Constraint Refinement for Online Verifiable Cross-Layer System Adaptation *

Minyoung Kim[1], Mark-Oliver Stehr[2], Carolyn Talcott[2], Nikil Dutt[1], Nalini Venkatasubramanian[1]
[1]School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA
{minyounk, dutt, nalini}@ics.uci.edu
[2]Computer Science Laboratory
SRI International, Menlo Park, CA 94025, USA
{stehr, clt}@csl.sri.com

Abstract

Adaptive resource management is critical to ensuring the quality of real-time distributed applications, particularly for energy-constrained mobile handheld devices. In this context, an optimization that simultaneously considers multiple layers (e.g., application, middleware, operating system) needs to be developed for continuous adaptation of system parameters. The tuning of system parameters greatly affects the system's ability to meet QoS requirements, and also directly affects the energy consumption and system robustness. We present a novel approach to developing cross-layer optimization for resource limited real-time distributed systems, based on a constraint refinement technique combined with formal specification and feedback from system implementation. Our approach tunes the parameters in a compositional manner allowing coordinated interaction among sub-layer optimizers that enables holistic cross-layer optimization. We present experiments on a realistic multimedia application which demonstrate that constraint refinement enables us to generate robust and near optimal parameter settings. The constraint language can be used as an interface for composition by encapsulating the details of local optimization algorithms.

1 Introduction

Next generation mobile embedded applications are highly networked, and involve end-to-end interactions among multiple layers (application, middleware, OS, hardware architecture) in a distributed real-time environment. The dual goals of ensuring adequate application QoS (Quality of Service, expressed as timeliness, reliability, and accuracy) and optimizing resource utilization at all levels of the system presents significant challenges. Therefore, we need to provide a unified framework that enables system designers to analyze designs in order to study design tradeoffs across multiple layers and tune the system parameters while predicting the possible property violations as the system evolves dynamically over time.

Especially, a distributed multimedia application requires frequent policy and parameter tuning based on user input and/or node/network conditions (e.g., residual power level, packet drop rate, noise level, etc.). As we see from the layered view of a device in Figure 1, policy selection determines how decisions are made at different layers: a specific video encoding/decoding algorithm at the application layer; network monitoring at the middleware layer; and DVS (Dynamic Voltage Scaling [5]) at the OS layer. Network traffic shaping and/or trans-coding at the middleware layer can be also utilized. Each policy has parameters that can be set to fine-tune the behavior. The policy itself can be regarded as a discrete parameter. In addition, there are hardware parameters that can be set.

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

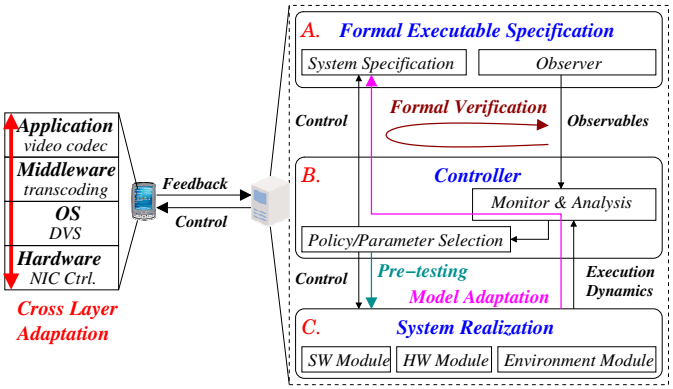


Figure 1. Cross-Layer Optimization Framework

Clearly, in such a scenario, policies made at one layer can affect behavior at other layers. Thus a *holistic* cross-layer optimization is needed [13, 16]. However, a global approach to the holistic optimization that is fully aware of the complex system dynamics can introduce high overhead. Therefore, we propose a compositional cross-layer optimization by coordinated interaction among local (sub-layer) optimizers through constraint refinement. The constraint refinement allows encapsulation of detailed system state information. The key idea underlying the compositional optimization is that each local optimizer uses refinement results of other optimizers as its constraints. The constraint representation can be used as the generic interface among different local optimizers, leading to substantial improvement of solution quality at low complexity.

Comprehensive analysis of cross-layer QoS-energy tradeoffs and coordinated interaction enable us to tune policy parameters of highly resource limited devices. In our chosen multimedia application, our approach selects policy parameters, such that the objective function (balancing QoS and energy requirements) is optimized while meeting basic QoS requirements (e.g., timing, frame drop, etc.) for the individual tasks. Our experimental results indicate that the compositional cross-layer optimization produces solutions that are reasonably close to those produced by a global approach, but with significantly less complexity. Furthermore these solutions are robust to small perturbations, making our approach applicable in a realistic dynamic setting.

2 Cross-Layer Optimization Framework

Before we dive into the compositional cross-layer optimization by constraint refinement, we clarify the scope of this work. Figure 1 presents our framework that employs iterative tuning using light-weight, on-the-fly formal verification with feedback from system execution for dynamic adaptation. We take three

major steps: (1) formal modeling, (2) analysis and optimization, and (3) model adaptation and proactive control.

In Figure 1, *Box A* represents the formal model. 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. Detailed explanation of our model representation and use of Maude [2, 1] as a reasoning tool based on the rewriting logic formalism [12] can be found in [8].

Box B in Figure 1 shows the evaluation phase of given specifications to generate statistics of monitored properties and values. Specifically, we have advanced existing analysis techniques by combining statistical and formal methods, and applied them to a case study that treats the videophone mode of a multi-mode multimedia terminal [8].

Using such models and analysis, tools can be developed to map adaptive system specification into appropriate policy/parameter settings. In [7], we proposed an *iterative* tuning strategy that combines formal methods with dynamic system execution behavior (obtained from either a simulation or an 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 adaptation*). 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).

Continuing this line of research, the focus of this paper is the policy/parameter selection (controller in *Box B*) for cross layer optimization. The continuously adapting formal models provide the predictive capabilities needed for the compositional cross-layer optimization.

3 Preliminaries

In this section, we define the terminology (policy, parameter) for our target application[1] and the key assumptions underlying our approach followed by a formal problem statement.

3.1 Terminology and Assumptions

We consider *proactive* PBPAIR (Probability-Based Power-Aware Intra Refresh) [6] 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. If PBPAIR is selected as an application layer policy, then algorithm-specific parameters such as the *IntraTh* (intra threshold) value must be chosen for appropriate execution. We also consider *buffer size estimation* as an application layer policy to balance the power consumption and

[1] The methodology itself can be applied to systems with tuning parameters (beyond multimedia systems) if the solution space (=utility function) exists in a well shaped form. If the solution space is too random, then constraint refinement (or any other optimization) may not find the solution space with significantly better property than random trials.

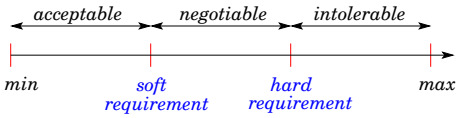


Figure 2. Types of Requirement

QoS by allocating appropriate size of buffer [14]. In the OS layer, the DVS parameter *Deadline Completion Ratio* [3] — the tolerance level of QoS in terms of task completion that satisfies its deadline — is tuned. Note that the parameter selection at one layer affects other layers. For example, PBPAIR increases intra-coding by lowering the *IntraTh* 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.

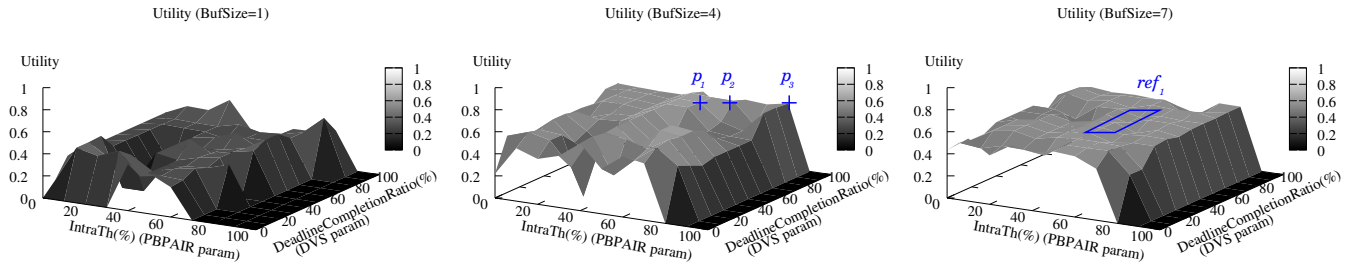
We next state our key assumptions: Our application is a firm real time system with QoS requirements, and the goal is to maximize the QoS in an energy-efficient way while satisfying all requirements.

• *Requirements* — We define two types of requirements: soft and hard requirements. As described in Figure 2, the situation that the system behavior resides below the *soft* requirement is the most desirable. When the system is observed in between *soft* and *hard* requirement, however, the controller needs to tune the parameters to utilize the tradeoff between QoS and the energy consumption. We also assume that there is an upper limit, above which a user cannot tolerate the quality degradation (*hard* requirement). For instance, assume that the soft requirement and hard requirement for deadline miss ratio are given as 5% and 20%, respectively. A user, i) cannot allow any parameter settings that lead to the deadline miss ratio over 20%, ii) is willing to tolerate more deadline misses up to 20% to reduce energy consumption, iii) accepts any parameter settings whose deadline miss ratio is below 5%.

• *Objective* — We define a *utility* function that captures the *effectiveness* of given parameter settings. The utility is a function of energy consumption, QoS (timing violation, frame drop ratio), bandwidth demand, and buffer size estimation. Specifically, we use a weighted sum of the evaluation functions of each component.Figure 3 presents an example of utility distribution according to the various parameter settings. In Figure 3(a), there are parameter settings that lead to *zero utility* meaning that a *hard* requirement is violated. Traditional optimization methods (e.g., simulated annealing) may pick any solution point with maximum utility such as p_1, p_2, and p_3 in Figure 3(b). Those solutions, however, cannot guarantee reliable performance in a dynamically changing environment since small perturbations of the parameters may lead to large drop in utility. In this context, operating in the refined region ref_1 in Figure 3(c) is more desirable for system robustness.

3.2 Problem Statement

Given a system specification (parameters and observables) and a method to evaluate solutions (utility and requirements), our problem is to find settings of parameters that maximize the system utility (considering the trade-off between energy and QoS), while ensuring that the solution guarantees a certain level of robustness. A solution means choosing the parameters for each policy — *IntraTh*, *BufSize*, and *DeadlineCompletionRatio*. Using this two layer (Application and OS) scenario, our problem is described more formally below:



(a) (b) (c)

Figure 3. Utility Distribution: Effect of the Parameter Settings on Varying IntraTh (Application Layer) and DeadlineCompletionRatio (OS Layer) with Buffer Size (a) 1 Frame, (b) 4 Frames, and (c) 7 Frames

Input– Parameters, Observables: The parameters (depicted as *controls* in Figure 1) at each layer and the observables (from system execution by using formal specification and/or system realization in Figure 1) provide different aspects of system dynamics.

• Parameter Space $\mathbb{P}$ is a Cartesian product of the parameter spaces of all layers (e.g., $\mathbb{P} = \mathbb{P}_{App} \times \mathbb{P}_{OS}$ with $\mathbb{P}_{App} = \mathbb{R} \times \mathbb{N}$ and $\mathbb{P}_{OS} = \mathbb{R}$). For instance, $((0.5, 3), 0.7) \in \mathbb{P}$ represents the parameter settings of 50% *IntraTh*, 3 frames *BufSize* (Application layer), and 70% *DeadineCompletionRatio* (OS layer). We assume that $\mathbb{P}$ is equipped with the natural ordering $<$ induced by the orderings of its components.

• Observation Space $\mathbb{O}$ is a Cartesian product of all possible observation results for each observable (e.g., $\mathbb{O} = \mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}$). For instance, $(0.14, 0.03, \cdots, 3.78\text{e+}7) \in \mathbb{O}$ represents 14% *DeadlineMissRatio*, 3% *FrameDropRatio*, and 3.78e+7(uJ) *EnergyConsumption*, respectively.

Input– Requirements, Utility Function: The system utility U is evaluated by a function of soft/hard requirements $\mathbb{Q}$ (Section 3.1) and observables $\mathbb{O}$ (results of parameter setting).

• Utility $U : \mathbb{P} \times \mathbb{O} \rightarrow \mathbb{R}$ can take many forms (e.g., a weighted sum). For instance, we assume $U = w_{App}u_{App} + w_{OS}u_{OS}$ where w_{layer} and u_{layer} correspond, respectively, to the weight and utility function (based on $\mathbb{Q}$ and $\mathbb{O}$) of *layer*. Now we define $U^* : \mathbb{P} \rightarrow \mathbb{R}$, i.e., $U^*(p) = U(p, \mathcal{M}(p))$, where $\mathcal{M}$ is a model (formal executable specification) and $\mathcal{M}(p)$ is the observation given by $\mathcal{M}$ for parameter p.

Note that a system model $\mathcal{M}$ is not constant due to dynamicity and non-determinism of a system, and we allow our formal model to effectively reflect on-the-fly system behavior (*model adaptation* in Figure 1).

Problem Objective: For the given $\mathbb{P}$, $\mathbb{O}$, $\mathbb{Q}$, and U, the objective is to determine a refined region P of parameter settings with a certain level of guarantee on the utility.

Now, we define a region P over $\mathbb{P}$.

• Region $P \in \mathcal{R}(\mathbb{P}) \iff P \subseteq \mathbb{P}$ is a closed convex set, (i.e., if $(x, z \in P) \bigwedge (x < y < z)$, then $(y \in P)$) and P is finitely representable (e.g., interval-based). Then, the objective is to find a refined region $P \in \mathcal{R}(\mathbb{P})$ such that $\widehat{U}^*(P) : \mathcal{R}(\mathbb{P}) \rightarrow \mathbb{R}$ is maximized and $size(P) = size(\mathbb{P}) \cdot \tau$, where $size(P)$ is a suitable measure of the size of a convex set P (e.g., area or volume), τ represents the refinement ratio $(0.0 < \tau < 1.0)$ and $\widehat{U}^*(P)$ can be defined to fulfill various optimization goals. For example, to maximize the average of utilities in P, $\widehat{U}^*(P)$ can be defined as $\widehat{U}^*_{avg}(P) = avg\{U^*(p)|p \in P\}$. On the other hand, $\widehat{U}^*_{min}(P) = min\{U^*(p)|p \in P\}$ can be used for maximizing the minimum of utilities in P.

4 Proposed Approach

Typical optimization techniques [11, 4, 13, 16] provide a single best design point for a given system specification and utility function. We discuss the potential drawbacks of those techniques — that find a *single* parameter setting p such that $U(p)$ is maximized — in the following:

• Lack of Robustness — The system model $\mathcal{M}$ needs to be adaptive to reflect the dynamic nature of the system (e.g., network status keeps changing based on a user's mobility). Besides, there is non-determinism (i.e., randomness) in the model as well as run-time variation such as data-dependent execution. Under such high dynamicity, reliable optimization becomes an issue. Therefore, we focus on providing a near optimal solution region P (rather than a single point p) that can further provide useful information on robustness and performance of the obtained solution space.

• Lack of Composability — As the system gets more complex, integrated treatment of individual optimizers with its own requirements and objective is important. In this case, one parameter setting p_a obtained by an optimizer opt_a can adversely affect the other optimizer opt_b, since p_a can over-restrict the solution space of opt_b. Our constraint refinement additionally enables us to support cooperative composition through the encapsulation of detailed optimization information using the generic interface of a constraint representation language.

In the following, we describe our solution in some detail. First, we explain constraint refinement for robust optimization. Then, we define our compositional cross-layer optimization based on this representation.

4.1 Constraint Refinement

Given an optimization problem for which the model $\mathcal{M}$ and the parameter space $\mathbb{P}$ are complex, and for which no better solution to find a region $P \in \mathcal{R}(\mathbb{P})$ is known than an extensive brute-force search, our approach attempts to quickly find an approximate solution by the following:

1. Recursive Resampling: We obtain observables by Monte-Carlo sampling over the current region $P_i \in \mathcal{R}(\mathbb{P})$ using the model $\mathcal{M}$. Subsequently, we refine P_i to P_{i+1} such that $\widehat{U}^*(P_{i+1})$ is maximized based on the samples available, and $size(P_{i+1}) = size(P_i) \cdot \tau_i$, where τ_i represents the i-th refinement ratio. The new region P_{i+1} is then used as the current region and the process is repeated.

2. Interval-based Description: For simplicity we use regions defined by the Cartesian product of intervals for

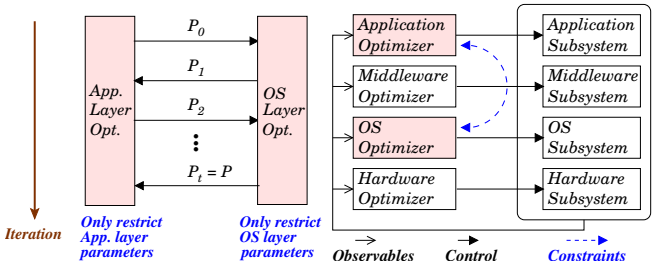


Figure 4. Constraint Refinement for Composition (a) Parallel Composition of Layers, (b) Compositional Cross-Layer Optimization

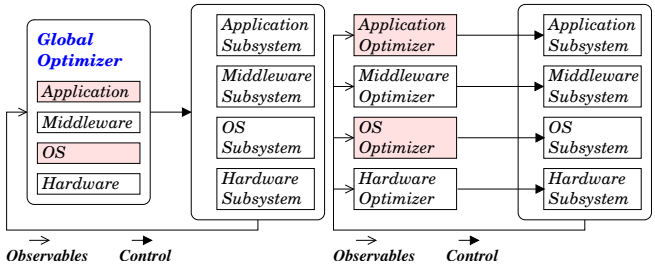


Figure 5. Comparison among On-line Optimizations (a) Global Cross-Layer Optimization, (b) Without Cross-Layer Optimization

each of the parameters. For example, an application layer region might be $P_{App} = [IntraTh_{min}, IntraTh_{max}] \times [BufSize_{min}, BufSize_{max}]$. Clearly, more expressive constraint languages are possible in our framework and should be investigated in the future.

3. Generic Constraint-based Interface: The input (P_i) and output (P_{i+1}) of each refinement step are regions (infinite sets), and our approach lifts the level of abstraction by treating P_i as *constraints* (finite symbolic representations) when we restrict the resampling space to find P_{i+1}.

The process of constraint refinement can be stated as

$$\mathbb{P} = P_0 \supseteq P_1 \supseteq P_2 \supseteq \cdots \supseteq P_t = P$$

where P is the set of admissible parameter settings at termination after t iterations.

Our experimental results indicate that the constraint refinement can be effectively used for robust parameter selection. One key feature of this approach is that we can coordinate parallel composition of individual optimizers as illustrated in Figure 4(a). Each sub-layer optimizer controls a subset of parameters. For instance, the application layer optimizer only restricts its own parameters ($\mathbb{P}_{App}$) while the OS layer optimizer only restricts OS related parameters ($\mathbb{P}_{OS}$). The constraints P_i are used as inputs *and* outputs of individual (sub-layer) optimizers.

Composition through constraint refinement reduces the possibility of conflicts because of the more general notion of a solution compared with traditional single point optimizers. More importantly, constraint refinement enables simple yet powerful cross-layer optimization via composition (Figure 4(b)), as discussed in the next Section.

4.2 On-line Cross-Layer Optimization

The primary goal of our framework is to enable on-line cross-layer optimization that provides the refined parameter settings from which a system can select any suitable operating point within the region as explained in Section 4.1. The constraint refinement allows encapsulation of detailed system optimization information. This opens up the possibility of coordinated interaction (composition) instead of relying on a global view. Figures 5 and 4(b) compare the global vs. local vs. compositional approach for cross-layer optimization. The key idea underlying the *compositional* optimization is to exchange the local optimizer's decision for an informed selection. This allows us to achieve a balance between *global* optimization's full awareness with high overhead and *local* optimization's minimal complexity with poor solution quality.

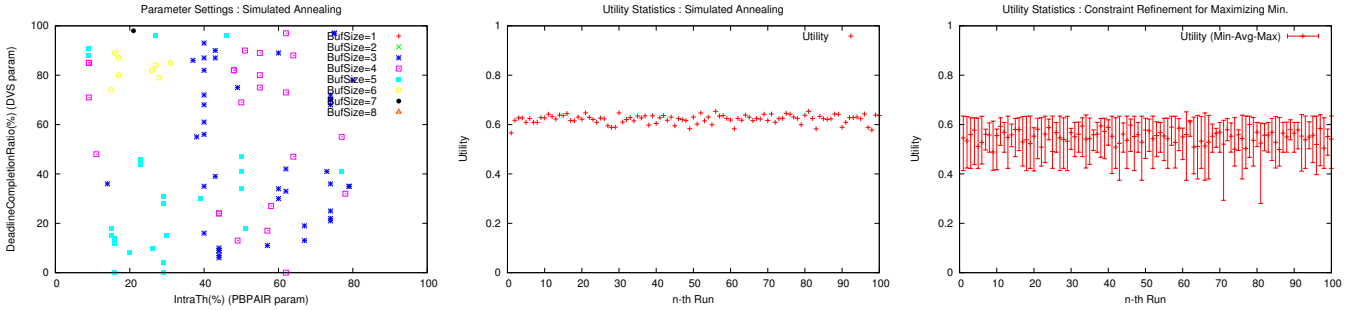
More specifically, each local optimizer uses the other optimizer's refinement results as its constraints. As an example, if the application layer optimizer refines the PBPAIR parameter *IntraTh* to [20%, 80%] and *BufSize* to [2 frames, 5 frames], then the OS layer optimizer refines its parameter *DeadlineCompletionRatio* to [50%, 100%] taking the application layer parameter ranges as *constraints*. The OS layer results are transmitted to the application layer optimizer for further refinement. Thus, the constraint language can be used as the generic interface among different local optimizers, leading to substantial improvement of solution quality at low complexity.

A similar strategy can be applied to other optimization techniques (e.g., simulated annealing [9, 10]). The strict convergence to a single point, however, may not be achievable in the sense that at each step the intermediate parameter settings may be totally different from the previous iteration. For instance, if the application layer chooses high *IntraTh* to reduce the energy consumption, the OS layer is informed about high *IntraTh* and tunes its parameter *DeadlineCompletionRatio* to a high value since high *IntraTh* indicates less workload from the OS perspective. Then, this high *DeadlineCompletionRatio* is transmitted to the application layer, which may cause it to decrease *IntraTh* since it can be interpreted as sufficient slack time in the system for further encoding quality upgrade by decreasing *IntraTh*. These types of abrupt and/or constant parameter changes are not desirable in practice. Constraint refinement can still undergo constant parameter changes, but with *lower* impact since any parameter settings (p_i) within the region (P_i) can be chosen, and the probability that p_i is valid after the next iteration ($p_i \in P_{i+1}$) is proportional to the refinement ratio. We can also easily see that the situation will worsen with conflicting local objectives.

It should be pointed out that our approach is not limited to a specific constraint refinement protocol scheme. Compositional optimization through constraint refinement enables a controller to coordinate existing optimizers (possibly distributed) that can accommodate different objectives by treating them as black-boxes, which in turn permits to process them in parallel. Different solutions obtained in parallel can be unified by taking the intersection, which corresponds to the conjunction at the symbolic level.

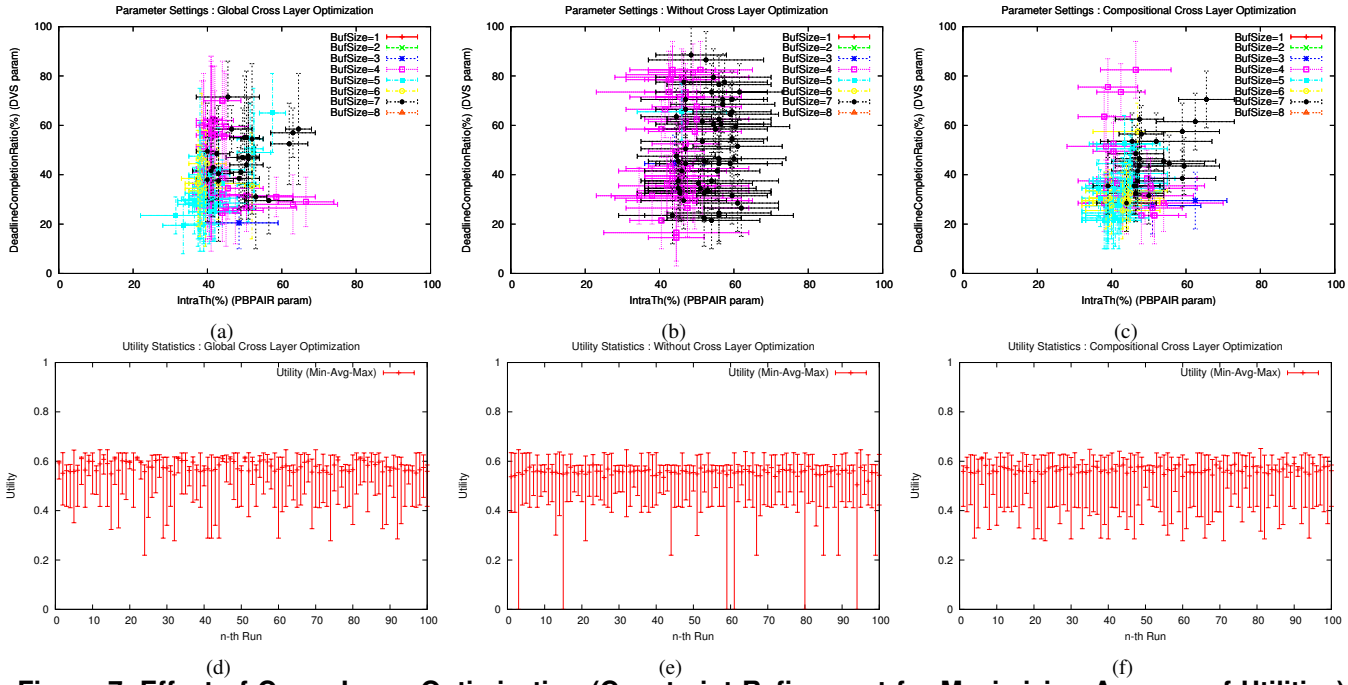
5 Experiments

We evaluated the effectiveness of our approach by carrying out a variety of experiments. Our first set of experiments focused on global constraint refinement for the reliable optimization of a mobile multimedia system. In our second set of exper-



(a) (b) (c)

Figure 6. Simulated Annealing (SA) vs. Constraint Refinement (CR) — (a) Parameter Settings of SA, (b) Utility Statistics of SA, (c) Utility Statistics of CR for Maximizing Minimum of Utilities



(a) (b) (c)

(d) (e) (f)

Figure 7. Effect of Cross-Layer Optimization (Constraint Refinement for Maximizing Average of Utilities) — Parameter Settings and Utility Statistics (a),(d) Global Cross-Layer Optimization; (b),(e) Without Cross-Layer Optimization; (c),(f) Compositional Cross-Layer Optimization

iments, we focused on the effect of composition in the context of on-line cross-layer optimization.

5.1 Constraint Refinement

Figures 6 and 7 show the results of parameter setting and utility statistics from different optimization techniques. We present 100 optimization results for our mobile multimedia system — parameter settings and utility statistics — using the utility distribution in Figure 3. We compare our constraint refinement with simulated annealing (SA) [9]. Specifically, we implemented a simulated annealing method with adaptive neighbor feature for the continuous parameter optimization proposed in [10].

In Figure 6(a) and 7(a), we see the parameter settings from simulated annealing and constraint refinement for maximizing the average of utilities, respectively. For fair comparison, we use same number of samples (evaluations of the model $\mathcal{M}$) as in simulated annealing. We also use uniform refinement ratio $\bar{\tau}$ (i.e., $\tau_0 \cdot \tau_1 \cdots \tau_{t-1} = \tau$ and $\forall i \in [0, \cdots, t-1] : \tau_i = \bar{\tau}$) and a constant number of iterations t. In each figure, the X-axis represents the application layer parameter *IntraTh* while the Y-axis represents the OS layer parameter *DeadlineCompletionRatio*. The *BufSize* parameter is pictured in different colors. In the case of simulated annealing, the parameter setting is represented as a distinctive point. Constraint refinement, however, provides a region represented as cross bars parallel with the x-y axes. Compared to constraint refinement approaches, simulated annealing obtains higher utility in most of the cases (Figure 6(b)). From the implementation perspective, however, it is practically infeasible to construct the controller with frequent jumps across the entire parameter space as shown in Figure 6(a). On the other hand, Figure 7(a) and 7(c) present the parameter settings as refined regions for different cross-layer optimizations. Our result is more deterministic thanks to the additional degree of freedom offered by the constraint representation.

Utility statistics from simulated annealing, constraint refine-

ment for maximizing the minimum of utilities, and constraint refinement for maximizing the average of utilities are presented in Figure 6(b), 6(c), and 7(d), respectively. We present different optimization results (X-axis) in terms of utility statistics (Y-axis with minimum-average-maximum). As we mentioned earlier, one advantage of constraint refinement is its ability to determine robust parameter settings (in form of a *region* that includes *many* points with a *guarantee* of solution quality) even though the average of utilities is lower than that of simulated annealing. In many cases, the maximum utility from a refined region is higher than the utility from simulated annealing, which indicates that the best parameter setting from simulated annealing can be dominated by one of the multiple points in the refined region. In addition, as shown in Figures 7(d) and 6(c), because our constraint refinement maintains utility statistics, not just single best options, alternative objectives may be achieved while preserving utility: optimization of average-case performance (Figure 7(d)) and system tuning to guarantee worst-case bound (Figure 6(c)).

5.2 On-line Cross-Layer Optimization

Our discussion so far indicates that constraint refinement can be effectively used for reliable parameter selection. In next set of experiments we study the effect of cross-layer optimization. The results are presented in Figure 7. For these experiments, we use constraint refinement in individual (sub-layer) optimizers to optimize for the average-case performance (i.e., maximizing the average of utilities). We first consider the two extremes: without cross-optimization (i.e., *local* optimization) in Figure 7(b),(e) vs. *global* optimization in Figure 7(a),(d). For the *local* optimization without cross-layer concerns, there is much higher chance of a failure (6% vs. 0% in case of global/compositional optimization) — parameter settings leading to the *hard* requirement violation — during the execution as indicated as *zero* utility in Figure 7(e).

By contrast, *compositional* cross-layer optimization in Figure 7(c),(f) presents reasonably close solutions to the *global* approach in the sense that the compositional approach does not introduce undesirable parameter settings that result in a failure, and the average utility resides between that of local and global optimization. The relative execution time of our compositional approach is longer than the local optimization (without any coordination) and shorter than the global approach (most complex). One last remark is that the refined region determined by local optimizers is very different (huge difference in *BufSize*) from that of global approach while our compositional optimization gives similar results.

6 Related Work

The authors of [11, 4] study the issues of cross-layer optimization as a new paradigm for network architecture to make better use of network resources by optimizing the boundaries of traditional network protocol stack layers. Especially, [11] presents a survey of the recent research towards a systematic understanding of "layering" as "optimization decomposition", where the overall communication network is modeled by a generalized network utility maximization problem. Those efforts are, however, mainly focused on the architectural decisions in networking, not tuning the system parameters for QoS-energy optimization.

In [15], the authors extend the idea of predictive learning by exhaustively searching over a limited look-ahead horizon to distributed systems. In particular, they suggest a multi-level decentralized control structure where components have independent local controllers, and the interaction between these components is managed by a global controller that addresses system-wide QoS requirements. Their approach is essentially a *horizontal* (non-layered) composition with a *hierarchy*. Unlike our approach, they exchange the control information through a high level controller that generates *global* control for each local controller. Our work is different in that the composition can be fully distributed and capable of utilizing different even conflicting local objectives through the generic interface of a constraint language.

7 Conclusion

In this paper, we analyzed cross-layer QoS vs energy issues in resource limited devices, and dynamically tuned system parameters. More specifically, we proposed a compositional cross-layer optimization by coordinated interaction among local optimizers through constraint refinement. Our experiments on composition by refinement for treating a realistic multimedia application demonstrated the capability of our approach to generate robust and sufficiently good parameter settings that further can be used as a basis of local optimization. In future work, we will extend our methodology to consider multiple distributed nodes as local optimizers (scalability to horizontal composition). In particular, we plan to construct an interface language for generic composition (e.g., negotiation and contract) and focus on carrying out a large scale demonstration with heterogeneous applications (mission critical, multimedia) on multiple devices in a distributed network.

References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The maude 2.0 system. In *RTA '03*, volume 2706 of *LNCS*, pages 76–87.

[2] http://maude.csl.sri.com.

[3] S. Hua, G. Qu, and S. S. Bhattacharyya. Energy reduction techniques for multimedia applications with tolerance to deadline misses. In *DAC '03*, pages 131–136.

[4] S. Khan, M. Sgroi, E. Steinbach, and W. Kellerer. Cross-layer optimization for wireless video streaming - performance and cost. In *ICME '05*.

[5] M. Kim and S. Ha. Hybrid run-time power management technique for real-time embedded system with voltage scalable processor. In *LCTES/OM '01*, pages 11–19.

[6] M. Kim, H. Oh, N. Dutt, A. Nicolau, and N. Venkatasubramanian. PBPAIR: an energy-efficient error-resilient encoding using probability based power aware intra refresh. *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, 10(3):58–69, 2006.

[7] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. Combining formal verification with observed system execution behavior to tune system parameters. In *FORMATS '07*, volume 4763 of *LNCS*, pages 257–273.

[8] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. A probabilistic formal analysis approach to cross layer optimization in distributed embedded systems. In *FMOODS '07*, volume 4468 of *LNCS*, pages 285–300.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[10] L. Nolle et al. On step width adaptation in simulated annealing for continuous parameter optimisation. In *Proceedings of the 7th Fuzzy Days on Computational Intelligence, Theory and Applications*, pages 589–598, 2001.

[11] M. Chiang et al. Layering as optimization decomposition:a mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, 2007.

[12] J. Meseguer. Conditional Rewriting Logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[13] S. Mohapatra, R. Cornea, H. Oh, K. Lee, M. Kim, N. D. Dutt, R. Gupta, A. Nicolau, S. K. Shukla, and N. Venkatasubramanian. A cross-layer approach for power-performance optimization in distributed mobile systems. In *IPDPS '05*.

[14] Q. Qiu, Q. Wu, and M. Pedram. Dynamic power management in a mobile multimedia system with guaranteed quality-of-service. In *DAC '01*, pages 834–839.

[15] S. Abdelwahed, N. Kandasamy, S. Neema. Online control for self-management in computing systems. In *RTAS '04*, page 368.

[16] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, and R. H. Kravets. Grace-1: Cross-layer adaptation for multimedia quality and battery energy. *IEEE Transactions on Mobile Computing*, 5(7):799–815, 2006.