem of how to integrate such tolerance to deadline misses into the optimization of soft real-time systems. We propose the novel concept of statistical design for multimedia systems and a simulation methodology to optimize the design. Given the execution time distribution of each task, we have developed a simulation algorithm to estimate the probabilistic timing performance and to manage system resources in such a way that the system achieves the required completion ratio probabilistically with a reduced amount of system resources. This method relaxes the rigid hardware requirements for software implementation and eventually avoids over-designing the soft real-time system.

The rest of the paper is organized as follows: Section 2 gives the review of design methods and rules developed by former writers. The overview of our statistical design methodology is discussed in Section 3. Probabilistic timing performance estimation is discussed in Section 4. The Random Critical Path simulation approach is discussed in Section 5. Section 6 introduces our percolated Stochastic simulation approaches. Finally, the conclusion and future study are discussed in section 7.

2 REVIEW OF DESIGN METHODS AND RULES

From the late 1980s, when schedule ability analysis became one of the traditional fields of investigation in real-time systems research, worst-case execution time analysis (WCET analysis) caught the attention of the research community, and the basic methods of how to calculate a safe and tight WCET using static analysis for imperative programming languages (like C) and simple hardware (like MC68000) were presented. In the 1990s, more and more research groups focused on WCET analysis. As a result, substantial progress has been made in this area in a relatively short time.

Together with schedule ability analysis, worst-case execution time analysis (WCET analysis) forms the basis for establishing confidence in the timely operation of a real-time system. WCET analysis does so by computing (upper) bounds for the execution times of the tasks in the system. These bounds are needed for allocating the correct CPU time to the tasks of an application. They form the inputs for schedulability tools, which test whether a given task set is schedulable (and will thus meet the timing requirements of the application) on a given target system.

Figure 1 represents a typical WCET circle of designing procedure. We start with a pool of target system architectures to select from. We first estimate the worst case execution time (WCET). If this estimation meets without ***any*** deadline misses, it is kept, and we go on to the next step, system synthesis and evaluation. If this is not the case, this estimation is transferred to the system architecture pool to find a way of optimization. The next estimation is then given, and a new circle begins.

This method is pessimistic and suitable for developing systems in a “hard real-time” environment, where any deadline miss will be catastrophic. However, it is not suitable for a soft real-time system design. As is discussed in Section 1, such a soft real-time system could bear some response delay or performance loss while a hard real-time system could stand no such delay or loss. Thus, this type of WCET method will lead to an over-design risk, which may bring huge needless time cost and operation cost losses in production.

Several studies on the probabilistic timing performance estimation for soft real-time systems design have been emerging since the late 1990s. The general assumption is that each task’s execution time can be described by a probability density function that can be obtained by applying path analysis and system utilization analysis techniques. In Kalavade and Moghe (1998) the authors extend the scheduling

algorithms and schedule ability analysis methods developed for periodic tasks in order to provide a probabilistic performance guarantee for semi periodic tasks when the total maximum utilization of the tasks on each processor is larger than one. They describe the transform-task method that transforms each semi-periodic task into a periodic task followed by a sporadic task. The method can provide an absolute guarantee for requests with shorter computation times and a probabilistic guarantee for longer requests. In Hua et al. (2003), a performance estimation tool that outputs the exact distribution of the processing delay of each application is introduced. It can help the designers develop multimedia networked systems requiring soft real-time guarantees in a cost efficient manner. Given that the execution time of each task is a discrete random variable, Hu et al. propose a state-based probability metric to evaluate the overall probabilistic timing performance of the entire task set. Their experimental results show that the proposed metric reflects well the timing behavior of systems with independent and/or dependent tasks.

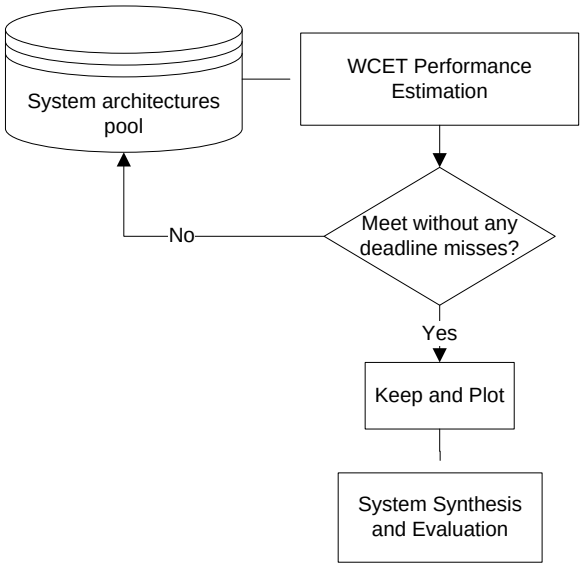


Figure 1. Hard Real-time Design Circle Based On WCET Analysis

3 OVERVIEW OF STATISTICAL DESIGN METHODOLOGY

Considering a soft real-time system, which can tolerate occasional missed deadlines, it is evidently not advisable to apply mechanically those methods and rules used in hard real-time system design. In order to avoid over-designing systems, we propose the concept of "statistical design," where we design the system to meet the timing constraints of periodic applications statistically. That is, the system may not guarantee the completion of every execution or iteration, but it will produce sufficiently many successful completions over a large amount of iterations to meet the user-specific completion ratio. Or even better, the probability that any execution will be completed is not lower than the desired completion ratio.

Clearly, the proposed "statistical design" will be preferred for many embedded soft real-time systems such as portable multimedia systems where high portability, low power consumption and reasonably good performance are equally important.

Figure 2 depicts our statistical design approach for rapid and economic system prototyping. We start with the popular dataflow graph representation of the embedded software, the system's performance requirements (in terms of timing and completion ratio constraints), and a pool of target system architectures to select from. We partition the application into a set of tasks and use profiling tools to collect detailed

execution information for each task. Next, we estimate the system timing performance by task path analysis to check whether it is feasible for the current system configuration to achieve the desired performance. If not, we change the hardware configuration and/or apply software optimization techniques and update the software profiling results that will be used in the next round of system timing performance estimation. We mention that any change to the target hardware configuration and/or software optimization may affect the application's actual execution information, and therefore, the software profiling process will need to be re-started. This iterative design loop terminates when all the design requirements are met. Once the completion ratio constraint is met, we move on to the phase of percolated stochastic simulation optimization. If the constraints can be guaranteed ***with user-specific possibility requires***, the design space of the current profile is kept and plotted. This is the key step in the proposed statistical design optimization where we 1) allocate minimum system resources to each task to make the desired completion ratio probabilistically achievable and 2) develop real time schedulers to manage the resources at run time such that the required completion ratio can be achieved probabilistically. Finally, we conduct system synthesis, simulation, and evaluation before prototyping the system.

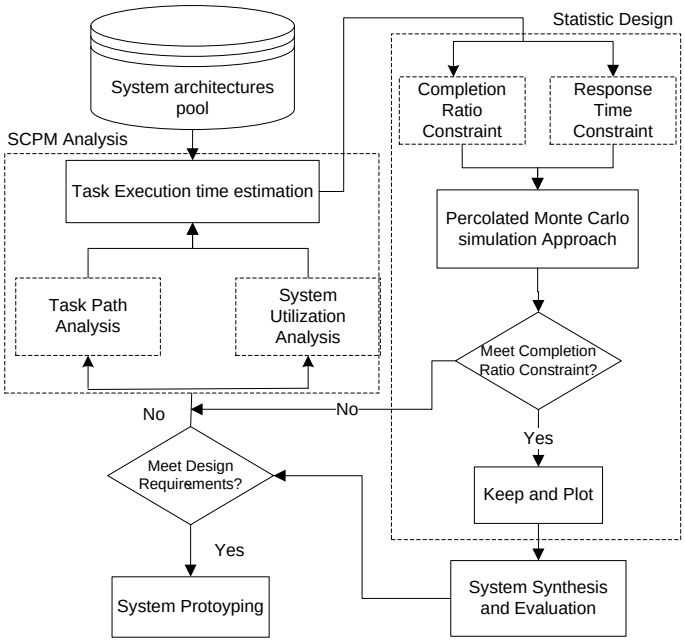


Figure 2. Statistical Design Methodology

In this paper, we will restrict the discussion to one specific part of the statistical design approach – optimization; i.e. we are assuming the existence of a high quality, fast-running simulation model (or meta-model of a simulation model).

4 ESTIMATING THE PROBABILISTIC TIMING PERFORMANCE

In order to determine whether a given system implementation can meet the desired completion ratio constraint, we need to estimate the system's probabilistic timing performance. Specifically, we calculate the upper bound of the completion ratio that the system can achieve to help us in exploring the statistical design space.

We consider the task graph $G = (V, E)$ for a given application. Each vertex in the graph represents one task computation and directed edges represent the data dependencies between vertices. We adopt the assumption that the execution time of each vertex can be described by a discrete or continue probability density function (Hu, 2005). Specifically, for each vertex v_i, we associate with a finite set of possible execution time $\{t_{i1}, t_{i2}, \ldots, t_{iki}\}$ (under a reference system configuration) and the set of probabilities:

$$\left\{ p_{i1}, p_{i2}, \ldots p_{ik_i} \text{while} \sum_{i=1}^{k_i} p_{il} = 1 \right\} \quad (1)$$

that such execution time will occur at run-time. That is, the time vertex v_i requires with probability p_{ij},.

The completion time of the task graph G (or equivalently the given application) under a fixed execution order $< v_1, v_2 \ldots, v_n >$ is the sum of each vertex's run-time execution time e_i:

$$A(< v_1, v_2, \ldots v_n >) = \sum_{i=1}^{n} e_i \quad (2)$$

The deadline constraint M specifies the maximum time allowed to complete the application. The application (or its task graph) will be executed periodically with its deadline M as the period. We say that an iteration is successfully completed if A (< v1, v2 …, vn >) ⩽M. The performance requirement is measured by a real-valued completion ratio Q0∈[0, 1], which is the minimum ratio of completions that the system has to maintain over a sufficiently large number of iterations. Let k be the number of successfully completed iterations over a total of $N \gg 1$ iterations; the actual completion ratio can be denoted by:

$$Q = \frac{k}{N} \quad (3)$$

We say that the completion ratio constraint is achievable if $Q \geqslant Q_0$.

For a given system configuration, let t_{ij} be the time to execute task v_i that requires an execution time t_{ij} under the reference configuration. We have a completion if the completion time is less than the deadline, that is,

$$\sum_{i=1}^{n} t'_{ij} \leq M \quad (4)$$

The probability that this occurs is $\prod_{i=1}^{n} p_{ij}$. Therefore, we have

Theorem 1. *The maximum achievable completion ratio is given by:*

$$Q_{max} = \sum_{\sum_{i=1}^{n} t_{ij} \leq M} \prod_{i=1}^{n} p_{ij} \quad (5)$$

where the sum is taken over the execution time combinations that meet the deadline constraint M and the product computes the probability each such combination happens.

This is similar to the state-based feasibility probability defined in (7). Q_{max} helps us to quickly explore the statistical design space. Specifically, if $Q_{max} < Q_0$, which means that the completion ratio requirement is not achievable under the current system, we can make the early and correct decision to reconfigure the hardware or optimize the software implementation rather than further investigating the current system configuration.

The drawback of this estimation is that Equation (1) is computationally expensive, particularly when there are many tasks and each task has multiple execution times. For example, a task graph with 50 vertices and each vertex has only the best, average, and worst case execution time yields 350 different execution time combinations! If each task's execution time can be described by a discrete probability density function, the following heuristic method (5) can be applied.

Assuming that the task's execution times are ordered such that $t_{i1} < t_{i2} < \ldots < t_{iki}$, we define the prefix sum of the occurrence probability:

$$P_{il} = \sum_{j=1}^{l} p_{ij} \tag{6}$$

which measures the probability that the computation at vertex v_i is not longer than t_{il}. If we allocate time t_{ili} to task v_i and drop the iteration if its actual execution time is longer, then we achieve a completion ratio:

$$Q = \sum_{i=1}^{n} P_{il_i} = \prod_{i=1}^{n} \sum_{j=1}^{l_i} p_{ij} \tag{7}$$

We use a greedy approach to estimate whether completion ratio Q_0 can be achieved within deadline M. First, we assign each vertex its WCET. This yields $Q = 1$ but the completion time:

$$\sum_{i=1}^{n} t_{ik_i} \tag{8}$$

will most likely exceed the deadline constraint. From Equation (3), if we cut the time slot of vertex v_i from t_{ili} to $ti(li-1)$, the completion ratio will be reduced by the factor of:

$$\frac{P_{i(l_i-1)}}{P_{il_i}} \tag{9}$$

We iteratively cut the time slot of vertex v_j that yields the largest

$$\left(t_{il_j} - t_{j(l_j-1)}\right) \times \frac{P_{j(l_j-1)}}{P_{jl_j}} \tag{10}$$

as long as it gives a completion ratio larger than Q_0. This greedy selection approach frees more assigned time slots at the minimum level of the completion ratio reduction. When we cannot reduce the completion ratio any further and the total assigned time

$$\sum_{i=1}^{n} t_{il_i} \tag{11}$$

is still larger than the deadline M, our heuristic concludes that the completion ratio cannot be guaranteed even though Q_{max} may still be larger than Q_0.

The drawback of the heuristic method is that it is difficult to understand and not visual. Due to the importance of determining whether the required Q_0 is achievable in designing fast statistical design space exploration techniques, we have developed the following Random Critical Path simulation approach. When each task's execution time is described by discrete or continue probability density function, this method can be applied.

5 RANDOM CRITICAL PATH SIMULATION APPROACH

The Critical Path Method (CPM) was invented by DuPont in 1957 for providing directions on how to control the schedule of some specific tasks. This method calculates the minimum completion time for a project along with the possible start and finish times for the project activities. The critical path is the one from the start of project to the finish where each slack time is zero. It is also the minimum time required to complete a project. Any delays along the critical path imply that additional time is required to complete the project. CPM has proved very valuable in evaluating project performance and identifying bottlenecks. Thus, CPM is a vital tool for the planning and control of complex tasks. Determining the critical path has been the correct way to manage task schedules.

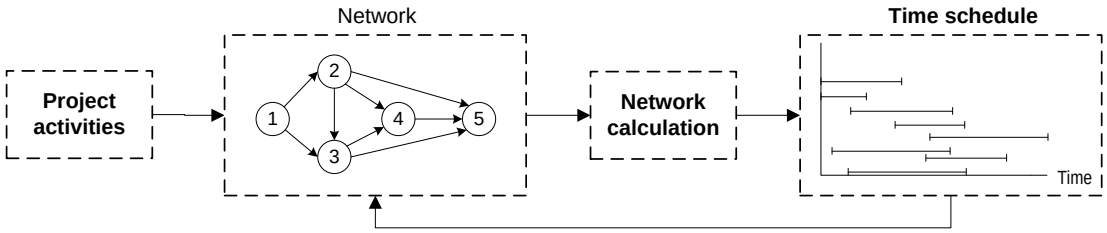


Figure 3. Critical Path Method (CPM) Technique

Figure 3 represents the steps of the CPM technique. Formally, critical path scheduling assumes that a project has been divided into activities of fixed duration and well defined predecessor relationships. A predecessor relationship implies that one activity must come before another in the schedule. Most critical path scheduling algorithms impose restrictions on the generality of the activity relationships or network geometries that are used. In essence, these restrictions imply that the construction plan can be represented by a network plan in which activities appear as branches in a network. The predecessor relationships among the activities are represented by a network as in Figure 3. Nodes are numbered, and no two nodes can have the same number or designation. Two nodes are introduced to represent the start and completion of the project.

The goal of CPM is to estimate the total duration of a project by determining the End Last Finish Time (ELFT). This is done by network calculation. Considering just the activities that do not have slack, the minimum finishing time of the project is computed. The activities without slack trace the critical path of the project schedule. The purpose of the critical path method (CPM) is to identify critical activities so that resources may be concentrated on these activities in order to reduce project length time. There may be more than one critical path among all the project activities, and completion of the entire project could be delayed by delaying activities along any of the critical paths. Therefore, the time schedule of the project is developed. However, in actuality, the project may not proceed as planned, as some of the activities may be expedited or delayed. In this case, the schedule must be revised to reflect the realities on the ground.

With the background provided above, we can formulate the critical path scheduling mathematically. We present an algorithm or set of instructions for critical path scheduling assuming an activity-on-branch project network. We also assume that all predecessors are of a finish-to-start nature so that a succeeding activity cannot start until the completion of a preceding activity.

Suppose that our project network has n nodes, the initial event being 1 and the last event being n. Let the time at which node events occur be x_1, x_2,...., x_n, respectively. The start of the project at x_1 is defined as time 1. Nodal event times must be consistent with activity durations so that an activity's successor node event time must be larger than an activity's predecessor node event time plus its duration. For an activity defined as starting from event *i* and ending at event *j*, this relationship can be expressed as the inequality constraint, $x_j \geq x_i + D_{ij}$, where D_{ij} is the duration of activity *(i,j)*. This same expression can be written for every activity and must hold true in any feasible schedule. Mathematically, then, the critical path scheduling problem is to minimize the time of project completion (x_n) subject to the constraints that each node completion event cannot occur until each of the predecessor activities have been completed.
Minimize

$$z = x_n$$

Subject to

$$x_1 = 0$$

$$x_j - x_i - D_{ij} \geq 0 \ for\ each\ activtiy \quad (i,j) \tag{12}$$

This is a linear programming problem because the objective value to be minimized and each of the constraints are linear equations. Rather than solving the critical path scheduling problem with a linear programming algorithm (such as the Simplex method), more efficient techniques are available that take advantage of the network structure of the problem. These solution methods are very efficient with respect to the required computations so that very large networks can be treated, even with personal computers. These methods also give some very useful information about possible activity schedules. The programs can compute the earliest and latest possible starting times for each activity that are consistent with completing the project in the shortest possible time. This calculation is of particular interest for activities not on the critical path (or paths) because these activities might be slightly delayed or re-scheduled over time by the manager without delaying the entire project.

An efficient solution process for critical path scheduling based upon node labeling is shown in Table 1. Three algorithms appear in the table. Define:

- $\square j$ - The Earliest Occurrence Possible Time of Event
- Δ_j - The Latest Occurrence Possible Time of Event
- D_{ij} - Duration of activity (i,j)

The *event numbering algorithm* numbers the nodes (or events) of the project such that the beginning event has a lower number than the ending event for each activity. Technically, this algorithm accomplishes a "topological sort" of the activities. The project start node is given number 1. As long as the project activities fulfill the conditions for an activity-on-branch network, this type of numbering system is always possible. The *earliest event time algorithm* computes the earliest possible time, E(*i*), at which each event, *i*, in the network can occur.

Table 1. Critical Path Scheduling Algorithms (Activity-on-Branch Representation)

Event Numbering Algorithm

Step 1: Give the starting event number *1*.
Step 2: Give the next number to any unnumbered event whose predecessor events are each already numbered.
Repeat Step 2 until all events are numbered.

Earliest Event Time Algorithm

Step 1: Let $\square_1 = 0$.

Step 2: For j = 2,3,...,n (where n is the last event), let

$$\square_j = \text{maximum } \{\square_i + D_{ij}\}$$

where the maximum is computed over all activities (*i*,*j*) that have j as the ending event.

Latest Event Time Algorithm

Step 1: Let Δ_n equal the required completion time of the project.

Note: Δ_n must equal or exceed $\square_n$.

Step 2: For i = n-1, n-2, ... , 1, let

$$\Delta_i = \text{minimum } \{\Delta_j - D_{ij}\}$$

where the minimum is computed over all activities (*i*,*j*) that have i as the starting event.

Earliest event times are computed as the maximum of the earliest start times plus activity durations for each of the activities immediately preceding an event. The earliest start time for each activity *(i,j)* is equal to the earliest possible time for the preceding event $\square_i$:

$$ES(i,j) = \square_i \tag{13}$$

The earliest finish time of each activity (*i*,*j*) can be calculated by:

$$EF(i,j) = \square_i + D_{ij} \tag{14}$$

Activities are identified in this algorithm by the predecessor node (or event) *i* and the successor node *j*. The algorithm simply requires that each event in the network should be examined in turn beginning with the project start (node 1).

The *latest event time algorithm* computes the latest possible time, Δ_j, at which each event *j* in the network can occur, given the desired completion time of the project, Δ_n for the last event *n*. Usually, the desired completion time will be equal to the earliest possible completion time so that $\square_n = \Delta_n$ for the final node *n*. The procedure for finding the latest event time is analogous to that for the earliest event time except that the procedure begins with the final event and works backwards through the project activities. Thus, the earliest event time algorithm is often called a *forward pass* through the network, whereas the latest event time algorithm is the *backward pass* through the network. The latest finish time consistent with completion of the project in the desired time frame of Δ_n for each activity *(i,j)* is equal to the latest possible time Δ_j for the succeeding event:

$$LF(i,j) = \Delta_j \tag{15}$$

The latest start time of each activity *(i,j)* can be calculated by:

$$LS(i,j) = \Delta_j - D_{ij} \tag{16}$$

The earliest start and latest finish times for each event are useful pieces of information in developing a project schedule. Events which have equal earliest and latest times, *E(i) = L(i)*, lie on the critical path or paths. An activity *(i,j)* is a critical activity if it satisfies all of the following conditions:

$$\square_i = \Delta_i$$

$$\square_j = \Delta_j$$

$$\square_i + D_{ij} = \Delta_j \tag{17}$$

Hence, activities between critical events are also on a critical path as long as the activity's earliest start time equals its latest start time, *ES(i,j) = LS(i,j)*. To avoid delaying the project, all the activities on a critical path should begin as soon as possible, so each critical activity *(i, j)* must be scheduled to begin at the earliest possible start time, $\square_i$.

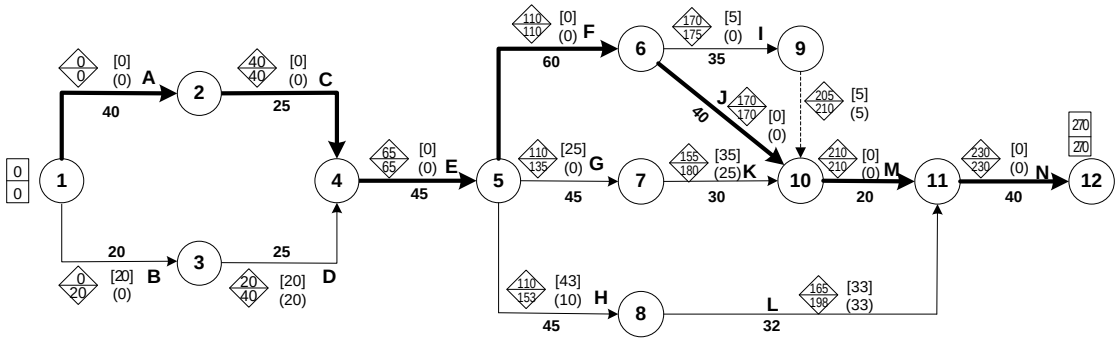


Figure 4. Single Point Project Scheduling

Suppose a task of fourteen relevant activities, described in Figure 4. Applying this approach to the example,

the End Last Finish Time (ELFT) for the project is 270 seconds, and the activities that lie on a critical path are: A, C, E, F, J, M, and N. That means if a delay occurs in any of these activities, the total completion time will be over 270 seconds.

However, most of the time using CPM does not guarantee that the task will finish on time because the successful implementation of CPM requires defining the relevant activities, including estimates of their durations and the sequence of the activities. The output is usually represented by a network diagram in which the critical path is identified. However, in practical situations this requirement is usually hard to fulfill as many activities will be executed for the first time. Hence, there is always uncertainty about the time duration of activities in the network planning. In this case, more often than not, there is a vague idea about activity durations that then must be estimated subjectively. Some soft real-time system designers use the single-point estimate (most likely scenario) for each task in the tasks, and others use three scenarios: pessimistic, optimistic, and most likely. Each method is problematic: the task will not reach the goals on time because the designer does not know anything about the probability of each scenario.

In this section, a simulation method is excogitated, allowing probabilities with the results to be associated, to determine all scenarios in which it is possible to trace the random critical paths. Most of the time, results from simulation can provide more critical paths than a scenario approach. Furthermore, the activities whose uncertainties have the greatest influence on the total completion time are not necessarily the same activities defining the most critical paths. That means one particular critical path is important depending on how probable it is. Obviously, using a single-point estimate or scenario analysis could be very inappropriate. With simulation, more realistic time forecasts can be guaranteed.

Scenario analysis determines the best-case, most likely case, and worst-case scenarios regarding the duration of each relevant activity on a project. Applying CPM to each scenario allows a project manager to identify a range for the completion time but not the associated probabilities of occurrence. Table 2 shows the scenarios for the example above. The completion time range is 198 to 395 seconds.

Table 2. Scenario Analysis for Project Scheduling

Activities		Durations			Activities		Durations		
ID	Description	OTD	LTD	PTD	ID	Description	OTD	LTD	PTD
A	Task A	30	40	60	H	Task H	40	45	60
B	Task B	15	20	30	I	Task I	30	35	50
C	Task C	15	25	40	J	Task J	30	40	60
D	Task D	20	25	40	K	Task K	25	30	60
E	Task E	40	45	65	L	Task L	28	32	42
F	Task F	30	60	95	M	Task M	18	20	25
G	Task G	35	45	60	N	Task N	35	40	50
						Total Time	198	270	395

OTD= optimistic time duration, PTD= pessimistic time duration, LTD= most likely time duration

It is easy to think of adopting “the average” or “the most important value” for a deterministic analysis because of the tendency to overestimate their meaning. In fact, the problem is the sequence observed in many natural ways. That sequence creates another dimension of analysis because “the average” or “the most important value” of one separate event is not independent and has meaning in context to other events. For example, for five sequential events with each having a 50% probability of occurrence, the probability of the sequence is: power (0.5, 5) = 3.1%. That means that there is 97% confidence that the five sequential events will not occur. This is a critical situation for managing schedule in tasks because of the sequence required for completion.

The only way to capture the uncertainty in the estimation of the total completion time is by creating a dynamic model that simulates not just a few scenarios but all possible scenarios. The static model must be transformed into a simulation model. The parameters for defining the model are:

- Assumptions: the duration of each activity is represented by a Gamma distribution

A skewed distribution represents the duration of an activity better than a triangular distribution, because it includes the upside values.

- Forecast: the total completion time
- Path Control Box: shows activities in critical path with bold font

A Path Control Box (PCB) has been created into the model to show us how the critical path is not only one critical path (see Figures 5 and 6). In fact, there are many potential critical paths depending on the conditions surrounding the time variables necessary to conclude the project.

STOCHASTIC PATH					
Attributes					
Unit of time	second				
Precision	0	decimals			
Path Control Box					
	Task A	**Task D**	Task G	Task J	**Task M**
	Task B	**Task E**	Task H	Task K	**Task N**
	Task C	**Task F**	**Task I**	Task L	
Results					
Total time	**280**	seconds			

Figure 5. Stochastic Critical Paths A

STOCHASTIC PATH					
Attributes					
Unit of time	second				
Precision	0	decimals			
Path Control Box					
	Task A	Task D	Task G	**Task J**	**Task M**
	Task B	**Task E**	Task H	Task K	**Task N**
	Task C	**Task F**	**Task I**	Task L	
Results					
Total time	306	seconds			

Figure 6. Stochastic Critical Paths B

5.1 Simulation Result

Figure 7 shows the forecast chart for the total completion time for the project, after 10,000 trials. The probability of not exceeding the base case time is approximately 30%. From this we can estimate the system timing performance to check whether it is feasible for the current system configuration to achieve the desired performance. If not, we take the sensitivity analysis to find out which part of system can be improved.

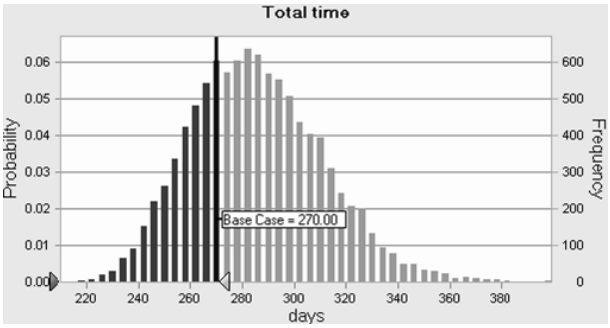
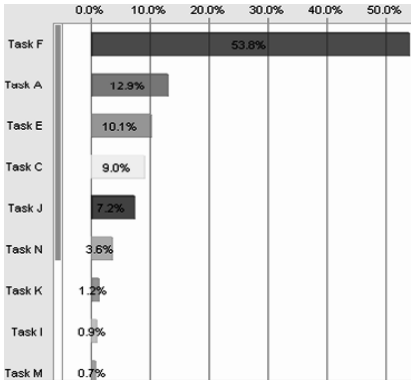
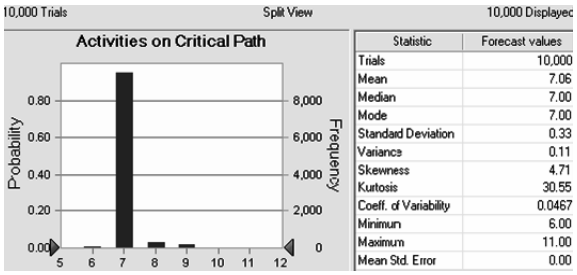


Figure 7. Total Completion Time Forecast

5.2 Sensitivity Analysis

The inputs whose variability contributes the most for the dispersion in the total completion time are Task F, Task A, and Task E. They contribute over 75% to the variance on the forecast (Figure 8).





Statistic	Forecast values
Trials	10,000
Mean	7.06
Median	7.00
Mode	7.00
Standard Deviation	0.33
Variance	0.11
Skewness	4.71
Kurtosis	30.55
Coeff. of Variability	0.0467
Minimum	6.00
Maximum	11.00
Mean Std. Error	0.00

Figure 8. Sensitivity Of The Total Completion Time

Figure 9. Number Of Activities On A Critical Path

Figure 9 shows the number of activities lying on a critical path for the project example, and Figure 10 displays the sensitivity of the number of activities on a critical path.

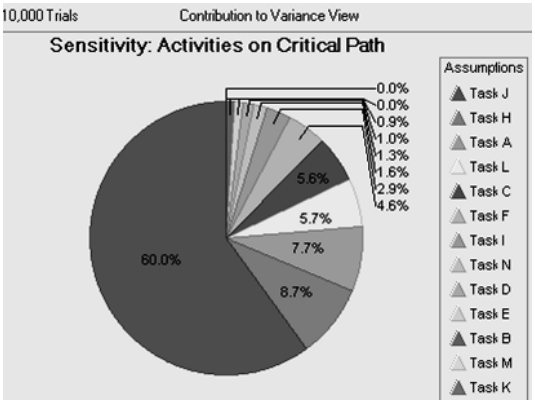


Figure 10. Sensitivity Of The Number Of Activities On A Critical Path

If we want to improve the system timing performance to achieve the desired performance, these tasks whose variability contributes the most to dispersion in the total completion time should be adjusted by changing the hardware configuration and/or applying software optimization techniques and updating the software profiling results that will be used in the next round of system timing performance estimation.

6 PERCOLATED STOCHASTIC SIMULATION APPROACH

When $Q_{max} \geq Q_0$, it becomes theoretically possible to deliver a probabilistic performance guarantee (in terms of completion ratio) with the current system configuration. The resource management phase in our design space exploration aims to reduce the design cost. It includes: 1) determining the minimum system resource required to provide the probabilistic performance guarantees and 2) developing on-line scheduling algorithms to guide the system to achieve such guarantees at run time with the determined minimum resource.

The approach below is what is called a Percolated Stochastic approach. The approach is clearly easy to use, fast, and visual. The initial steps of the approach follow the standard application of Monte Carlo analysis. The range of the input variables (the *Xs*) to be explored are determined and fed as input through the simulation model to obtain the distribution output of the predicted performance variables (the *Ys*). Then post-processing of the output occurs through a series of percolates to ensure constraints are met, hence the name Percolated Stochastic.

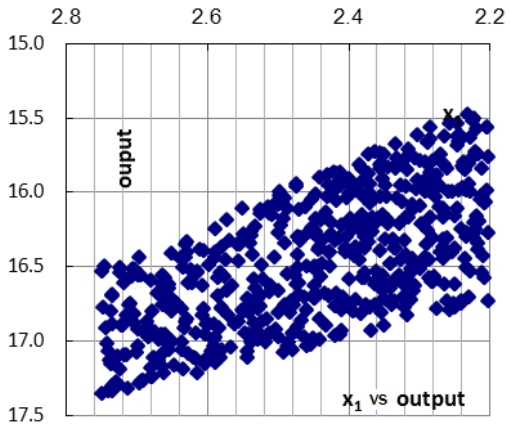


Figure11-a. X_1 vs output

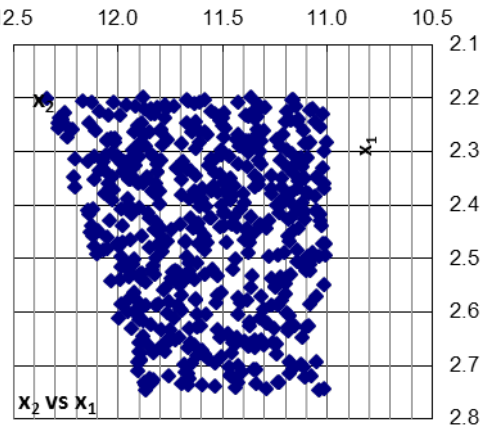


Figure11-b. X_2 vs X_1

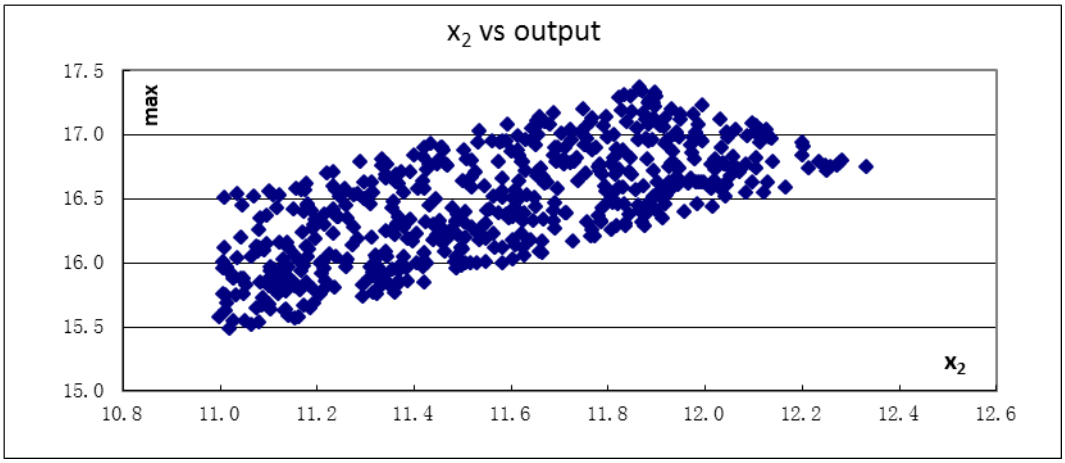


Figure 12. Plot of Output vs X_2

The approach will be demonstrated through the use of a simple example: Suppose we wish to maximize $f(X_1, X_2) = 2(X_1) + X_2$, subject to the constraints

(1) $X_1 \leq 4.5$
(2) $10(X_1) + 5(X_2) \leq 56$
(3) $2(X_1) + 0.5(X_2) \leq 17.5$
(4) $2(X_1) + 1.5(X_2) \leq 44.5$
(5) $-0.25(X_2)^2 + 8.5(X_2) + 2(X_1) \leq 64.$

The starting distributions for X_1 and X_2 will be assumed to be
$X_1 \sim$ unif (0,5)
$X_2 \sim$ unif (0,10)

Post-processing of the output occurs through a series of percolate to ensure constraints are met. For example, if the percolate for constraint 1 includes values in the range of [-∞, 4.5], other values are discarded. All the data meeting the constraints will be kept and plotted in Figures 11 and 12.

The exciting result here is that one is able to visualize sensitivity to variation in the Xs – you can ensure that you are not close to a cliff and can determine how much to "back off" the deterministic optimum to account for expected variation.

7 CONCLUSION AND FURTHER STUDY

The Percolated Stochastic approach can be used in a variety of applications that include optimization, robustness studies, and design space exploration. However, the corresponding "statistical design space" becomes larger than the above mentioned pessimistic design space because it includes designs that fail some iterations while still meeting the desired completion ratio requirement statistically. This increases the design complexity and makes early design space exploration difficult. The "statistical design" will thrive only when designers can quickly explore the huge statistical design space.
The next big challenge will be problems with a large number of Xs and/or Ys (>10). The key need is for improved visualization and data mining tools that will allow designers to rapidly explore design spaces while understanding sensitivity to variation.

8 REFERENCES

Benini, L., Bogliolo, A., & De Micheli, G. (2000)A survey of design techniques for system-level dynamic power management. *IEEE Trans. on VLSI Systems*, 8(3), 299‑316.

Gustafsson, J. (2002) Worst case execution time analysis of object-oriented programs, Object-Oriented Real-Time Dependable Systems, 2002. (WORDS 2002). *Proceedings of the Seventh International Workshop on 2002*, 71-76, Digital Object Identifier 10.1109/WORDS.2002.1000038

Hu, X., Zhou, T., & Sha, H.-M. (2001) Estimating probabilistic timing performance for real-time embedded systems. *IEEE Trans. on VLSI systems*, 9(6):833‑844.

Hua, S., Qu, G., & Bhattacharyya, S. (2003) Energy reduction techniques for multimedia applications with tolerance to deadline misses. *40th ACM/IEEE Design Automation Conference*.

Kalavade, A. & Moghe, P. (1998) A tool for performance estimation of networked embedded end-systems. Proc. *Design Automation Conference*, 257‑262.

Kligerman, E., &Stoyenko, A. (1986) Real-time euclid: A language for reliable real-time systems. *IEEE Transactions on Software Engineering* SE-12(9), 941–949.

Mok, A. K., Amerasinghe, P., Chen, M., &Tantisirivat, K. (1989) Evaluating tight execution time bounds of programs by annotations. Proc. 6*th IEEE Workshop on Real-Time Operating Systems and Software, Pittsburgh, PA, USA*, 74–80.

Park, C.Y. & Shaw, A. (1990) Experiments with a Program Timing Tool Based on a Source-Level Timing Schema. In Proc. *11th IEEE Real-Time Systems Symposium(RTSS'90)*, 72–81.

Puschner, P. (2000) Guest Editorial: A Review of Worst-Case Execution-Time Analysis[J]. *Real-Time Systems* Vol.18, No.2/3.

Puschner, P., & Koza, C. (1989) Calculating the maximum execution time of real-time programs. *Real-Time Systems* 1(2), 159–176.

Shaw, A. C. (1989) Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering* SE-15(7), 875–889.

Taha, H. (2006) *Operations Research: An Introduction (8th Edition)*, Prentice Hall.

(Article history: Received 8 January 2009, Accepted 9 February 2010, Available online 28 February 2010)