A Probabilistic Formal Analysis Approach to Cross Layer Optimization in Distributed Embedded Systems *

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. We present a novel approach, based on probabilistic formal methods, to developing cross-layer resource optimization policies for resource limited distributed systems. One objective of this approach is to enable system designers to analyze designs in order to study design trade-offs and predict the possible property violations as the system evolves dynamically over time. Specifically, an executable formal specification is developed for each layer under consideration (for example, application, middleware, operating system). 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 QoS properties. We describe how existing statistical approaches have been adapted and improved to provide analyses of given cross-layered optimization policies with quantifiable confidence. The ideas are tested in a multi-mode multi-media case study. Experiments from both theoretical analysis and Monte-Carlo simulation followed by statistical analyses demonstrate the applicability of this approach to the design of resource-limited distributed systems.

Key words: Probabilistic Formal Methods, Statistical Analysis, Cross-layer Optimization, Resource Management

1 Introduction

The next generation of distributed applications will be built around massive scale distributed environments with heterogeneous systems (servers, desktops, mobile devices, sensors, wireless access points, routers, etc.) and networks (WLAN, LAN, WAN, etc.). Such networked applications span multiple domains ranging from mission critical applications for military command/control and disaster response to general purpose end-user applications including education, entertainment, and commerce. An overarching characteristic of these applications are that they are often data intensive and rich in multimedia content with images, GIS (Geographical Information Systems)-based satellite imagery, video and audio data that is fused together from disparate distributed information sources.

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

The content-rich data are expected to be obtained from, delivered to and processed on resource-constrained devices (sensors, PDAs, cellular handsets) carried by users in the distributed network. The dual goals of ensuring adequate application QoS (expressed as timeliness, reliability and accuracy) and ensuring optimal resource utilization at all levels of the system presents significant challenges in system design.

A holistic approach to understanding timing in such systems is essential for several reasons. Firstly, applications are often confronted with end-to-end hard or soft real-time needs. Secondly, existing techniques for timing analysis do not account for the spectrum of granularities of timing which can vary by orders of magnitude across layers. Thirdly, several system level optimizations for effective utilization of distributed resources can interfere with the timing properties of executing applications. For instance, dynamic voltage scaling mechanisms slow down processors to achieve power-savings but at the cost of increased execution times for tasks. Also, knowledge of timing parameters at the different levels can dramatically improve the performance of applications that often execute in constrained environments where CPU, memory, network and device energy is limited.

Our prior experience developing algorithms for managing QoS/power tradeoffs in distributed mobile multimedia applications [1,2] has given us valuable insights into the issues to be addressed. A preliminary study [3] demonstrated the need for integration of formal methods with experimentally based cross-layer optimization methods [1,2]. Systematic analysis based on well-defined models ensures that corner-cases are covered and allows bounds for critical performance parameters to be determined. Our long term goal is to develop a formal methodology to specify and analyze timing constraints at each level, and to correlate timing properties across levels. Furthermore, the formal analyses will be integrated with simulation and experimental methods for developing and adapting system designs. Multimedia applications operated on battery-powered mobile devices are viewed as one of the key application drivers for these next generation distributed systems. Such mobile multimedia applications provide a rich set of QoS/power issues at multiple abstraction levels. Thus, 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.

Our approach is to start with an executable formal model specifying a space of possible behaviors and analyze these possible behaviors using probabilistic/statistical techniques, paying attention to the mathematical meaning of the results. We use the Maude [4] rewriting logic formalism to develop executable specifications that are the basis for subsequent analysis. We have developed two *probabilistic formal* analysis techniques: statistical model checking and statistical quantitative analysis. These analysis results enable policy-based operation and adaptation as well as parameter setting of selected policies.

This paper contains the following contributions:

– a first attempt to integrate *probabilistic formal* methods with cross-layer optimization;
– adaptation and improvement of existing statistical approaches for statistical model-checking and statistical quantitative analysis;
– modeling, simulation and analysis of a fairly complex system.

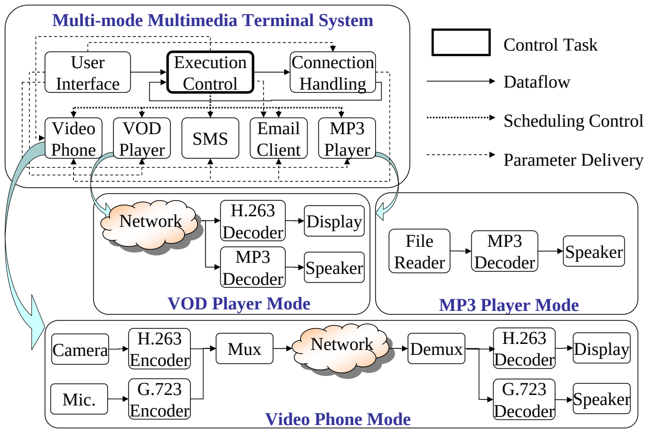

Fig. 1. Case Study: MMMT (Multi-Mode Multimedia Terminal)

The rest of this paper is organized as follows: we start by presenting a multi-mode multimedia communication system as a case study. Next, we describe the modeling effort and specification details of our case study. We then introduce our formal analysis beginning with a brief review of theory, followed by our implementation and experimental results. The last section summarizes our approach and discusses future research directions.

2 Case Study: Multi-Mode Multimedia Terminal

Figure 1 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 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.

We distinguish between two types of optimizations:

– vertical composition, which depicts QoS/energy relationships in a single task, but across several *vertical* layers of abstraction, and
– horizontal composition, which depicts the QoS/energy relationships across multiple tasks in a dynamic environment

In the context of our driver application (MMMT), vertical composition needs to address the application's QoS requirement across layers in the context of resource management constraints. On the other hand, horizontal composition addresses QoS properties between multiple tasks that may be assigned priorities dynamically based on QoS and resource constraints. In this paper, we restrict our discussion to the videophone mode. Horizontal extension for a complete MMMT system remains a topic of future research.

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; 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 DVS decision at OS layer since the execution profile of the application is changed.

3 Formal Modeling and Analysis for Cross Layer Optimization

3.1 What to Model

In this subsection, we explain which features of the MMMT case study will be formally modeled at each layer.

[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 wakeup 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.

– **Application Layer - Proactive PBPAIR:** As an application policy, we utilize proactive PBPAIR. It takes the user's QoS expectation, the network packet loss rate, and raw video sequences as inputs to generate a bitstream robustly encoded against network transmission errors. Therefore, our formal specification needs to generate the execution profile (e.g., when does encoding start/end? how much time is required?). Particularly, we specify an encoding workload profile as a distribution function. For example, we model actual execution time by a uniform distribution between best case execution time (BCET) and worst case execution time (WCET). We also consider a Gaussian distribution with the average and boundary value.
– **Middleware Layer - Network Monitoring:** As briefly mentioned, the middleware layer deals with network status monitoring. We define mobility as a triple *(current zone, speed, trajectory)* to identify the network situation in the current zone and to anticipate the next zone based on user's speed and trajectory. Zone information includes network delay, packet drop rate within the particular zone. Specifically, network transmission delay is modeled as exponential inter-arrival time (Poisson) with mean.
– **OS Layer - Power Management:** Various DPM and DVS power management schemes assuming a worst-case scenario are modeled at the OS layer. The OS layer generates slack time information based on workload from the application layer. This slack time will be used later to reduce energy consumption while guaranteeing QoS requirements for the next frame. Since we are targeting multitask environments, we need to specify various scheduling algorithms (for horizontal composition) like EDF (Earliest Deadline First) and RM (Rate Monotonic).
– **Hardware Layer - Enabling Technology:** To support a DPM and DVS strategy at the OS layer, we assume that the enabling technology (e.g., voltage scalable processor, power-state controllable network card, etc.) is available at hardware layer. In the case of a micro-processor, wakeup/sleep delay and power overhead for a state transition, DVS characteristics (i.e., power consumption for different operating mode/voltage-frequency) should be modeled. As a result of execution, the hardware layer reports residual energy to upper layers.

3.2 Modeling Using Maude

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 [9,4].

Rewriting logic [10] 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

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

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

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

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.

Figure 2 shows a PBPAIR object in the Maude specification for the application layer. In Maude syntax, objects have the general form

$$< \mathit{ObjectName} : \mathit{ClassName} \mid \mathit{Attribute}_1 : \mathit{Value}_1, ..., \mathit{Attribute}_n : \mathit{Value}_n >$$

where *ObjectName* is an object identifier, *ClassName* is a class identifier, and each *'Attribute : Value'* pair specifies attribute identifier and its value. The object *PBPAIR* in Figure 2 has attributes like *WCET, BCET* for generating workload profile.

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,

[5] The idea of a tick rule is taken from Real-Time Maude [11].

```
*** Variables
  vars initWCETProfile initBCETProfile : Map .
  vars T T' : Nat .
  vars I I' Q TH miss cm cmc cmm lost lm clc clm : Int .

*** Object
< PBPAIR : Application |
  WCET : initWCETProfile,          *** worst case execution time
  BCET : initBCETProfile,          *** best case execution time
  accEncTime : T,                  *** accumulated encoding time
  seqN : I,                        *** sequence number
  Timer : T',                      *** next frame arrival time
  IntraTh : TH,                    *** intra threshold (parameter)
  Qsize : Q,                       *** encoding queue size
  bufferedReq : I',                *** buffered frame (initialized as 1/2 × Qsize)
  deadlineMiss : miss,             *** total number of deadline misses
  consecutiveMiss : cm,            *** current consecutive deadline misses
  consecutiveMissCount : cmc,      *** incidence of consecutive deadline miss
  consecutiveMissMax : cmm,        *** maximum consecutive deadline miss
  lostReq : lost,                  *** total number of lost requests
  consecutiveLost : cl,            *** current consecutive lost requests
  consecutiveLostCount : clc,      *** incidence of consecutive lost request
  consecutiveLostMax : clm,        *** maximum consecutive lost request
>
```

Fig. 2. Maude Specification: Application Layer

```
*** 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

then the battery does not expire. 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. For example, *deadlineMiss* and *lostReq* in Figure 2 are observables for this kind.

Furthermore, we use the Maude API, a foreign language interface to embed the Maude rewriting engine into larger applications, to extract observables from

Algorithm 1 Java Foreign Interface for Observables Extraction

```
public static void main (String args[ ])
{
    Maude.initialize("filename.maude");
    mod = Maude.findModule(ModuleName);
    clockedSystem = init(mod, seed);
    conf = extractConfiguration(mod, clockedSystem);
    printObservables(mod, conf);
}
```

the Maude execution and generate statistics of results. Specifically, we use a Java/Maude interface that calls Maude as a dynamic library. The Algorithm 1 gives a simplified overview of this procedure. At the beginning, the *initialize* function of the API initializes the Maude engine and loads the formal specification. Then *findModule* is used to locate the appropriate module identified by *ModuleName*. Now, the *init* function uses other functions of the Maude API to perform rewriting from a suitable initial configuration until a terminal configuration is reached. A random seed will be embedded in the initial configuration to initialize the random number generator that determines which execution path is selected in the model. The *init* function returns the result of executing the model as a *clockedSystem*. The associated configuration will be used for data extraction by *extractConfiguration*. In summary, the Java interface provides a convenient way to deal with data extraction and subsequent processing.

3.3 Analysis

In this section, we explain two statistical evaluation methods that we implemented: statistical model checking and statistical quantitative analysis. 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 %" will be examined. In case of 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 Theory Background and Our Implementation To evaluate a stochastic system properly, we need to remove non-quantifiable non-determinism [12]. We replace all non-determinism with probabilistic choices and stochastic timed operations in the tick rule[6].

– **Statistical Model Checking:** We use statistical model checking to verify probabilistic properties, more precisely hypothesis testing based on Monte-Carlo simulation results. In hypothesis testing, we test whether the probability p of a property under examination is above or below the threshold θ. We can formulate this as the problem of testing the hypothesis $H : p \geq \theta$ against

[6] Non-determinism that is not probabilistic in nature would require the exploration of all possibilities and is currently not supported in our approach. Hence, we use sufficient conditions similar to those of [12] to guarantee the absence of this form of non-determinism.

the alternative hypothesis $K : p < \theta$. Specifically, we implemented two statistical model checking techniques in our framework: *sequential* testing [13] and *black-box* testing [14]. Sequential testing generates sample execution paths until its answer can be guaranteed to be correct within the required error bounds. Black-box testing instead computes a quantitative measure of confidence for given samples. Here, *black-box* means that the system cannot be controlled to generate execution traces, or trajectories, on demand starting from arbitrary states. The implementation of sequential testing and black-box testing can be found as part of the Ymer [15,16] and VeStA [14] tools, respectively.

– **Statistical Quantitative Analysis:** 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 [17]. The normality is determined by testing the null hypothesis (that the sample in vector X comes from a normal distribution with unknown mean and variance) against the alternative (that it does not come from a normal distribution). The JB test computes the *p-value* (the smallest level of significance at which a null hypothesis may be rejected) from the JB test statistic and χ^2 (chi-square) distribution. One rejects the null hypothesis if the p-value is smaller than or equal to the significance level α. For example, a p-value=0.05 indicates that the probability of getting a value of test statistics as extreme as or more extreme than that observed is at most 5% if the null hypothesis is actually true. Once normality of the data is ensured with high confidence, for a large enough number of sample traces n, the approximate average falls inside a (1 - α)100% confidence interval

$$(\bar{x} - Z_{\frac{\alpha}{2},n-1}\frac{s}{\sqrt{n}}, \bar{x} + Z_{\frac{\alpha}{2},n-1}\frac{s}{\sqrt{n}})$$

where $\bar{x}$ is the average of the sample variables, s is samples' standard deviation, and $Z_{\frac{\alpha}{2},n-1}$ is a *standard score* (also called *Z-score* or *normal score*) of normal distribution [18]. To obtain the desired confidence, we want the size of this (1 - α)100% confidence interval to be less than or equal to δ, that is:

$$2Z_{\frac{\alpha}{2},n-1}\frac{s}{\sqrt{n}} \leq \delta.$$

Our Focus: Simplified Formulae and On-demand Sample Generation

– **Statistical Model Checking:** We note that both, the Ymer and VeStA tools, target complex properties of stochastic systems. For instance, those tools take properties specified in a temporal logic, namely Continuous Stochastic Logic (CSL), for Continuous Time Markov Chains (CTMC) [19] system specifications. The reason is that they want to support complex property checking, (e.g., nested temporal/probabilistic operators, and also a form of hybrid model checking in-between numerical and statistical methods), even though the idea of hypothesis testing based on Monte-Carlo simulation does not need to be tied to any specific specification model or temporal logic.

Algorithm 2 Statistical Quantitative Analysis

```
Input: error bound $\alpha$, confidence interval $\delta$, observable under consideration
Output: Expected value E[observable]
initialize $d$ as a negative number;
while ($d > \delta$)
{
    trace generation until normality test succeeds;
    $d = 2Z_{\frac{\alpha}{2},n-1}\frac{s}{\sqrt{n}}$;
}
return the average of observable;
```

This can be an overkill when it comes to analyzing practical optimization problems, if we only test simple properties such as "*Probability* that a system can survive with a given residual energy in t time units is more than θ %". Those formulae are essentially a restricted version of CSL without nesting. Indeed we found no need for nested formulae or an exact numerical solution for our application domain.

– **Statistical Quantitative Analysis:** The Pseudocode 2 shows the statistical quantitative analysis algorithm [12]. As we mentioned earlier, to approximate the expected value by the mean of n samples such that the size of (1 - α)100% confidence interval is bounded by δ, the sample data should follow normal distribution [18]. For the normality test, we need to have a sufficiently large data set. Since the trace generation takes most of the evaluation time, we generate sample traces only if more samples are required (i.e., JB test cannot accept or reject the normality of data.)[7]. By generating traces on demand, we can significantly reduce the evaluation time since it is linearly proportional to the trace generation time (i.e., Monte-Carlo simulation time with a different seed).

4 Experiments

To demonstrate the applicability of our framework to the QoS/energy tradeoff management, we are exploring several aspects of the system optimization. Our formal executable specification (Maude) and evaluation method can serve as a simulation study as well as a statistical guarantee for the design. The outcome of the formal analysis helps us determine the right blend of policies/parameter settings that will enable better QoS and better energy efficiency. The following items are examples of the various facets that we want to address.

– Effect of cross-layer optimization: To evaluate the effect of the cross layer optimization, first we need to quantify the impact of the optimization at each layer and their composition.

[7] Besides, we use the average value from the randomly chosen traces. This random selection may affect the normality test. However, we believe the effect is negligible.

- Effect of confidence requirements: Statistical model checking involves errors by its nature (e.g., the probability of false negatives, the probability of false positives, etc.). Likewise, statistical quantitative analysis provides the answer with confidence interval and error bound. Confidence requirements for the answer have an effect on the number of samples needed, which in turn affects the solution quality and the evaluation time.
- Effect of worst case vs. average case analysis: Currently, we model energy optimization policies (e.g., DVS, DPM, etc.) to reduce energy consumption while satisfying the QoS requirements even in the worst case scenario. However, typical multimedia applications finish execution much earlier than the worst case execution time in most of the situations, which allows more aggressive optimization based on an average case execution scenario.
- Effect of constraint relaxation: QoS degradation due to optimization is sometimes not noticeable from the viewpoint of an end user (e.g., a user may not recognize video quality drop from a single deadline miss for the video decoding.). In such a case, we can relax system's QoS constraints to enable further optimization.

In the following subsections, we will explain experimental results that illustrate the effect of cross-layer optimization. We model *PBPAIR* as an application layer policy as well as various power management schemes – *Greedy, Cluster, DVS* – 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. The *DVS* scheme lowers supply voltage as long as the deadline constraint is satisfied. The arrival of incoming processing requests from the network is modeled as a Poisson process with an average arrival rate. When the processor runs at full speed, the execution times of the tasks are modeled 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. For other types of distributions, we can simply change the Maude operator for the distribution function. 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 [20]. The network zone information is assumed to be given and the DVS capable hardware implementation is from [21]. The experiments were carried out on a 2.8 GHz Pentium 4 processor running Linux.

4.1 Experimental Results

Monte-Carlo Simulation Monte-Carlo simulation in Maude is done with the fair rewrite command that generates one possible behavior of the system, starting from a given initial state using a user specified seed for sampling from distributions. Figure 4(a) presents the energy profiles according to the different policies and buffer sizes. *DVS* with *PBPAIR* outperforms other policies from the perspective of relative energy consumption with respect to *Always-on* (i.e., without any policy). QoS measures such as average deadline miss ratio are also examined to evaluate the effect of cross layer optimization. Figure 4(b) shows

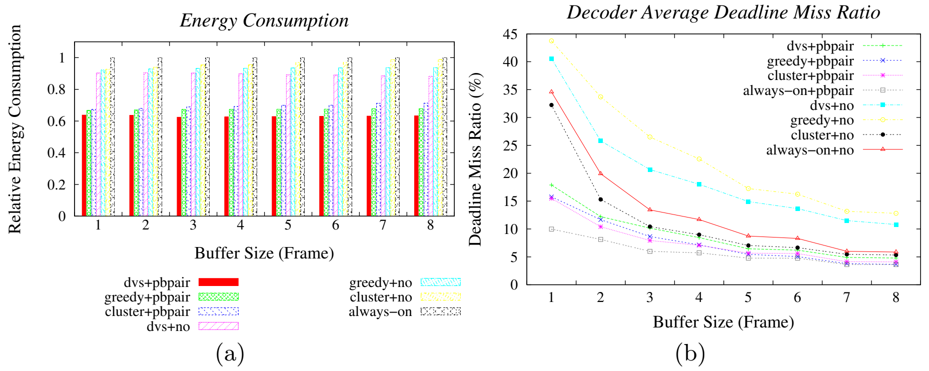

Fig. 4. Monte-Carlo Simulation: Effect of Cross Layer Optimization

that *PBPAIR* combined with any OS layer policy delivers more timely decoding than any OS layer policy without *PBPAIR*. Detailed experimental results on QoS aspects are omitted due to space limitations [22]. Note that the number of possible traces depends on the random seed generator and runtime is linearly proportional to the single trace generation time (i.e., Maude rewriting time from initial state). If we consider the rate of 50 frames each (5 frames/sec) for both encoding and decoding, single trace generation takes around 400-500 msecs. Therefore, it is infeasible to produce all possible traces to evaluate policy and parameter changes in dynamic situations. This led us to propose statistical approaches with quantifiable confidence for our evaluation/decision.

Statistical Model Checking Statistical model checking enables quick detection of problematic situations (e.g., battery expiration) that can arise due to the selection of policy/parameter settings. As an example of sequential testing, we performed statistical model checking of the property

$$Probability\ [\ battery\ expires < 0.1\ ]$$

with arguments $\alpha = 0.05$ (false negative), $\beta = 0.05$ (false positive), $\theta = 0.1$ (threshold), and $\delta = 0.01$ (indifference region), respectively. Sequential testing accepts the hypothesis H_1: $p \ll \theta$ - δ with 133 traces, that is the *batteryExpires* property checker in Figure 3 gives *false* for all traces. With the same 133 traces that were generated for sequential testing, black-box testing also confirms the formula with error of 8.20E-7. The run time for each statistical model checking is 10-20 msecs in addition to the sample generation, which indicates that this is a feasible proposition for the on-the-fly adaptation.

Statistical Quantitative Analysis In Section 3.3, we explained the prerequisite and theoretical background for the statistical quantitative analysis and this section provides experimental results on that. Specifically, we test normality of data before we apply the central limit theorem to approximate the expected value by the average of sample mean. We first generate an initial number of sample traces for JB normality test followed by additional trace generation until

(a) Energy Consumption:
[nSample = 100] Fail to reject Ho (p-value = 0.821)
E[Energy Consumption] = 3.7121E9 ($\alpha = 5.0\%$, $d = 0.036\%$)

(b) Decoder Average Deadline Miss Ratio:
[nSample = 100] Reject Ho (p-value = 0.035)
[nSample = 110] Fail to reject Ho (p-value = 0.194)
E[Decoder Avg Deadline Miss Ratio] = 0.2032 ($\alpha = 5.0\%$, $d = 0.466\%$)

(c) Decoder Maximum Consecutive Lost:
[nSample = 100] Fail to reject Ho (p-value = 0.884)
[nSample = 100] ($d = 0.01053$) > ($\delta = 0.01$)
[nSample = 110] ($d = 0.01002$) > ($\delta = 0.01$)
[nSample = 121] ($d = 0.00958$) $\leq$ ($\delta = 0.01$)
E[Decoder Maximum Consecutive Lost] = 3.2314 ($\alpha = 5.0\%$, $d = 0.958\%$)

Fig. 5. Statistical Quantitative Analysis

the normality test succeeds. The p-value should be more than or equal to the error bound (α) to accept normality of given data set (i.e., fail to reject the null hypothesis Ho). If we can not *statistically* limit the confidence interval by δ (*while* loop condition in Pseudocode 2), we produce more samples on-demand.

The Figure 5(a) and 5(b) show our statistical quantitative analysis results with arguments of α (error bound) and δ (confidence interval) as 5% and 1%, respectively. In Figure 5(a), the observable *EnergyConsumption* passes the normality test with 100 initial samples since its p-value (0.821) is more than error bound α (0.05). The resulting confidence interval d (0.036%) is less than the desired value δ (1%). Therefore, we can estimate the expected value for *EnergyConsumption* within error bound and confidence interval. On the other hand, in case of the *DecoderAvgDeadlineMissRatio* observable (Figure 5(b)), we need to generate more samples (10% in this experiment) since the first JB test fails. Even if the sample data follows a normal distribution, we may need more samples for limiting the confidence interval by δ. Figure 5(c) presents such a case. The confidence interval from initial samples (d) is 1.053% and the desired interval (δ) is 1%. Therefore, more samples are generated until d is less than δ.

5 Previous and Related Work

In our previous work (FORGE project [1]), we have identified interaction parameters between the different computational levels that can facilitate effective cross-layer coordination. We have also developed a set of APIs at the various computational layers. As a result, we successively exploit the cross-layer approach to achieve energy gains while preserving QoS requirements of distributed multimedia applications (e.g., streaming video, video conferencing). Specifically, we have studied how to annotate application data with specific information that can be used to improve power efficiency and how to optimize parameters in various layers (e.g., image quality in application layer, the compressed size in network layer, and execution time/power consumption in hardware layer) [2]. We also explored the trade-off between the error resiliency level, compression efficiency, and power consumption for streaming multimedia applications [7].

To leverage our prior effort, we are integrating formal methods with cross-layer optimization in a unified framework.

Previous work on statistical model checking for stochastic systems includes PMaude (Probabilistic Maude) [23,12], a rewriting-based specification language for modeling probabilistic concurrent and distributed systems. The associated tool, VeStA [14], was developed to statistically analyze various quantitative aspects of models such as those specified in PMaude using a query language QuaTEx (Quantitative Temporal Expression) [12] based on CSL (Continuous Stochastic Logic). However, this approach does not provide any procedure by which they can determine the sample size required to achieve normality. Moreover, the authors approximate the expected average by applying *Student's T*-distribution. This is unnecessary since as the sample size n grows, the T-distribution approaches the normal distribution with mean 0 and variance 1. Therefore, we extended their approach by an on-demand sample generation that can compute the sample size sufficient to guarantee the normality of data, and utilize the normal distribution to obtain the error bound and confidence interval.

Ymer [15,16] implements statistical techniques, based on discrete event simulation and sequential acceptance sampling for CSL model checking. The system is modeled by continuous-time Markov chains (CTMCs) and generalized semi-Markov processes (GSMPs). Properties are expressed using Continuous Stochastic Logic (CSL). Ymer also integrates numerical techniques to solve nested CSL queries, by including the hybrid engine of the PRISM [24] tool for CTMC model checking. This, however, limits the modeling power compared with our approach. On the other hand, the expressive power of the Maude language (extended with probability and time) opens a wide spectrum of applications that are beyond the scope of Markovian models.

6 Summary and Future Work

This paper presents the results of the first phase of a project to develop formal analytical methods for understanding cross-layer and end-to-end timing issues in highly distributed systems incorporated resource limited devices, and to integrate these methods into the design and adaptation processes for such systems. We have developed new analysis techniques that combine statistical and formal methods and applied them in a case study treating the videophone mode of a multi-mode multimedia terminal. The results are encouraging, as the underlying formal executable models are moderately simple to develop, and the analyses seem feasible.

Ongoing and future work in this project includes:

– modeling and analysis of the remaining modes of the MMMT (Section 2) as well as scheduling policies and sharing of resources between tasks.
– carrying out a trade-off analysis on the effect of confidence requirements, worst case vs. average case execution models, and constraint relaxation (as discussed in Section 4)

– integration of formal analysis with the simulation framework that includes real system prototypes. This will result 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.

References

1. Forge Project: http://forge.ics.uci.edu.
2. 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: International Parallel and Distributed Processing Symposium (IPDPS '05)
3. Kim, M., Dutt, N., Venkatasubramanian, N.: Policy construction and validation for energy minimization in cross layered systems: A formal method approach. In: Real-Time and Embedded Technology and Applications Symposium (RTAS '06) Work-in-Progress Session. 25–28
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All about maude, a high-performance logical framework. Lecture Notes in Computer Science **4350** (2007)
5. Kim, D., Kim, M., Ha, S.: A Case Study of System Level Specification and Software Synthesis of Multimode Multimedia Terminal. In: 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. ACM SIGPLAN Notices **36**(8) (August 2001) 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. 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 & Applications (RTA '03). 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. Meseguer, J.: Conditional Rewriting Logic as a unified model of concurrency. Theoretical Computer Science **96**(1) (1992) 73–155
11. Real-Time Maude 2.2: http://www.ifi.uio.no/RealTimeMaude.
12. Agha, G.A., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. Electr. Notes Theor. Comput. Sci. **153**(2) (2006) 213–239
13. Wald, A.: Sequential tests of statistical hypotheses. Annals of Mathematical Statistics **16**(2) (1945) 117–186
14. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Computer Aided Verification (CAV '04). 202–215
15. Younes, H.: Ymer: A statistical model checker. In: Computer Aided Verification (CAV '05). 429–433
16. Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. International Journal on Software Tools for Technology Transfer (STTT) **8**(3) (2006) 216–228
17. Jarque, C., Bera, A.: A test for normality of observations and regression residuals. Internat. Statist. Rev. **55**(2) (1987) 163–172
18. Hogg, R., Craig, A.: Introduction to Mathematical Statistics. Fifth edn. (1995)
19. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic **1**(1) (2000) 162–170
20. Image Process. Lab., Univ. British Columbia: TMN 10 (H.263+), ver. 3.2.0 (1998)
21. http://www.intel.com/design/pca/prodbref/252780.htm
22. Kim, M., Stehr, M.O., Talcott, C., Dutt, N., Venkatasubramanian, N.: Modeling and Exploiting Cross-Layer Optimization in Distributed Embedded Systems. Technical Report SRI-CSL-07-02, SRI International (Feb. 2007)
23. Kumar, N., Sen, K., Meseguer, J., Agha, G.: A rewriting based model for probabilistic distributed object systems. In: Formal Methods for Open Object-based Distributed Systems (FMOODS '03). 32–46
24. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS '06). 441–444