Semantic-less Coordination of Power Management and Application Performance

Aman Kansal, Jie Liu, Abhishek Singh, and Ripal Nathuji
Microsoft Research
Redmond, WA
[kansal,liuj,absingh,ripaln]@microsoft.com

Tarek Abdelzaher
University of Illinois
Urbana-Champaign, IL
zaher@cs.uiuc.edu

Abstract

A computer system often has multiple power management modules controlling different power knobs. Uncoordinated operation of these knobs not only leads to suboptimal operation but may also cause unsafe behaviors. Coordination methods have thus been proposed to jointly control the power knobs and performance. However, in many systems, such joint design is not feasible due to lack of visibility into all modules to be coordinated. This occurs, for instance, in commodity software that runs on multiple platforms, and emerging cloud hosted applications that operate on platforms outside developers' control and alongside unknown other workloads. We propose an approach for semantics-free coordination where power-performance management can be performed within each module without semantic knowledge regarding other modules.

1 Introduction

The need to reduce computational energy consumption has lead to the development of many power management features across the system stack. Hardware components support multiple active and idle states that enable improved power management, and applications are being increasingly written to scale resource usage and performance. However, in most cases these power management functionalities are not coordinated. Further, the hardware power management is often performed by the system so as not to impact application performance, but without actual visibility into application performance. This can cause problems, such as described in [6, 9, 11], including unstable system behavior leading to power or performance crashes. In particular, the lack of coordination between *application* layer modules that do have visibility into offered performance modes and lower layer *system* modules that typically control the majority of hardware power states, can lead to undesirable operating points.

Joint system and application optimization would thus be more appropriate and has been considered [7, 11]. While the joint methods offer improved and potentially optimal power-performance management, they require communicating semantic information about the behaviors of multiple modules across the system-application boundary. This semantic information is generally hard to obtain and use. For example, an application may be required to run on many different types of servers with varying power management capabilities. It would then be necessary to design the application to recognize the different semantics used by system modules on each server type. Moreover, physical systems may be shared by multiple applications, as is the case for a cloud computing infrastructure or for software from multiple vendors on a single laptop, implying that no single entity can control all knobs.

Consider as an example, an application developer who writes a travel booking website to be hosted on Microsoft's Azure cloud with unknown other applications on unknown hardware. The cloud's power management modules are unaware of application specific details. We need an interface that the cloud may expose to applications such that power management decisions can be coordinated between multiple applications and the underlying platform. We design such an *interface* along with *semantic-less coordination methods* for individual modules at the system and application layers. Semantic-less operation implies that (i) the values shared via the interface cannot be compared to other values, and (ii) a module cannot know, except for its own values, whether a higher or lower value is better. The goal is to **compose** multiple modules, with their independent power-performance management strategies, without resulting in undesirable behavior.

Such semantic-less design poses various challenges, akin to operating a market without conversion rates between currencies. This paper presents our initial thoughts that suggest that semantic-less coordination is indeed possible and exposes some of the related challenges.

Contributions: We propose a semantic-less mechanism, consisting of a narrow data interface and a generic coordination algorithm, for multiple applications and system layer power management modules to coordinate their actions. Only semantic-less numbers, derived from performance and energy, are shared. As an example, an application that can tune its QoS and a system that can change its processor voltage and frequency (P-state) are shown to coordinate without the application knowing anything about system P-states and vice versa. Additionally, multiple applications are shown to compose without causing undesirable behaviors. We illustrate this with a scenario of a utility computing data center that exposes the proposed interface, and multiple third party applications that use it to coordinate power and performance. Finally, we discuss key challenges that arise due to semantic-less operation.

Related Work: The design proposed is partially inspired by blackboard systems [4] in AI, where the analogy is with different domain experts sharing notes on a *blackboard* to jointly solve a large problem that no individual expert can solve. However, our design shares data in a much more constrained interface than blackboard systems, for well-specified coordination.

Coordination of power management controls has been recognized as an important problem and prior works have considered the coordination among system modules [6], applications [3], and among applications and system modules [11, 12, 8, 2]. Joint optimizations of system and application performance have been considered as well [1, 7]. However, these methods assume semantic information about the coordinated entities. We explore an approach that will enable each module to be designed without explicit knowledge regarding others.

2 Power Performance Coordination

An application understands the performance impact of various changes it can make and their effect on quality of service (QoS). For instance, a movie streaming server may know the exact relationship between video resolutions and revenues from rental fees, and the effect on customer loyalty due to reduced frame rates, encoding quality, and number of sound channels. However, the application may not understand how these performance tuning knobs affect resource and energy use. The resource usage may vary depending on processor architecture, cache sizes, disk speeds and so on. The impact on energy may vary depending on hardware power scaling capabilities. It may even depend on additional workload hosted by the server: for instance, for a disk-array, reducing the I/O rate for the application may not reduce energy usage significantly until the disk can be spun down. However, if other applications also need the disk, then reducing the I/O bandwidth of one application may let this disk array be used by more applications, resulting in a denser consolidation and reduced number of active servers.

The system layer on the other hand understands the power scaling knobs and may have models for their impact on energy. However, the system may not know how the settings on those knobs affect applications. For instance, the system may find a transaction processing application's processor utilization to be low and scale down the frequency. This does not affect throughput but causes per transaction latency to increase. The increased latency may cause the application to loose customers, leading to still lower utilization, causing the system to further scale down frequency. The application owners may then increase the number of instances hosting the application to make up for lost performance, causing more servers to be powered on and increasing overall energy usage. In other cases reducing the processor frequency may be required because of a thermal cap. This may render the application useless. For instance, a media server may end up dropping frames and frustrating customers. On the other hand, it may have been possible for the media server to reduce resolution, allowing for a lower processor frequency without losing customers.

Suppose the application can operate at n different performance (QoS) levels, denoted $A_0, A_1, ..., A_{n-1}$, referred henceforth as A-states. For each A-state, suppose the optimal system configuration, including choice of hardware and power management settings, allows serving a constant workload with energy costs shown in Figure 1, for $n = 4$. The polygon $OQ_{max}A_0A_1A_2A_3E_{min}$ represents the feasible energy-performance trade-offs for this application. A real system will operate somewhere within this feasible region, such as denoted by the dashed curve in the figure. Spending more energy than E_{max} does not yield improvements in QoS but may happen due to suboptimal hardware configurations.

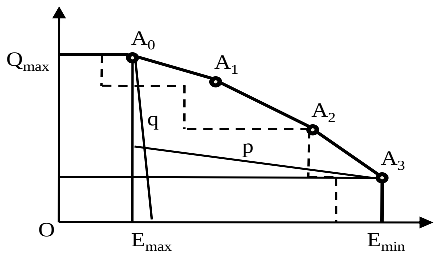

Figure 1. Performance trade-off region.

The key challenge here is to automatically select power management settings that keep the system operating close to the optimal curve. In the absence of proper coordination, such as if the application reduces QoS to save energy, it may operate on line segment q shown in the figure: QoS is dropped without any energy savings as may occur when the hardware does not have power scaling options.On the other hand, if the system scales up power without any insight into application performance, it may operate along line p where the application continues to operate at low QoS because it is bottlenecked on a resource that the system has not scaled up.

We wish to determine a coordinated mapping between application A-states and system power states without exchanging any semantic information.

3 Coordinated System Design

Intuitively, it seems that a key piece of information missing from the system is application layer performance (QoS). We build on this intuition and propose an approach that exposes application QoS without any semantic information regarding what the QoS metric represents. The QoS could be based on throughput, media encoding quality, latency, query processing accuracy or other metric of interest to a specific application.

The proposed approach is based on a constrained data structure, denoted *coordination interface*, shared among the applications and various system power management modules (Figure 2). Multiple applications may share a

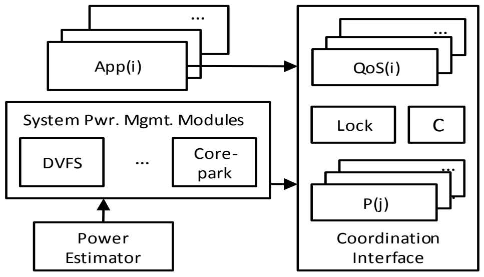

Figure 2. Semantic-less coordination mechanism

common computer system, consisting of multiple components, each with various power control knobs. Coordination is achieved through a common *coordination algorithm* implemented by each coordinating entity.

3.1 Coordination Interface

Each application i publishes its achieved performance $QoS(i)$. QoS may internally consist of complex behavior changes but is externally exposed as a single semantic-less number. $QoS(i)$ is write-able only by application i.

Each system module publishes its power setting, $P(j)$, as a number in the range $\{1,...,n_s\}$, where n_s is the number of power settings for that module. Other modules need not know the meaning of this number (it could be a P-state, throughput cap, sleep mode, etc). A module, however, ensures that each of its settings is represented by a unique number and the same number is published every time that setting is used.

The system also publishes a signal $C \in \{-1,0,1\}$ to indicate if energy needs to be reduced ($C = -1$), is available for increasing performance ($C = 1$), or should be kept constant ($C = 0$). C is write-able only by the system module that understands the constraints on power usage and can decide when there is a need to save power. A system power measurement or estimate derived from performance counters based power models [5, 10], may be used to help determine when power usage needs to be reduced. Applications may voluntarily reduce power usage even when not indicated via C.

A lock object is also shared. Any system module or application that wishes to change a setting first acquires the lock, so that simultaneous uncoordinated changes do not lead to undesirable states. System modules have higher priority for acquiring the lock because if energy can be saved without reducing QoS, that is desirable.

3.2 Coordination Algorithm

The following algorithm allows the various modules to coordinate their actions by only exchanging semantic-less values through the shared data structure. It is assumed for this description that changes in workload are slow and workload is constant during the convergence times of these algorithms.

System Algorithm: This algorithm is run by each system module that controls any power knob. The goal of this algorithm is to reach the lowest power setting that supports currently required performance.

Any system module that controls a power management knob may determine if the resource controlled by it is under-utilized and available for power scaling. It then acquires the lock, and scales down the power to the next lower state. It checks if this has impacted QoS negatively. If yes, the setting is reversed, and the lock is released. If not, it continues the previous step.

Algorithm at each power management module j:

```
1. Set state $M = ACTIVE$.
2. Acquire lock.
3. Change power knob to next lower setting.
4. IF this causes any $QoS(i)$ to change:
   (a) Revert setting to previous value.
   (b) Set $M = INACTIVE$ and release lock.
   ELSE publish new $P(j)$, release lock, and goto Step 2.
5. Monitor shared interface. If any $QoS(i)$ or $P(j)$ values change,
   go to step 1.
```

Different modules may succeed in acquiring the lock in any order and eventually each module reaches a state beyond which it cannot turn its power down without affecting application QoS (system modules avoid change in QoS even if QoS was improved, since they have no semantics as to whether higher or lower QoS value is better). If no module is requesting access to the lock and the system determines that energy usage must be reduced, then it sets $C = -1$. Since no $QoS(i)$ or $P(j)$ values have changed, this does not cause any system power management module to request the lock. Instead, the applications request the lock.

Application Algorithm: An application may continue to operate at its current QoS unless it wishes to save energy (eg., based on cloud hosting fees) or it receives the signal $C = -1$. It then attempts to reduce its resource usage to allow the system to reduce energy usage, as follows.

Algorithm at each application i:

```
1. Continue to check if $C = -1$ or user requested to save power. If
   need to reduce power, and not already in deepest A-state:
   (a) Acquire lock.
   (b) Switch to next lower A-state.
   (c) Publish new $QoS(i)$ and release lock.
```

The action for $C = 1$ is similar except that A-state is changed to increase QoS. After the application releases the lock, the system modules will detect that at least one $QoS(i)$ has changed, and having higher priority on the lock than applications, will attempt to reduce power. The reduced resource utilization due to a change in i's A-state may have reduced power usage to a desired value and the system updates $C = 0$. If not, the applications and system modules continue to contend for the lock and reduce power usage until $C = 0$. To prevent oscillations, the system will not supply $C = +1$

if the previous configuration (the set of $QoS(i)$ and $P(j)$ values published) did not have the target power usage.

An application or system module does not block its ongoing functions when waiting for the lock. Also, it is possible that energy usage is not reduced sufficiently after all applications have scaled QoS to their minimum and then the system has to fall back to non-coordinated mechanisms.

Property: If it is feasible to lower energy use to target level by application QoS tuning, the above algorithm will achieve the reduced QoS state before the system caps resources.

Proof is omitted for brevity. The practical implication is that this approach, without exchanging any semantic information, safely avoids the undesirable behaviors mentioned in section 2, such as violation of latency constraints due to processor frequency reduction at low utilization. The coordination allows achieving better operating points than the non-coordinated operating points along lines p and q in Figure 1. The algorithms are generic so that they can be easily included in any application and system module.

The next section shows prototype implementations of the above approach for some useful coordination scenarios.

4 Experiments

The following experiments illustrate two scenarios: (i) an application that tunes its performance in coordination with the system's P-state control without passing any semantic information about its functionality or learning about the nature of system's power management capability, and (ii) multiple applications coordinate their resource usage on a shared platform without any explicit knowledge about the other applications' functionality and performance trade-offs.

Experiment 1: Consider a battery operated laptop decoding high definition video. A common operation used in media decoding is the Discrete Fourier Transform (DFT). Suppose the application can perform DFT computations to varying degrees of accuracy, and the perceived QoS varies with the inverse of the logarithmic error. This yields 4 A-states (Table 1) where the error falls as we consider successively higher order terms in the cosine expansion. The QoS is zero when the data cannot be decoded at the required frame rate (this occurs at lower processor frequencies if using high precision). The laptop, a Lenovo T61p, has the P-states (known only to the system) and power usage shown in Table 2, measured using a WattsUp Pro power meter. Suppose the battery is running low and the system desires to reduce power usage. A first possibility, entirely system managed and in current usage, is that the OS lowers the processor frequency. A second possibility is application managed: the application reduces its QoS hoping to save power but without the OS changing the P-state. The third, is a coordinated strategy (Section 3). The three options are compared in Figure 3. In this simple case, the coordination strategy achieves the optimal power settings, though this need not always be the case.

State	A_0	A_1	A_2	A_3
QoS	11.60	6.38	5.37	3.42

Table 1. Application QoS

P-State (MHz)	2200	1600	1200	800
Idle (W)	34.06	28.59	26.28	25.08
Active (W)	47.78	37.32	29.45	26.18

Table 2. System P-states.

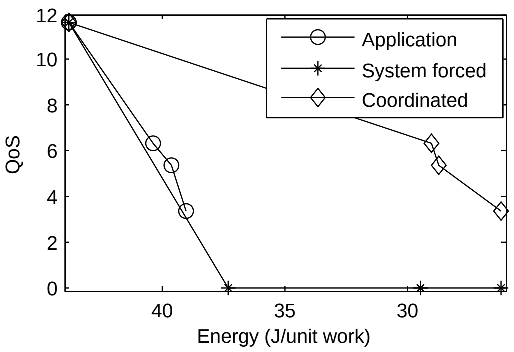

Figure 3. Coordination for a single application on one machine.

Experiment 2: This experiment considers multiple applications with different functionalities sharing a server. Suppose one application is a stream server that can serve videos at HD resolution (typically 3.2Mbps), DVD equivalent (2Mbps), broadband quality (300kbps) and dial-up quality (28kbps). Suppose the application's revenues (QoS levels) are \$4, \$3, \$2, and \$0.5 respectively in these A-states. Suppose another application is serving search results over an in-memory index of flight schedules. Varying the quality of search causes varying CPU usage: 100%, 75%, 50%, and 25%, and the conversion of searches to purchases varies with search quality leading to varying revenues of \$6, \$5, \$4, and \$1 respectively. Power measurements for operations in different A-state combinations are performed on an HP DLG380 blade server with 2x4-core Xeon processors and an 8-disk RAID array.

Note that while we have selected both QoS metrics in the same units for purposes of a comparison with a semantics based algorithm, the QoS metrics are not necessarily comparable unless we reveal the semantics that they are all measured in revenues. The semantics based optimal algorithm performs a joint optimization, intelligently reducing the A-state of the application that would maximize the combined revenue for given energy use depending on system power characteristics. Optimal coordination can be practically achieved under certain conditions (convexity of QoS curves, etc.) if common semantics are assumed, such as using bidding mechanisms [3]. The semantic-less approach has no notion of a combined

QoS, and is executed by each application without any knowledge of other applications or system power management capabilities.

Figure 4 plots the power-QoS behavior assuming the system was supplying a $C = -1$ signal to reduce energy use. Lock acquisition in the semantic-less approach is random and different runs may lead to different behaviors: 20 such runs are plotted along with their average. Server idle power is excluded from the power numbers. The semantic-less algorithm is clearly not optimal but is able to avoid undesirable behaviors such as QoS crashes due to blind resource capping by the OS.

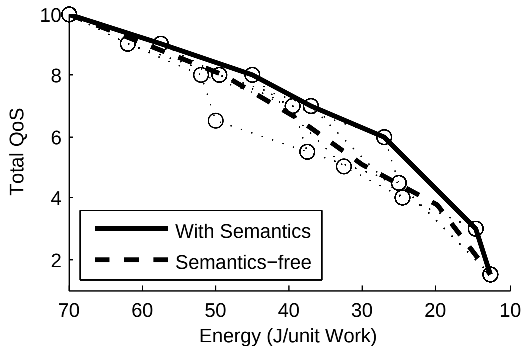

Figure 4. Comparing semantics free (dotted lines are multiple random runs and dashed curve their average) and optimal (solid curve).

5 Discussion

The semantic-less approach proposed in this paper has several opportunities for improvement. Some of these, such as limiting the number of power knob changes, avoiding oscillations around desired power levels, eliminating the use of a lock through simultaneous small QoS changes, etc. can be addressed in the semantic-less domain but are omitted for brevity. Here, we discus some of the more significant challenges in improving the algorithm's performance.

Optimality: Clearly, the semantic-less approach may not achieve optimal operation. In fact, the definition itself of optimal operation requires semantics, such as a method to measure the combined QoS of applications based on their relative importance. Limited semantic information, such as a common currency to measure QoS, may enable the coordinated modules to reach joint optima, such as by exploring various possible configurations [1] and learning good configurations among those attained over time. Further, it would be interesting to explore if the loss of performance due to semantic-less operation can be bounded. The bounds may depend on scale factors that characterize the relative weights of the QoS from various applications as well as the relative energy impact of multiple power knobs.

Convergence Time: While the coordination mechanism is guaranteed to converge (all entities quit requesting the lock), an important concern in practical deployments is the convergence time. For a small number of applications and system modules this time may be small compared to the time over which thermal or power caps and workloads change. But as the number of coordinated entities grows, this time becomes large and external factors may change before the coordination mechanism has converged. Heuristics can be designed to speed up the convergence, such as incorporating memory in each module or the shared data interface to store the QoS and energy settings that have been explored before, thus reducing the the number of state changes attempted. Provably safe mechanisms to ensure stable operation are required.

All applications were assumed independent. Interdependence among applications may affect resource usage and combined QoS in complex ways. Sharing of semantic information among interdependent modules may thus seem appropriate. It is worth exploring if such interdependence can cause systems using the semantic-less approach to end up in undesirable configurations.

6 References

[1] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Automatic exploration of datacenter performance regimes. In *First Workshop on Automated Control for Datacenters and Clouds (ACDC09)*, Barcelona, Spain, June 2009.

[2] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *IEEE Real-Time Systems Symposium*, December 1998.

[3] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.*, 35(5):103–116, 2001.

[4] D. D. Corkill. An introduction to blackboard systems. *AI Expert*, 6(9):40–47, September 1991.

[5] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2007.

[6] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *IEEE International Real-Time Systems Symposium*, pages 227–238, 2007.

[7] X. Liu, P. Shenoy, and M. D. Corner. Chameleon: Application Level Power Management. *IEEE Transactions on Mobile Computing*, 7(8):995–1010, August 2008.

[8] R. Nathuji, P. England, P. Sharma, and A. Singh. Feedback driven qos-aware power budgeting for virtualized servers. In *Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, San Francisco, CA, USA, April 2009.

[9] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multi-level power management for the data center. *SIGOPS Oper. Syst. Rev.*, 42(2):48–59, 2008.

[10] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. In *HotPower'08: Workshop on Power Aware Computing and Systems*, December 2008.

[11] X. Wang and Y. Wang. Co-con: Coordinated control of power and application performance for virtualized server clusters. In *17th IEEE International Workshop on Quality of Service (IWQoS)*, Charleston, South Carolina, July 2009.

[12] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *SOSP*, October 2003.