Combining Formal Verification with Observed System Execution Behavior to Tune System Parameters *

Minyoung Kim[1], Mark-Oliver Stehr[2], Carolyn Talcott[2]
Nikil Dutt[1], Nalini Venkatasubramanian[1]

[1] University of California, Irvine, USA
{minyounk,dutt,nalini}@ics.uci.edu
[2] SRI International, USA
{stehr,clt}@csl.sri.com

Abstract. Resource limited DRE (Distributed Real-time Embedded) systems can benefit greatly from dynamic adaptation of system parameters. We propose a novel approach that employs iterative tuning using light-weight, on-the-fly formal verification with feedback for dynamic adaptation. One objective of this approach is to enable system designers to analyze designs in order to study design tradeoffs across multiple layers (for example, application, middleware, operating system) and predict the possible property violations as the system evolves dynamically over time. Specifically, an executable formal specification is developed for each layer of the distributed system under consideration. The formal specification is then analyzed using statistical model checking and statistical quantitative analysis, to determine the impact of various resource management policies for achieving desired end-to-end timing/QoS properties. Finally, integration of formal analysis with dynamic behavior from system execution will result in a feedback loop that enables model refinement and further optimization of policies and parameters. We demonstrate the applicability of this approach to the adaptive provisioning of resource-limited distributed real-time systems using a multi-mode multimedia case study.

Key words: Iterative System Tuning, Formal Modeling, Statistical Formal Methods, System Realization, Cross-layer Timing/QoS/resource Provisioning for Distributed Systems

1 Introduction

Next generation mobile embedded applications are highly networked, and involve end-to-end interactions among multiple layers (application, middleware, network, OS, hardware architecture) in a distributed environment. Timing plays a critical role in QoS-aware system design for a large class of such distributed applications. Firstly, timing can impact application semantics. Multimedia applications have soft real-time needs often stated using parameters such as jitter,

* This work was partially supported by NSF award CNS-0615438 and CNS-0615436.

synchronization skews and bounded end-to-end delays. Secondly, there are several sources of unpredictability and timing violations in a distributed network; this introduces nondeterminism in timing. The ability to compensate on-the-fly for timing violations at different levels is of paramount importance. Thirdly, several system level optimizations for effective utilization of distributed resources can interfere with the timing properties of executing applications. Finally, many applications have flexible QoS needs that dictate how tolerant they are to delays and errors — the lack of stringent timing needs can be exploited for better end-to-end resource utilization.

The dual goals of ensuring adequate application QoS (expressed as timeliness, reliability, and accuracy) and optimizing resource utilization at all levels of the system presents significant challenges. In this context, our preliminary study [1] demonstrated the need for integration of formal methods with experimentally based cross-layer optimization methods [2,3]. Systematic analysis based on well-defined models ensures that corner-cases are covered and allows bounds for critical performance parameters to be determined. Recently, we proposed probabilistic formal methods to provide analysis of given cross-layered optimization policies with quantifiable confidence [4].

To leverage these prior efforts, we propose an iterative tuning approach for DRE systems that couples two important facets:

1. a light-weight, on-the-fly formal verification system that can be used dynamically to evaluate the impact of different resource management policies for achieving end-to-end timing/QoS properties, and
2. a system realization that enables feedback of additional information on dynamic system execution behaviors to enhance our light-weight formal modeling and analysis.

The integration of formal analysis combined with observed system execution behavior permits better analysis of both cross-layer and end-to-end timing/QoS properties for highly distributed systems that employ resource constrained devices.

This paper contains the following contributions:

– We present a generic framework to address iterative system tuning of distributed embedded systems by integrating two synergistic approaches: on-the-fly formal verification and learning from system realization. The light-weight formal verification provides degrees of confidence in the feasible solutions satisfying multidimensional constraints. System realization enables dynamic adaptation by refining the model of the system and the environment.
– Our work is validated and tested in the context of distributed mobile multimedia applications that have wide consumer applicability, execute in highly dynamic changing environments and present interesting opportunities for tradeoff analysis and enforcement.

The rest of this paper is organized as follows: we start by motivating our approach. Next, we present the overview of our framework, followed by a description of our case study (multi-mode multimedia communication system). In

Section 5, we explain our approach in depth. Specifically, we describe the modeling and specification of our case study. We then introduce our probabilistic formal analysis, followed by discussion of the feedback loop with our system realization. Our implementation and experimental results show the applicability of our framework to the distributed real-time multimedia communication domain. The last section summarizes our approach and discusses future research directions.

2 Supporting Adaptation under Timing Constraints

Timing can affect, and be affected by several system and resource parameters such as storage/buffer, CPU, network topology and communication characteristics as follows.

- Power/Timing Tradeoffs: Power optimization strategies have a significant implication on the timing properties at different levels of the system. For instance, dynamic voltage scaling techniques within the operating system dictate slowdown of the CPU while lengthening the execution time, which results in possible deadline misses.
- Quality/Timing Tradeoffs: A change in the quality of communicated information represents changes in the execution time, communication time and buffering needs; for instance, lower quality video requires less decoding effort and time.
- Buffering/Timing Tradeoffs: The presence of a buffer in the datapath of the streaming information can relax the timing needs at various layers. For instance, larger buffers at the end device imply that timing constraints on receiving new data can be less stringent at the cost of memory usage. This in turn can translate into longer sleep durations (low-power operation) for power-intensive communication components. Buffering can also be used to compensate for noise, and hence delays, in the communication networks.
- Error Resilience/Timing Tradeoffs: Error resilience needs are often posed by applications to dictate fidelity requirements. In streaming applications, errors are often introduced in the communication process due to the presence of the network noise. Avoiding these errors typically involves strategies that retransmit information, encode and check for integrity of data transfer — all of which have significant implications on the overall timing behavior of the system.

This problem will be much more complex if we consider that the system and environment may keep evolving, requiring dynamic adaptation. Given a current configuration and a set of changes (e.g., new application/task; parameters for existing tasks such as framerate/resolution/synchronization; device residual power level; network delay/jitter, etc.), we need to perform dynamic adaptation (re-determine the policy, followed by bound/sensitivity analysis on the impact of the selected policy) since all the changes are critically related to timing.

In this context, we propose a unified framework for iterative and proactive system tuning to support adaptations. Initially, our framework performs prop-

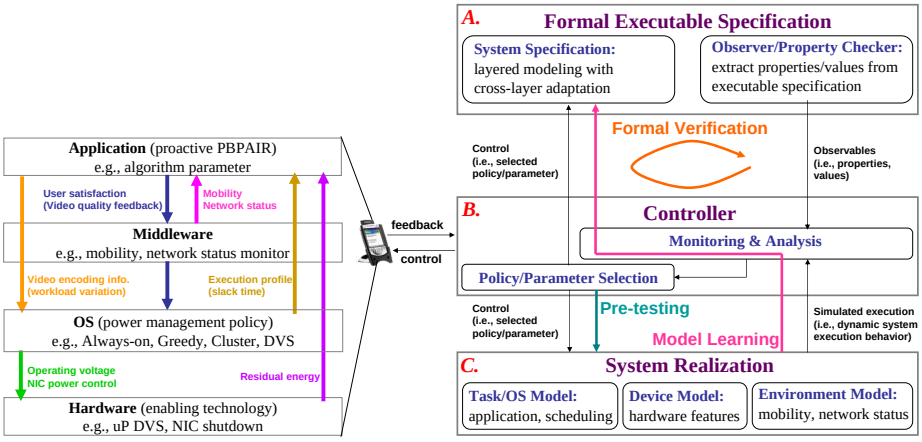


Fig. 1. The Iterative System Tuning

erty checking and quantitative analysis among candidate policy/parameter settings via formal executable specifications followed by probabilistic formal analysis. In our framework, the *iterative* tuning allows model refinement from up-to-date and continuous observations of system execution behavior. Furthermore, this can be used to improve adaptation by verifying given system properties or by relaxing constraints. A priori information, as forecasted by the system realization, enables *proactive* control.

3 A Framework for Iterative/Proactive System Tuning

Figure 1 presents the overall flow of our approach. We take three major steps: (1) formal modeling, (2) probabilistic formal analysis, and (3) model refinement and proactive control. In Figure 1, the *Box A* represents the formal modeling. The core of our formal modeling approach is to develop formal executable models of system components at each layer of interest. These models express functionality, timing, and other resource considerations at the appropriate level of detail and using appropriate interaction mechanisms (clock ticks, synchronous or asynchronous messages). Models of different layers are analyzed in isolation and composed to form cross-layer specifications. The use of Maude as a reasoning tool will be discussed in more detail in Section 5.1.

One advantage of formal executable models is that they can be subjected to a wide range of formal analysis, including: single execution scenarios, search for executions leading to states of interest, and model-checking to understand properties of execution paths. The *Box B* in Figure 1 shows the evaluation phase of given specifications to generate statistics of monitored properties and values. Specifically, we have developed new analysis techniques (statistical model-checking and statistical quantitative analysis) that combine statistical and formal methods, and applied them to a case study treating the videophone mode of a multi-mode multimedia terminal [4]. Section 5.2 introduces a brief review of our probabilistic formal methods.

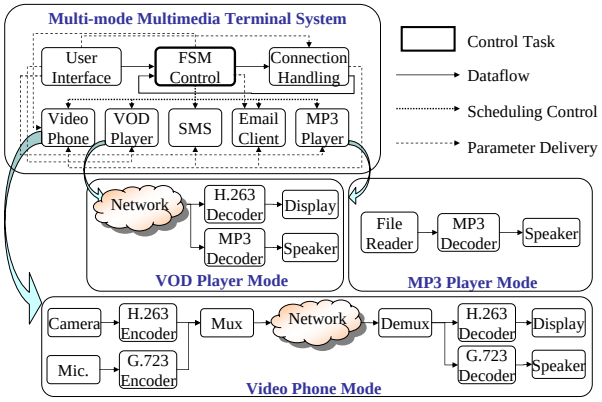


Fig. 2. Case Study: MMMT (Multi-Mode Multimedia Terminal)

Using such models and analysis, tools can be developed to achieve adaptive refinement of an end-to-end system specification into appropriate policy/parameter settings. We propose an *iterative* tuning strategy that combines formal methods (verification) with dynamic system execution behavior (obtained by either simulation or implementation). The execution behavior from system realization (*Box C* in Figure 1) is fed back into the formal modeling to refine the executable specification (*model refinement*). In addition, we can assure the quality of a new policy/parameter constructed by the controller. In Figure 1, *Pre-testing* on a system realization can lead to improvements because typically the formal model can not cover all the possible implementation details of a real system (*proactive* control). We will explain iterative tuning and proactive control in Section 5.2 and 5.3 in more detail.

4 Case Study: Multi-mode Multimedia Terminal

Although we intend our approach to be widely applicable, we begin by developing and evaluating formal specification models in the context of distributed multimedia applications.

Figure 2 shows an example of a multi-mode multimedia terminal (MMMT) system [5] that we are using as a research vehicle. The figure depicts a hierarchical composition of tasks within the MMMT system. At the top level, three types of hierarchical tasks are defined to specify each mode of operation: soft real-time (a videophone, a VoD player, an MP3 player), event-driven (email client), and time-critical emergency messaging (SMS-Short Message Services). Three other tasks are also specified at the top level for user interface, connection handling, and task execution control. In addition, each mode of operation consists of multiple tasks as shown in the figure. This type of application requires frequent task set changes based on user input and/or node/network conditions (e.g., residual power level, packet drop rate, noise level, etc.). As an example, a high-end videophone mode would be able to better meet its timing constraints at maximum CPU performance while receiving packets via a reliable channel. However, if the

residual power level dropped or packet loss rate increased significantly, then we might need to save energy by reducing QoS or suspending some tasks. A user also can explicitly change modes and assign different priorities for each task/mode.

As you see from the layered view of a device in Figure 1, the resource management policies that are used in the different layers include: a specific video encoding/decoding algorithm at the application layer; network monitoring at the middleware layer; and DPM (Dynamic Power Management) and/or DVS (Dynamic Voltage Scaling)[3] at the OS layer [6]. Network traffic shaping and/or trans-coding at the middleware layer can be also utilized. Each policy has parameters that can be used to fine-tune the behavior. In addition, there are hardware parameters that can be set.

For instance, we consider *proactive* PBPAIR (Probability-Based Power-Aware Intra Refresh) [7] as an application layer policy. The *PBPAIR* scheme inserts intra-coding (i.e., coding without reference to any other frame) to enhance the robustness of the encoded bitstream at the cost of compression efficiency. Intra-coding improves error resilience, but it also contributes to reducing encoding energy consumption since it does not require motion estimation[4] (which is the most power consuming operation in a predictive video compression algorithm). The additional *proactive* feature means that we have a priori information on the user's mobility (e.g., current zone, speed and trajectory, etc.) and network situation (e.g., packet loss rate, delay, etc.) that later will be used for selection among policies and related parameter tuning before the user enters a new zone. If PBPAIR is selected as an application layer policy, then algorithm-specific parameters such as *Intra threshold* value must be chosen for appropriate execution. Note that the parameter selection at one layer affects other layers. For example, PBPAIR increases intra-coding by lowering the *Intra threshold* parameter when there is high network packet loss (monitored at middleware layer), which impacts the DVS decision at OS layer since the execution profile of the application is changed.

5 Iterative Tuning by Formal Verification Combined with System Realization

Our approach combines

– **Modeling, specification and reasoning about cross-layer and end-to-end properties:** We propose a novel approach based on concurrent rewriting logic to formally specify and reason about end-to-end timing/QoS issues across layers and study their inter-relationships.

[3] DPM puts a device into a low power/performance state to save energy when the device is not serving any request during a suitably long time-period determined by the shutdown and wake-up overhead of the device. DVS aims at saving energy by scaling down the supply voltage and frequency when the system is not fully loaded.

[4] In predictive coding, motion estimation eliminates the temporal redundancy due to high correlation between consecutive frames by examining the movement of objects in an image sequence to try to obtain vectors representing the estimated motion.

- **Design of policies and mechanisms for addressing tradeoffs based on the cross-layer analysis:** Our work examines the impact of various resource management techniques on end-to-end timing/QoS properties and enables informed selection of resource management policies along with rules for instantiation of parameters that derive the policies.
- **Model refinement and proactive control:** We enhance our light-weight formal modeling and analysis by integrating it with observations of system execution behavior to achieve adaptive reasoning and proactive control by providing more precise information on current execution and future state.

In the following subsections, we explain each component; formal executable specification (Section 5.1), controller (Section 5.2), and system realization (Section 5.3) of our proposed framework (Figure 1) in depth beginning with our modeling effort.

5.1 Modeling Effort

Our formal modeling approach utilizes Maude [8] to formally specify the environmental changes as well as the policies/parameter settings that can be made at each of these levels in isolation and for the combined layers. Maude is a specification language based on rewriting logic with supporting analysis tools. The Maude system has been used in the specification and analysis of a wide range of logics, languages, architectures and distributed systems [9,10].

Rewriting logic [11] is a simple logic well-suited for distributed system specification. The state space of a distributed system is formally specified as an algebraic data type by giving a set of sorts (types), operations, and equations. The dynamics of such a distributed system is then specified by rewrite rules of the form

$$t \rightarrow t' \ \ \textbf{if}\ c$$

where t, t' are terms (patterns) that describe the local, concurrent transitions possible in the system, and c is a condition constraining the application of the rule. Specifically, when a part of the distributed state matches the pattern t, and satisfies c, then this part can change to a new local state t'. Rewriting logic specifications are executable, as proofs in rewriting logic are carried out by applying rewrite rules which can also be viewed as steps of a computation.

The Maude system is based on a very efficient rewriting engine, supporting use of executable models as prototypes. It also provides the capability to search the state space reachable from some initial state by the application of rewrite rules. This can be used to find reachable states satisfying a user-defined property. The system also includes an efficient model-checker for checking properties expressed in linear temporal logic. The Maude system, its documentation, and related papers and applications are available from the Maude website `http://maude.cs.uiuc.edu`.

In the object-oriented specification style supported by Maude, the system state (configuration) is typically represented as a multiset of objects and messages. Passage of time is modeled by functions that update the configuration

```
*** Property checker
  op batteryExpires : Configuration → Bool .
  eq batteryExpires(< CPU : HW | residualEnergy : F, atts > C:Configuration)
     = (if (F ≤ 0.0) then true else false fi) .
*** Observer
  msg Obs : Bool → Msg .
  msg EnergyConsumption : Float → Msg .
  msg BatteryExpires : Bool → Msg .

  rl [cpuObs] :
    < CPU : HW | consumedEnergy : F, policy : P, atts >
    ⇒
    EnergyConsumption(F)
    BatteryExpires(batteryExpires(< CPU : HW | consumedEnergy : F, atts >)) .
```

Fig. 3. Maude Specification: Property Checker and Observer

appropriately, for example decrementing timers or decreasing remaining power. Rules can either be instantaneous or tick rules of the form

$$C \rightarrow delta(C, T)\ in\ time\ T\ \ \textbf{if}\ T \leq mte(C)$$

where C is a term representing the system configuration. This tick rule advances time non-deterministically, according to a chosen time sampling strategy, by a time T less than or equal to *mte(C)*, the maximal time allowed to elapse in one step, in configuration C, and alters the system state, C, using the function *delta*[5]. Both *delta* and *mte* are user-defined to capture how time passes in a particular model.

In Maude syntax, objects have the general form

$$< ObjectName : ClassName \mid Attribute_1 : Value_1, ..., Attribute_n : Value_n >$$

where *ObjectName* is an object identifier, *ClassName* is a class identifier, and each *Attribute : Value* pair specifies attribute identifier and its value.

At the end of each execution, we examine the final configuration of a Maude specification that has several objects and messages. From those objects and messages, we need to extract meaningful data — observables. Observables can be properties or values. For example, to check whether the battery expires or not at the end of the execution, we need to check the *residualEnergy* attribute in *CPU* object at hardware layer. If the value for the *residualEnergy* attribute is positive, then the battery is not empty. Otherwise, the *batteryExpires* property returns *true* meaning the system used up the battery. We encode the check of properties into the model so that the result contains *true* or *false* depending on whether a property holds or not. On the other hand, if we want to have the energy consumption rather than the answer for property hold, we can utilize the observer such as the one shown in Figure 3. The observer replaces each object with suitable messages that have data values for the observables. Furthermore, we use the Maude API, a foreign language interface to embed the Maude rewriting engine into larger applications, to extract observables from the Maude execution and to generate statistics of results.

[5] The idea of a tick rule is taken from Real-Time Maude [12].

5.2 Adaptation by Statistical Evaluation and Reinforcement

Once we extract observables from runs of a formal executable specification, the controller performs formal verification and pre-testing as illustrated in Figure 1. For verification purposes, we use *probabilistic formal methods*. The controller also interacts with a system realization to pre-test selected policies/parameters, and to obtain information on dynamic system behavior to improve the formal model. These two techniques are summarized below.

Probabilistic Formal Methods To evaluate feasible design points, we adapt and improve two statistical evaluation methods — statistical model checking and statistical quantitative analysis [4]. For statistical model checking, probabilistic properties such as "*Probability* that a system can survive with given residual energy in t time units is more than θ %" are examined. Those formulae are essentially a restricted version of Continuous Stochastic Logic (CSL) [13] without nesting. Indeed we found no need for nested formulae or an exact numerical solution for our application domain. Therefore, we use statistical model checking to verify such probabilistic properties, more precisely hypothesis testing based on Monte-Carlo simulation results.

For statistical quantitative analysis, we estimate the expected value of certain observables such as "*Average* energy consumption in t time units within confidence interval (δ) and error bound (α)". Statistical evaluation can be performed with a large quantity of data that follows a normal distribution, and hence allows the estimation of the expected value and our confidence. To determine the mathematical soundness of the approximation, we perform a Jarque-Bera (JB) normality test [14]. More importantly, we generate traces on demand to reduce the evaluation time since it is linearly proportional to the trace generation time (i.e., Monte-Carlo simulation time with a different seed). Detailed explanation on statistical theory background and our implementation can be found in [4].

Model Refinement Within our framework, there are at least two roles for feedback from observation of system execution behavior: it can be used to improve the model (to make it more accurately match the real environment), and it can also be used to directly improve the policy using optimization or learning methods. In this particular work, we only consider the former case, that is *model refinement* using the information from dynamic system behavior.

For instance, the formal specification initially models the execution times of the tasks as a normal (Gaussian) distribution with the average of $\frac{(BCET+WCET)}{2}$ and the boundary value of $3\times\delta$, where δ represents the standard deviation, based on profiled best case execution time (BCET) and worst case execution time (WCET) from sample runs. This model is refined in turn by replacing BCET and WCET with observations from dynamic system execution (either by system realization in 5.3 or real implementation), in order to more realistically reflect the actual executions characterizing the system in practice such as data dependent execution times. In our framework, the controller (written in Java) uses a Java/Maude foreign language interface to execute/update Maude specifications and to extract the results for analysis.

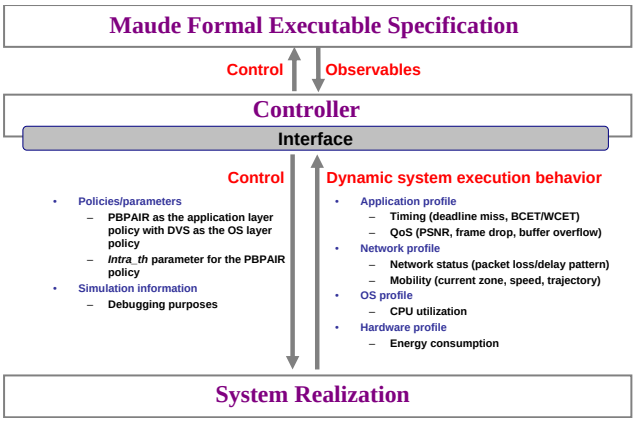


Fig. 4. Interactions between Controller and System Realization

5.3 System Realization

As we briefly mentioned in Section 3, the integration of formal analysis with a system realization (as illustrated in Figure 1) will result in a feedback loop that includes the formal models, simulation, and monitoring of running systems for analysis of the system behavior and for optimizing the choice of policies and parameters. Specifically, the system realization takes policies/parameters and returns the dynamic system execution behavior at each layer as seen in Figure 4. For instance, if the controller selects PBPAIR (with appropriate *Intra_Th* parameter) as the application layer policy and DVS as the OS layer policy, the system realization executes using the appropriate settings and reports profiled information such as consumed energy, timing/QoS aspects.

For this purpose, we define a collection of library routines and their arguments that can be used to implement a system realization [15]. At the application layer, we need to create a task set for a chosen mode. As an example, video phone mode has four tasks; video encoder/decoder and audio encoder/decoder, each with its own parameters (e.g., PBPAIR has *Intra_Th* parameter.). Besides, the input/output data structure is task specific. For instance, an H.263 encoder with PBPAIR policy takes the *Intra_Th* parameter, the network packet loss rate (precisely, this information will be provided as middleware layer input), and raw video sequences as inputs to generate a bitstream robustly encoded against network transmission errors. In addition, there are encoder QoS related parameters (e.g., quantization value, IP ratio, frame-rate, buffer size). As a result, application profile data such as QoS (PSNR(Peak Signal to Noise Ratio), frame drops) and timing (deadline misses, BCET, WCET) aspects should be reported.

The most two important observations from our system realization are timing (BCET, WCET) and network related information. The framework uses timing information to refine the model and network related information to generate *proactive* control. Therefore, in the experiments, we will demonstrate how the framework can achieve iterative system tuning for proactive control using those two pieces of feedback from system execution behavior.

6 Experimental Results

6.1 Evaluation Platform

Using formal executable specifications in Maude, we model *PBPAIR* as an application layer policy as well as two power management schemes — *Greedy* and *Cluster* — as OS layer policies. In the *Greedy* scheme, the power manager shuts down whenever the device is idle, while the *Cluster* scheme tries to aggregate idle periods to maximize energy efficiency. A subset of the MMMT system — video encoder and decoder for videophone mode — is modeled with the workload variation of a PBPAIR encoder [7] and an H.263 decoder [16]. The network zone information is assumed to be given and the hardware implementation is from [17,18].

For the system realization, we use the Simics [19] full system simulation platform, capable of simulating target systems that include real network connection and run operating systems and workloads. Specifically, we use the Simics model of a PowerPC-based Ebony card [17] with a PPC440GP processor [18] that boots Linux 2.4. The execution profile from the Simics environment is reported and used to the formal specification via a real network (port forwarding feature in Simics). As explained in Section 5.2, execution profile from the system realization is fed back into the formal model to enhance the solution quality.

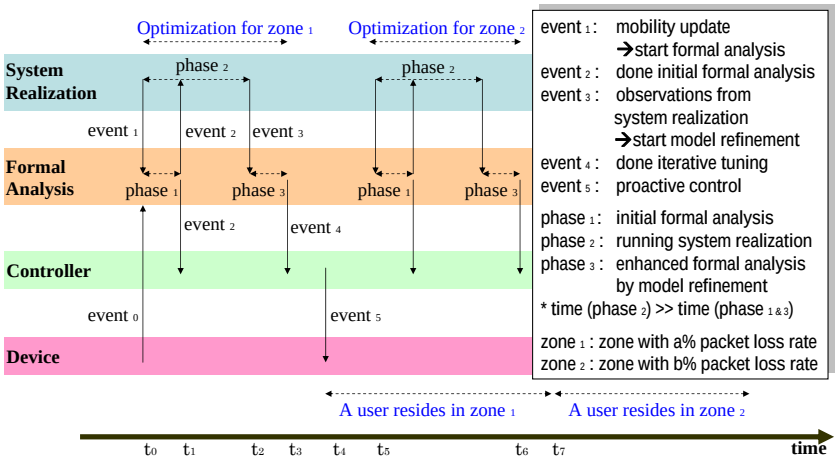
6.2 Model Refinement

Modeling with formal executable specifications, rather than implementing simulators of distributed systems under consideration, enables us to carry out formal analysis (e.g., statistical model checking and quantitative analysis). However, there exist opportunities to improve the formal model to adapt to the system dynamics. For this purpose, we allow model refinement from observed system execution behavior by equipping the controller with a loop to experiment with the system realization.

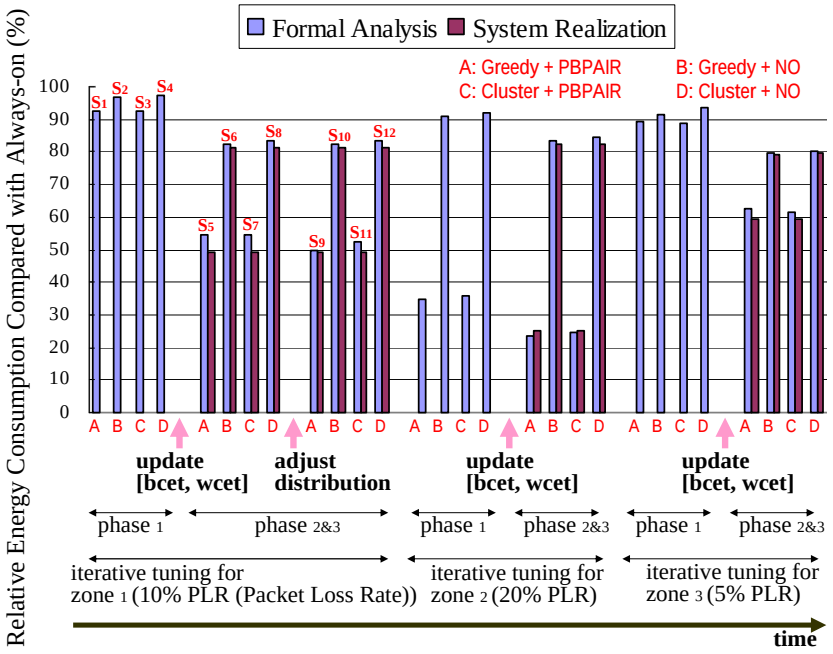
Figure 5 illustrates model refinement based on the dynamic system execution behavior from a system realization. The formal specification initially models the execution times of each task as a function of BCET and WCET from samples, and performs verification/evaluation of the given policies based on that model shown as $phase_1$ in the Figure 5(a). The Maude traces followed by statistical quantitative analysis provide the initial estimations up to time t_1 in Figure 5(a). Since obtaining execution behavior from the system realization usually takes much longer time than formal analysis (e.g., in our case, it is of the order of hundreds times slower than formal analysis), it is beneficial to find the best policy by formal analysis first. At time t_2, the system realization starts generation of the BCET and WCET that reflect the actual executions as described as $event_3$ in Figure 5(a). Then, the formal model is refined by updating BCET and WCET to enhance the analysis results as shown between t_2 and t_3 ($phase_3$).

Let us take an example. At time t_0, the formal specification models PBPAIR execution with [BCET, WCET] as [109 msec, 202 msec][6], and provides the anal-

[6] These profiled values are from [7] for various video inputs.



(a) Test Scenario



(b) Test Result

Fig. 5. Experimental Results: Model Refinement and Proactive Control

ysis results s_1 to s_4 for four different policy/parameter selections (A, B, C, D) in Figure 5(b), respectively. Since our system realization reports dynamic execution of PBPAIR as shown in Figure 6, we can refine [BCET, WCET] to be [66 msec, 125 msec] that leads to formal analysis results s_5 to s_8 in Figure 5(b). Furthermore, we adjust the parameter of the frame encoding time distribution model in the formal specification since many frame encoding times are close to BCET as shown in Figure 6. Instead of simply providing simulated [BCET, WCET] as

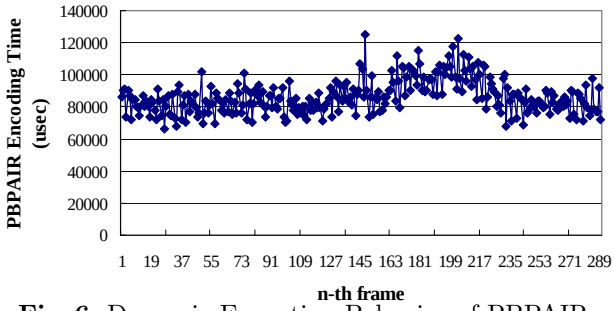


Fig. 6. Dynamic Execution Behavior of PBPAIR

the parameter of the normal distribution model explained in Section 5.2, we use the *actual* average execution time observed from the system realization. Formal analysis results s_9 to s_{12} in Figure 5(b) show a better approximation (i.e., closer to the estimation based on dynamic execution behavior from system realization) due to this adjustment.

It should be pointed out that $phase_1$ (initial formal analysis) and $phase_3$ (enhanced formal analysis) takes much less time than $phase_2$ (running a system realization). The goodness of a policy/parameter selection, however, remains same through $phase_1$ to $phase_3$. This indicates that on-the-fly, light-weight formal verification can be effectively used in adaptation by rapidly narrowing down the search space of potential policies and parameters. Furthermore, the quality of adaptation can be improved by combining formal analysis with observed system execution behavior.

6.3 Proactive Control

As we mentioned in Section 4, we exploit a priori information on a user's mobility (e.g., current zone, speed, and trajectory, etc.) and network situation (e.g., packet loss rate, delay, etc.) to select among policies and related parameter tuning before the user enters a new zone. The mobility information is used to identify the network situation in the current zone and to anticipate the next zone based on a user's speed and trajectory. Ideally, we need prediction techniques like time series analysis [20] to model the future trends in network traffic with some defined level of confidence ($event_0$ in Figure 5(a)). This is, however, beyond the scope of this paper. Currently, we assume that the next zone information is forecast by the system realization ($event_1$).

Figure 5 also illustrates proactive control initiated by network status update. At time t_0, the middleware layer is informed about the next zone information that a user will reach, $zone_1$ with 10% packet loss rate at time t_4. Our framework performs the iterative tuning process — formal execution followed by statistical analysis ($phase_1$) with subsequent model refining ($phase_2$ and $phase_3$) — for the next zone. As a result, our framework can generate controls ($event_5$) to the device before the user enters the new zone (any time between t_3 and t_4). Similarly, at time t_5 the formal model is informed that a user will be in a zone

with 20% packet loss rate at time t_7. By the time t_6, our framework can provide *proactive* controls for $zone_2$.

7 Related Work

The authors of [21] explore probabilistic model checking (PMC) in solving the DPM problem. They obtain the optimal DPM policy by formulating the optimization problem as a discrete time Markov chain (DTMC) model and solve it using an equation solver (e.g., MAPLE [22]). Once a policy has been constructed, its performance is validated using a probabilistic model checking (PMC) tool PRISM [23]. Even though PMC enables experimenting with the effectiveness of a selected DPM algorithm in a quantitative way, the challenge still remains to determine how to actually implement a good power manager that considers complex system dynamics since their work is essentially a validation process for a specific policy using an equation solver. Besides, their analysis of stochastic systems is carried out using numerical solution techniques that are far more memory intensive. On the other hand, our approach is to start with an executable formal model specifying a space of possible behaviors and analyze these possible behaviors using light-weight statistical techniques.

Model-predictive control approaches [24,25] also attempt to address power management issues. In [24], the authors propose predictive learning to shutdown a device by exhaustively searching over a limited look-ahead horizon. In [25], a closed loop feedback control based on queuing theory is presented to optimize CPU frequency. Their solution quality depends on the future events forecasted by a mathematical model (e.g., filter). The applicability of these approaches, however, is limited to systems having a small number of control inputs with synthetic workloads.

The authors of [26] propose an incremental methodology to analyze the effect of a simple time-out based DPM scheme. They start with the functional model without timing and perform a noninterference check for behavior. Then, they extend it to a Markovian model (i.e., the execution time of each action is modeled as an exponential function) and the effect of DPM is evaluated by standard numerical techniques. Lastly, they extend it to a general model by using profiled information from real-world measurements and simulate it to compare the result with that of Markovian model. Our framework can be seen as a generalization of their work. First, since we use Maude formal executable specifications that can have any distribution in timing by controlling the *tick* rule, our formal model corresponds with their general model (a Markovian model can be treated as a specific distribution in the general model). Their mathematical soundness is only guaranteed when the model follows exponential timing. More importantly, our primary focus is on-line adaptation based on abstract formal models complemented by a system realization, not the validation at design time.

8 Conclusions and Future Work

This paper presents a unified framework to develop formal analytical methods for understanding cross-layer and end-to-end timing issues in highly distributed systems that incorporate resource limited devices, and to integrate these methods into the design and adaptation processes for such systems. We propose iterative/proactive system tuning for DRE systems and apply them in a case study treating the videophone mode of a multi-mode multimedia terminal. The integration of formal analysis with the observation of system execution behavior results in a feedback loop that includes the formal models, simulation, and monitoring of running systems for analysis of system behavior and optimizing choice of policies and parameters. The underlying formal executable models are moderately simple to develop, and the analyses seem feasible. The experiments on a fairly complex case study demonstrate the capability of our framework — formal verification combining with observation from system realization — to dynamic tuning of DRE systems.

Ongoing and future work in this project includes:

- considering of a richer class of timed properties
- policy improvement via learning
- modeling and analysis of cross-cutting concerns (e.g., reliability, security)
- carrying out a large scale demonstration with heterogeneous applications (mission critical, multimedia) on multiple devices in a distributed network.

References

1. Kim, M., Dutt, N., Venkatasubramanian, N.: Policy construction and validation for energy minimization in cross layered systems: A formal method approach. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '06) Work-in-Progress Session
2. Forge Project: http://forge.ics.uci.edu.
3. Mohapatra, S., Cornea, R., Oh, H., Lee, K., Kim, M., Dutt, N.D., Gupta, R., Nicolau, A., Shukla, S.K., Venkatasubramanian, N.: A cross-layer approach for power-performance optimization in distributed mobile systems. In: IEEE 19th International Parallel and Distributed Processing Symposium (IPDPS'05)
4. Kim, M., Stehr, M.O., Talcott, C., Dutt, N., Venkatasubramanian, N.: A probabilistic formal analysis approach to cross layer optimization in distributed embedded systems. In: 9th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'07). Volume 4468 of LNCS. 285–300
5. Kim, D., Kim, M., Ha, S.: A Case Study of System Level Specification and Software Synthesis of Multimode Multimedia Terminal. In: IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia'03). 57–64
6. Kim, M., Ha, S.: Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. In: ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'01). 11–19
7. Kim, M., Oh, H., Dutt, N., Nicolau, A., Venkatasubramanian, N.: PBPAIR: an energy-efficient error-resilient encoding using probability based power aware intra refresh. ACM SIGMOBILE Mob. Comput. Commun. Rev. **10**(3) (2006) 58–69

8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: The maude 2.0 system. In: Rewriting Techniques and Applications (RTA'03). Volume 2706 of LNCS. 76–87
9. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Science **285**(2) (2002) 187–243
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All about maude, a high-performance logical framework. LNCS **4350** (2007)
11. Meseguer, J.: Conditional Rewriting Logic as a unified model of concurrency. Theoretical Computer Science **96**(1) (1992) 73–155
12. Real-Time Maude: http://www.ifi.uio.no/RealTimeMaude.
13. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying continuous time markov chains. In: 8th International Conference on Computer Aided Verification (CAV'96). 269–276
14. Jarque, C., Bera, A.: A test for normality of observations and regression residuals. Internat. Statist. Rev. **55**(2) (1987) 163–172
15. Kim, M., Stehr, M.O., Talcott, C., Lee, K., Dutt, N., Venkatasubramanian, N.: Iterative system tuning for proactive systems by formal verification and system prototype: System prototype program interface. CECS Technical Report, UC Irvine (Feb 2007)
16. Signal Process. Multimedia Lab., Univ. British Columbia: TMN 10 (H.263+) encoder/decoder, version 3.2.0 (Sept. 1998)
17. Ebony Board: http://www.amcc.com/Embedded/.
18. http://www.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_440_Embedded_Core
19. Simics Full System Simulation Platform: http://www.simics.net.
20. Han, Q., Venkatasubramanian, N.: AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering. IEEE Distributed Systems On-line **2**(7) (2001)
21. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., Gupta, R.: Using probabilistic model checking for dynamic power management. Formal Aspects of Computing **17**(2) (2005) 160–176
22. Paleologo, G.A., Benini, L., Bogliolo, A., Micheli, G.D.: Policy optimization for dynamic power management. In: 35th Annual Conference on Design Automation (DAC'98). 182–187
23. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). Volume 3920 of LNCS. 441–444
24. Abdelwahed, S., Kandasamy, N., Neema, S.: Online control for self-management in computing systems. In: 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04). 368
25. Lu, Z., Hein, J., Humphrey, M., Stan, M., Lach, J., Skadron, K.: Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'02). 156–163
26. Acquaviva, A., Aldini, A., Bernardo, M., Bogliolo, A., Bonta, E., Lattanzi, E.: Assessing the impact of dynamic power management on the functionality and the performance of battery-powered appliances. In: International Conference on Dependable Systems and Networks (DSN'04). 731