Dynamic Knobs for Responsive Power-Aware Computing

Henry Hoffmann * Stelios Sidiroglou* Michael Carbin Sasa Misailovic
Anant Agarwal Martin Rinard

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{hank,stelios,mcarbin,misailo,agarwal,rinard}@csail.mit.edu

Abstract

We present PowerDial, a system for dynamically adapting application behavior to execute successfully in the face of load and power fluctuations. PowerDial transforms static configuration parameters into *dynamic knobs* that the PowerDial control system can manipulate to dynamically trade off the accuracy of the computation in return for reductions in the computational resources that the application requires to produce its results. These reductions translate directly into performance improvements and power savings.

Our experimental results show that PowerDial can enable our benchmark applications to execute responsively in the face of power caps that would otherwise significantly impair responsiveness. They also show that PowerDial can significantly reduce the number of machines required to service intermittent load spikes, enabling reductions in power and capital costs.

Categories and Subject Descriptors C.4 [*Performance of Systems*]: Reliability, availability, and serviceability

General Terms Performance, Reliability, Experimentation

Keywords Accuracy-aware Computing, Power-aware Computing, Self-aware Systems

1. Introduction

Many applications exhibit a trade-off between the accuracy of the result that they produce and the power and/or time that they require to produce that result. Because an application's optimal operating point can vary depending on characteristics of the environment in which it executes (for example, the delivered computational capacity of the underlying computing platform), developers often provide a static interface (in the form of configuration parameters) that makes it possible to choose different points in the trade-off space for different executions of the application. Configured at startup, the application operates at the selected point for its entire execution.

But phenomena such as load fluctuations or variations in available power can change the optimal operating point of the application as it is executing. Static configuration leaves the application with two unappealing choices: either continue its execution at a suboptimal point in the trade-off space (potentially impairing properties such as responsiveness to users) or terminate its current execution and restart at a new operating point (and incur service interruptions as it drops its current task and restarts).

1.1 Dynamic Knobs and Adaptive Response

We present a new system, PowerDial, for dynamically adapting the behavior of running applications to respond to fluctuations in load, power, or any other event that threatens the ability of the underlying computing platform to deliver adequate capacity to satisfy demand:

- **Dynamic Knob Insertion:** PowerDial uses dynamic influence tracing to transform static application configuration parameters into dynamic control variables stored in the address space of the running application. These control variables are made available via a set of *dynamic knobs* that can change the configuration (and therefore the point in the trade-off space at which it executes) of a running application without interrupting service or otherwise perturbing the execution.
- **Dynamic Knob Calibration:** PowerDial explores the underlying accuracy versus performance trade-off space (originally available via the configuration parameters) to characterize the accuracy and performance of each dynamic knob setting. It uses a quality of service (QoS) metric to quantify the accuracy of each setting.
- **Dynamic Knob Control:** PowerDial is designed for applications that are deployed to produce results at a target frequency (with performance measured as the time between results). It uses the Application Heartbeats framework [24] to dynamically monitor the application. An existing control strategy [35] is combined with a novel actuation mechanism to maintain performance. When the performance either drops below target (i.e., the time between results exceeds a given threshold) or rises above target (i.e., the time between results drops below the threshold), the PowerDial system uses the calibrated dynamic knobs to move the application to a more appropriate point in its trade-off space (the new point may, for example, give up some accuracy in return for increased performance and decreased power consumption). The goal is to maximize accuracy while preserving responsiveness in the face of fluctuations in the capabilities of the underlying computing platform.

1.2 Summary of Experimental Results

We evaluate PowerDial's ability to control the behavior of four benchmark applications (the x264 video encoder, the bodytrack human tracking application, the swaptions financial analysis application, and the swish++ search engine) dynamically in environments with fluctuating load and power characteristics. Our results show:

* Henry Hoffmann and Stelios Sidiroglou contributed equally to the research presented in this paper.

ASPLOS'11, March 5–11, 2011, Newport Beach, California, USA.

- **Trade-Off Space:** All of the applications exhibit a large viable trade-off space — three of the applications (x264, bodytrack, and swaptions) can execute from four to six times faster than their baseline (which defines the default quality of service) with acceptable quality of service losses. swish++ can execute approximately 1.5 times faster than its baseline (at the cost of dropping lower-ranked search results).
- **Power Capping:** Systems often respond to power caps (reductions in the delivered power imposed, for example, in response to cooling system failures) by dynamic voltage/frequency scaling (DVFS) (reducing the frequency and voltage at which the system operates [51]). The ensuing reduction in the delivered computational capacity of the system can make it difficult or impossible for applications to continue to deliver responsive service.

  Our results show that PowerDial enables applications to adapt effectively as a power cap (which reduces the processor frequency from 2.4 GHz to 1.6 GHz) is first imposed, then lifted. When the power cap is imposed, PowerDial preserves responsiveness by moving the applications to new Pareto-optimal points with less computational demands and slightly lower quality of service. When the power cap is lifted, PowerDial restores the original quality of service by moving the applications back to the baseline.
- **Peak Load Provisioning:** Systems are often provisioned to service the peak anticipated load. Common workloads often contain intermittent load spikes [9]. The system therefore usually contains idle machines that consume power but perform no useful work.

  Our results show that PowerDial can reduce the number of machines required to successfully service time-varying workloads. When a load spike overwhelms the ability of the system to service the load with the baseline application configuration, PowerDial preserves responsive performance by dynamically reconfiguring the application to use less computation to produce (slightly) lower quality results. Specifically, our results show that PowerDial can make it possible to reduce (by a factor of 3/4 for x264, bodytrack, and swaptions and by a factor of 1/3 for swish++) the number of machines required to provide responsive service in the face of intermittent load spikes. The system provides baseline quality of service for the vast majority of tasks; during peak loads, the system provides acceptable quality of service and (at most) negligible performance loss.

PowerDial is not designed for all applications — it is instead designed for applications that 1) have viable performance versus QoS trade-off spaces and (as is evident in the availability of appropriate configuration parameters) have been engineered to operate successfully at multiple points within those spaces and 2) operate in contexts where they must satisfy responsiveness requirements even in the face of fluctuations in the capacity of the underlying computing platform. In this paper we focus on fluctuations in power and load, but PowerDial can enable applications to adapt dynamically to any change that affects the computational capacity delivered to the application.

1.3 Contributions

This paper makes the following contributions:

- **Dynamic Knobs:** It presents the concept of dynamic knobs, which manipulate control variables in the address space of a running application to dynamically change the point in the underlying performance versus quality of service trade-off space at which the application executes.
- **PowerDial:** It presents PowerDial, a system that transforms static configuration parameters into calibrated dynamic knobs and uses the dynamic knobs to enable the application to operate successfully in the face of fluctuating operating conditions (such as load spikes and power fluctuations).
- **Analysis and Instrumentation:** It presents the PowerDial analysis and instrumentation systems, which dynamically analyze the application to find and insert the dynamic knobs.
- **Control:** It presents the PowerDial control system, which uses established control techniques combined with novel actuators to automatically maintain the application's desired performance while minimizing quality of service loss.
- **Resource Requirements:** It shows how to use dynamic knobs to reduce the number of machines required to successfully service peak loads and to enable applications to tolerate the imposition of power caps. It analyzes the resulting reductions in the amount of resources required to acquire and operate a computational platform that can successfully deliver responsive service in the face of power and load fluctuations.
- **Experimental Results:** It presents experimental results characterizing the trade-off space that dynamic knobs make available in our benchmark applications. It also presents results demonstrating PowerDial's ability to enable automatic, dynamic adaptation of applications in response to fluctuations in system load and power.

2. Dynamic Knobs

Dynamic knobs are designed for applications that 1) have static configuration parameters controlling performance versus QoS trade-offs and 2) use the Application Heartbeats API [24] (our system can automatically insert the required API calls, see Section 2.3). These applications typically exhibit the following general computational pattern:

- **Initialization:** During initialization the application parses and processes the configuration parameters, then computes and stores the resulting values in one or more control variables in the address space of the running application.
- **Main Control Loop:** The application executes multiple iterations of a main control loop. At each iteration it emits a heartbeat, reads the next unit of input, processes this unit, produces the corresponding output, then executes the next iteration of the loop. As it processes each input unit, it reads the control variables to determine which algorithm to use.

With this computational pattern, the point in the performance versus QoS trade-off space at which the application executes is determined by the configuration parameters when the application starts and does not change during its execution. A goal of PowerDial, as illustrated in Figure 1, is to augment the application with the ability to dynamically change the point in the trade-off space at which it is operating. At a high level, PowerDial accomplishes this goal as follows:

- **Parameter Identification:** The user of the program identifies a set of configuration parameters and a range of settings for each such parameter. Each combination of parameter settings corresponds to a different point in the performance versus QoS trade-off space.
- **Dynamic Knob Identification:** For each combination of parameter settings, PowerDial uses dynamic influence tracing (which traces how the parameters influence values in the running application) to locate the control variables and record the values stored in each control variable.
- **Dynamic Knob Calibration:** Given a set of representative inputs and a QoS metric, PowerDial executes a training run for each input and combination of parameter settings. For each training run it records performance and QoS information. It then processes this information to identify the Pareto-optimal points in the explored performance versus QoS trade-off space.

- **Dynamic Knob Insertion:** PowerDial inserts callbacks that the PowerDial control system can use to set the control variables to values previously recorded during dynamic knob identification, thereby moving the application to a different Pareto-optimal point in the performance versus QoS trade-off space. Subsequent iterations of the main control loop will read the updated values in the control variables to (in effect) process further input as if the configuration parameters had been set to their corresponding different settings at application startup.

The result is an application that enables the PowerDial control system to dynamically control the point in the performance versus QoS trade-off space at which the application executes. In standard usage scenarios PowerDial is given a target heart rate. If the application's dynamically observed heart rate is slower than the target heart rate, PowerDial uses the calibrated dynamic knobs to move the application to a new point in the trade-off space with higher performance at the cost, typically small, of some QoS. If the observed heart rate is higher than the target, PowerDial moves the application to a new point with lower performance and better QoS.

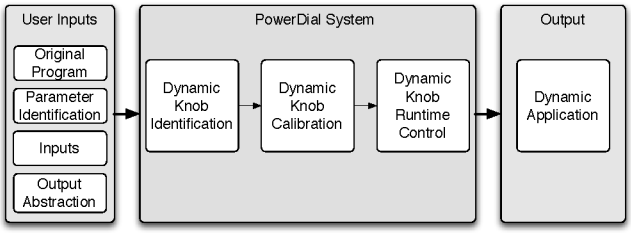


Figure 1: Dynamic Knob work flow.

2.1 Dynamic Knob Identification

For PowerDial to transform a given set of configuration parameters into dynamic knobs, it must identify a set of control variables that satisfy the following conditions:

- **Complete and Pure:** All variables whose values are derived from configuration parameters during application startup (before the application emits its first heartbeat) are control variables. The values of control variables are derived only from the given set of configuration parameters and not from other parameters.
- **Relevant and Constant:** During executions of the main control loop, the application reads but does not write the values of the control variables.

PowerDial uses influence tracing [11, 19] to find the control variables for the specified configuration parameters. For each combination of configuration parameter settings, PowerDial executes a version of the application instrumented to trace, as the application executes, how the parameters influence the values that the application computes. It uses the trace information to find the control variables and record their values, applying the above conditions as follows:

- **Complete and Pure Check:** It finds all variables that, before the first heartbeat, contain values influenced by the specified configuration parameters. It checks that these values are influenced only by the specified configuration parameters.
- **Relevance Check:** It filters out any variables that the application does not read after the first heartbeat — the values of these variables are not relevant to the main control loop computation.
- **Constant Check:** It checks that the execution does not write a control variable after the first heartbeat.

Finally, PowerDial checks that the control variables are *consistent*, i.e., that the different combinations of parameter settings all produce the same set of control variables. If the application fails any of these checks, PowerDial rejects the transformation of the specified configuration parameters into dynamic knobs.

For each combination of parameter settings, PowerDial records the value of each control variable. The PowerDial control system uses this information to automatically change the values of the control variables at runtime. Note that because PowerDial uses a dynamic influence analysis to find the control variables, it is possible for unexercised execution paths to violate one or more of the above conditions. The influence analysis also does not trace indirect control-flow or array index influence. To enable a developer to (if desired) check that neither of these potential sources of imprecision affects the validity of the control variables, PowerDial produces a control variable report. This report lists the control variables, the corresponding configuration parameters from which their values are derived, and the statements in the application that access them. We have examined the reports for all of our benchmark applications (see Section 4) and verified that all of the automatically computed control variables are valid.

Our influence tracing system is implemented as a static, source-based instrumentor for C and C++. It is built on the LLVM compiler framework [11, 30] and inserts code to trace the flow of influence through the values that the application computes. For each value, it computes the configuration parameters that influenced that value. The currently implemented system supports control variables with datatypes of int, long, float, double, or STL vector. It augments the production version of the application with calls to the PowerDial control system to register the address of each control variable and read in the previously recorded values corresponding to the different dynamic knob settings. This mechanism gives the PowerDial control system the information it needs to apply a given dynamic knob setting.

2.2 Dynamic Knob Calibration

In this step, PowerDial explores the performance versus QoS trade-off space available to the application via the specified configuration parameters. The user provides an application, a set of representative inputs, a set of specified configuration parameters, a range of values for each parameter, and a QoS metric. Given these values, PowerDial produces, for each combination of parameter settings, a specification of the point in the trade-off space to which the parameter settings take the application. This point is specified relative to the baseline performance and QoS of the parameter setting that delivers the highest QoS (which, for our set of benchmark applications, is the default parameter setting).

The PowerDial calibrator executes all combinations of the representative inputs and configuration parameters. For each parameter combination it records the mean (over all representative inputs) speedup of the application. It computes the speedup as the execution time of the application running with the default parameter settings divided by the execution time of the application with the current parameter combination. In a separate instrumented execution, it also records the values of the control variables (see Section 2.1).

For each combination of configuration parameters PowerDial also records the mean (over all representative inputs) QoS. The QoS metric works with a user-provided, application-specific *output abstraction* which, when provided with an output from the program, produces a set of numbers $o_1, \ldots, o_m$. The output abstraction typically extracts relevant numbers from the output or computes a measure of output quality (such as, for example, the peak signal-to-noise ratio of the output). Given the output abstraction from the baseline execution $o_1, \ldots, o_m$ and an output abstraction $\hat{o}_1, \ldots, \hat{o}_m$ from the execution with the current parameter settings, we compute

the QoS loss as the *distortion* [43]:

$$qos = \frac{1}{m}\sum_{i=1}^{m} w_i \left|\frac{o_i - \hat{o}_i}{o_i}\right| \quad (1)$$

Here each weight w_i is optionally provided by the user to capture the relative importance of the ith component of the output abstraction. Note that a qos of zero indicates optimal QoS, with higher numbers corresponding to worse QoS. PowerDial supports caps on QoS loss — if a specific parameter setting produces a QoS loss that exceeds a user-specified bound, the system can exclude the corresponding dynamic knob setting from further consideration.

2.3 The PowerDial Control System

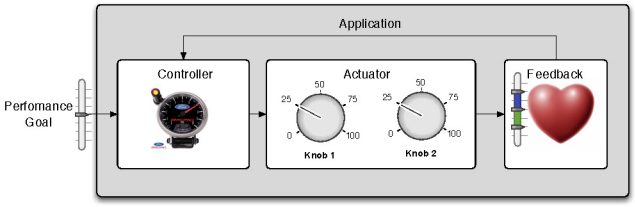


Figure 2: The PowerDial Control System.

The PowerDial control system automatically adjusts the dynamic knobs to appropriately control the application. As shown in Figure 2, the PowerDial control system contains the following components: a feedback mechanism that allows the system to monitor application performance, a control component which converts the feedback into a desired speedup, and an actuator which converts the desired speedup into settings for one or more dynamic knobs.

2.3.1 Feedback Mechanism

PowerDial uses the Application Heartbeats framework as its feedback mechanism [24]. In general, PowerDial can work with any application that has been engineered using this framework to emit heartbeats at regular intervals and express a desired performance in terms of a target minimum and maximum heart rate. For our set of benchmark applications, the PowerDial instrumentation system automatically inserts the API calls that emit heartbeats — it profiles each application to find the most time-consuming loop (in all of our applications this is the main control loop), then inserts a heartbeat call at the top of this loop. In general, the PowerDial control system is designed to work with any target minimum and maximum heart rate that the application can achieve. For our experiments (see Section 4), the minimum and maximum heart rate are both set to the average heart rate measured for the application using the default configuration parameters.

We use Application Heartbeats as a feedback mechanism for several reasons. First, this interface provides direct measurement of a specific application's performance, obviating the need to infer this measure from low-level metrics such as hardware performance counters or high-level metrics such as total system throughput. Second, Application Heartbeats are applicable to a variety of applications as they are more general than other common metrics such as flop rate (which is not designed to measure the performance of integer applications) and instructions per clock (which is not designed to measure the true performance of I/O bound applications or applications lacking instruction-level parallelism). Third, the interface is simple and can easily be customized by users who want to control the placement of heartbeat calls or the target performance. Finally, heartbeat-enabled applications can be controlled using a general and extensible control framework.

2.3.2 Control Strategy

PowerDial employs an existing, general control strategy which can work with any application that uses Application Heartbeats as a feedback mechanism [35]. This controller monitors the heartbeat data and determines both when to speed up or slow down the application as well as how much speedup or slowdown to apply. As described in [35], the control system models application performance as

$$h(t+1) = b \cdot s(t) \quad (2)$$

where $h(t)$ is the application's heart rate at time t, b is the speed of the application when all knobs are set to their baseline values, and $s(t)$ is the speedup applied at time t.

Given this model of application behavior, the control system computes the speedup $s(t)$ to apply at time t as

$$e(t) = g - h(t) \quad (3)$$

$$s(t) = s(t-1) + \frac{e(t)}{b} \quad (4)$$

where $e(t)$ is the error at time t and g is the target heart rate.

We demonstrate several properties of the steady-state behavior for the system defined in Equations 2–4. First, we show that the system converges, i.e., the performance reaches the desired heart rate, g. Second, we show that it is stable and does not oscillate around the desired heart rate. Third we show that the convergence time is bounded and fast.

To demonstrate these properties, we use standard control theoretic techniques and analyze the system in the Z-domain [23]. To begin, we compute the Z-transforms $F(z)$ and $G(z)$ of the systems defined in Equations 2 and 4, respectively as[1]:

$$G(z) = \frac{H(z)}{S(z)} = \frac{b}{z} \quad (5)$$

$$F(z) = \frac{S(z)}{E(z)} = \frac{z}{b(z-1)} \quad (6)$$

Using Equations 5 and 6 we compute the transfer function $F_{loop}(z)$ of the closed-loop system which drives the current heart rate, $h(t)$, to the target heart rate g as

$$F_{loop}(z) = \frac{F(z)G(z)}{1 + F(z)G(z)} \quad (7)$$

$$F_{loop}(z) = \frac{1}{z} \quad (8)$$

We use standard techniques [23, 35] to demonstrate the desired properties of the system whose Z-transform is given by Equation 8. To show that the system converges, we show that it has a unit steady state gain, i.e., $F_{loop}(1) = 1$. Clearly, when $z = 1$, $F_{loop}(z) = 1$, so the system converges to the desired heart rate. To demonstrate that the system is stable and does not oscillate, we must ensure that the poles[2] of $F_{loop}(z)$ are less than 1. $F_{loop}(z)$ has a single pole p at 0, so it is stable and will not oscillate. The time t_c required for the system to converge is estimated as $t_c \approx -4/\log(|p_d|)$, where p_d is the dominant pole of the system [23]. As the only pole for $F_{loop}(z)$ is $p = 0$, the system will converge almost instantaneously. In practice, the convergence time will be limited by a time quantum used to convert the continuous speedup signal into a discrete knob setting as described in the next section.

[1] A Z-transform converts an infinite list of values (e.g. $s(t)$ for all $t \geq 0$) into an infinite sum. In many cases, this infinite sum has a compact representation which is easier to analyze than the infinite list representation.

[2] A pole is a point p such that $\lim_{z \to p} F(z) = \infty$.

2.3.3 Actuation Policy

The PowerDial actuator must convert the speedup specified by the controller into a dynamic knob setting. The controller is a continuous linear system, and thus, the actuator must convert the continuous signal into actions that can be realized in the application's discrete, potentially non-linear, dynamic knob system. For example, the controller might specify a speedup of 1.5 while the smallest speedup available through a knob setting is 2. To resolve this issue, the actuator computes a set of actions to take over a time quantum. We heuristically establish the time quantum as the time required to process twenty heartbeats. In the example, the actuator would run with a speedup of 2 for half the time quantum and the default speedup of 1 for the other half. Therefore, the average speedup over the time quantum is 1.5, the desired value.

The actuator determines which actions to take for the next time quantum by optimizing a system of linear constraints. Let b be the heart rate in the baseline configuration, while g is the target heart rate of the system. Let s_{max} be the maximum achievable speedup for the application given its dynamic knobs, and let s_{min} be the minimum speedup corresponding to a knob setting such that $s_{min} \geq g/b$. Let unknowns t_{max}, t_{min}, and $t_{default}$ correspond to the percentage of time during the next quantum to run with the application's knobs set to the maximum speedup, the minimum required speedup, and the default settings, respectively. The following system of constraints captures the behaviors considered for the next time quantum.

$$s_{max} \cdot t_{max} + s_{min} \cdot t_{min} + \frac{b}{g} \cdot t_{default} = 1 \quad (9)$$

$$t_{max} + t_{min} + t_{default} \leq 1 \quad (10)$$

$$t_{max}, t_{min}, t_{default} \geq 0 \quad (11)$$

While there are many solutions to this system of constraints, two are of particular interest for making power versus performance versus QoS trade-offs. First, for platforms with sufficiently low idle power consumption (for more detail see Section 3), PowerDial supports *race-to-idle* execution by setting $t_{min} = t_{default} = 0$, which forces the application to run at the highest available speedup. If $t_{max} < 1$ the system can idle for the remaining $1 - t_{max}$ portion of the time quantum to save power. The second solution PowerDial considers results from setting $t_{max} = 0$ and requiring $t_{min} + t_{default} = 1$. This solution will run the application at the lowest obtainable speedup that will enable the application to meet its heart rate target and delivers the lowest feasible QoS loss. Having determined values for t_{max}, t_{min}, and $t_{default}$ for the next time quantum, the PowerDial controller executes the corresponding plan, then computes a new plan when the quantum expires.

3. Analytical Models

Data center power consumption is experiencing significant growth. By 2011, U.S. data centers are predicted to use 100 Billion kWh, at a cost of $7.4 billion per year [50]. Of particular concern is the low average data center utilization, typically around 20-30% [9, 37], which coupled with high idle power consumption (at idle, current servers use about 60% of peak power), leads to significant waste.

The combination of brief but frequent bursts of activity with latency requirements results in underutilized machines remaining online. Server consolidation through the use of virtual machines, commonly used for non-critical services, cannot react quickly enough to maintain the desired level of service [37]. Turning idle systems off (even in low power mode), has similar problems.

To deal with idle power waste, researchers have proposed that system components be designed to consume energy proportional to their use [9]. Dynamic voltage and frequency scaling (DVFS) is a power management technique commonly found in modern processors [1, 3] that demonstrates this concept.

Beyond direct energy costs, data centers also incur capital costs (e.g. power provisioning, cooling, etc.,). Over the lifetime of the facility, these capital costs may exceed energy costs. To reduce such costs, researchers have proposed techniques that aim to operate facilities as close to maximum power capacity as possible, sometimes guaranteeing availability using various forms of *power capping* [18, 31, 41]. Power capping throttles server performance during utilization spikes to ensure that power budgets are satisfied. As a consequence of power capping, applications may experience increased latency due to the lower operating frequency. This increased latency may violate latency service level agreements.

We next present several analytical models that characterize the effectiveness of dynamic knobs in enabling applications to respond to dynamic voltage/frequency scaling (caused, for example, by the imposition or lifting of power caps) and in reducing the number of machines required to maintain responsiveness in the face of intermittent load spikes.

DVFS and Dynamic Knobs: Figure 3 shows how operating at lower power states can enable systems to reduce power consumption at the cost of increased latency. The area within the boxes represents the total energy required to complete a workload. For a task which takes time t and consumes average power of P_{avg}, the total energy can be calculated as: $E_{task} = P_{avg} \cdot t$. Without DVFS (Figure 3 (a)), the workload consumes power P_{nodvfs} for time t_1 and power P_{idle} for the remaining time t_{delay}. With DVFS (Figure 3 (b)), the consumed power is reduced to P_{dvfs} but the execution time increases to $t_2 = t_1 + t_{delay}$. To accurately calculate DVFS energy savings, the idle power consumed by the non-DVFS system (P_{idle}) must be included. Thus the energy savings due to DVFS can be computed as:

$$E_{dvfs} = (P_{nodvfs} \cdot t_1 + P_{idle} \cdot t_{delay}) - (P_{dvfs} \cdot t_2) \quad (12)$$

For CPU-bound applications, t_2 can be predicted by the change in operating frequency as: $t_2 = \frac{f_{nodvfs}}{f_{dvfs}} \cdot t_1$. We note that any power savings here come at the cost of added latency.

Dynamic knobs can complement DVFS by allowing systems to save power by reducing the amount of computational resources required to accomplish a given task. There are two cases to consider depending on the idle power of the system P_{idle} as illustrated in Figure 4. Figure 4(a) illustrates the first case. This case applies to systems with low idle power consumption (i.e., small P_{idle}). In this case, the best energy savings strategy is to complete the task as quickly as possible, then return to the low-power idle state, a strategy known as *race-to-idle*. Dynamic knobs can facilitate race-to-idle operation by decreasing the amount of computational resources required to complete the task (in return for some QoS loss), thereby reducing t_1. Figure 4(b) illustrates the second case, which applies to systems with high idle power consumption (i.e., large P_{idle}), common in current server class machines. In this case, dynamic knobs can allow the system to operate at a lower power state for the time t_2 allocated to complete the task.

In both cases the energy savings available through combining DVFS and dynamic knobs can be calculated as:

$$t'_1 = \frac{t_1}{S(QoS)}, \quad t'_{delay} = t_{delay} + t_1 - \frac{t_1}{S(QoS)} \quad (13)$$

$$E_1 = P_{nodvfs} \cdot t'_1 + P_{idle} \cdot t'_{delay} \quad (14)$$

$$t'_2 = \frac{t_2}{S(QoS)}, \quad t''_{delay} = t_2 - \frac{t_2}{S(QoS)} \quad (15)$$

$$E_2 = P_{dvfs} \cdot t'_2 + P_{idle} \cdot t''_{delay} \quad (16)$$

$$E_{elasticdvfs} = \min(E_1, E_2) \quad (17)$$

$$E_{dvfs} = \min(P_{nodvfs} \cdot t_1 + P_{idle} \cdot t_{delay}, P_{dvfs} \cdot t_2) \quad (18)$$

$$E_{savings} = E_{dvfs} - E_{elasticdvfs} \quad (19)$$

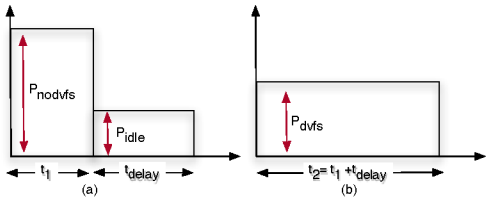


Figure 3: Power Consumption with DVFS

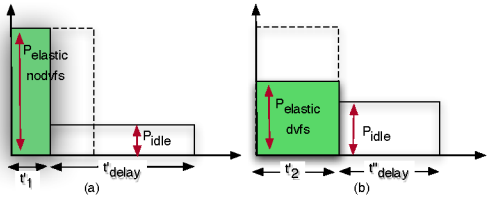


Figure 4: Power Consumption with DVFS and Dynamic Knobs

where $S(QoS)$ represents the speedup available as a function of acceptable QoS loss (i.e., the desired level of accuracy). In the power cap scenario discussed in Section 5.4, $t_{delay} = 0$ and the control objective is to find a point that maximizes delivered quality of service for a given constant t_2.

Server Consolidation: Dynamic knobs can also enable power savings by reducing the total number of machines required to meet peak load without increasing latency. This is achieved by increasing the throughput of each individual machine (by trading QoS for performance) thus reducing the number of machines that are needed to service peak demand. The resulting reduction in computational resources required to service the load enables the system to service the load spike without increasing the service time. The reduction in the number of machines improves server utilization during normal operation and reduces energy wasted on idle resources. It can also provide significant extra savings in the form of reduced cooling costs.

To quantify how dynamic knobs can help improve server consolidation, we need to examine the total work required to meet system peak load requirements. This can be calculated as follows:

$$W_{total} = (W_{machine} \cdot N_{orig}) \tag{20}$$

W_{total} represents the total work, where $W_{machine}$ represents the work per machine and N_{orig} is the total number of machines. Let $S(QoS)$ be the speedup achieved as a function of the reduction in QoS. For the default QoS (obtained with the default knob settings) the speedup is 1. The number of machines required to meet peak load with some loss of accuracy can be computed as:

$$N_{new} = \left\lceil \frac{W_{total}}{S(QoS)} \cdot \frac{1}{W_{machine}} \right\rceil \tag{21}$$

Here N_{new} is the new, lower number of machines that can be used to meet peak load requirements after consolidation. To measure the savings achieved by this consolidation let $U_{new} = N_{orig}/N_{new}$ be the average utilization of the consolidated system and U_{orig} be the average utilization in the original system. Further assume that the two systems both use machines that consume power P_{load} under load and P_{idle} while idle. Let P_{orig} be the average power in the original system, while P_{new} is the average power in the smaller consolidated system. Then we can calculate the average power savings of the consolidated system as:

$$P_{orig} = N_{orig}(U_{orig} \cdot P_{load} + (1 - U_{orig})P_{idle}) \tag{22}$$

$$P_{new} = N_{new}(U_{new} \cdot P_{load} + (1 - U_{new})P_{idle}) \tag{23}$$

$$P_{save} = P_{orig} - P_{new} \tag{24}$$

This power savings can reduce both the power required to operate the system and indirect costs such as cooling and conversion costs. It can also reduce the number of machines and therefore the capital cost of acquiring the system.

4. Benchmarks and Inputs

We report results for four benchmarks. swaptions, bodytrack, and x264 are all taken from the PARSEC benchmark suite [10]; swish++ is as open-source search engine [48]. For each application we acquire a set of representative inputs, then randomly partition the inputs into *training* and *production* sets. We use the training inputs to obtain the dynamic knob response model (see Section 2) and the production inputs to evaluate the behavior on previously unseen inputs. Table 1 summarizes the sources of these inputs. All of the applications support both single- and multi-threaded execution. In our experiments we use whichever mode is appropriate.

4.1 swaptions

Description: This financial analysis application uses Monte Carlo simulation to solve a partial differential equation that prices a portfolio of swaptions. Both the accuracy and the execution time increase with the number of simulations — the accuracy approaches an asymptote, while the execution time increases linearly.

Knobs: We use a single command line parameter, `-sm`, as the dynamic knob. This integer parameter controls the number of Monte Carlo simulations for each swaption. The values range from 10,000 to 1,000,000 in increments of 10,000; 1,000,000 is the default value for the PARSEC native input.

Inputs: Each input contains a set of parameters for a given swaption. The native PARSEC input simply repeats the same parameters multiple times, causing the application to recalculate the same swaption price. We augment the evaluation input set with additional randomly generated parameters so that the application computes prices for a range of swaptions.

QoS Metric: Swaptions prints the computed prices for each swaption. The QoS metric computes the distortion of the swaption prices (see Equation 1), weighting the prices equally to directly measuring the application's ability to produce accurate swaption prices.

4.2 x264

Description: This media application encodes a raw (uncompressed) video according to the H.264 standard [53]. Like virtually all video encoders, it uses lossy encoding, with the visual quality of the encoding typically measured using continuous values such as peak signal-to-noise ration.

Knobs: We use three knobs: `--subme` (an integer parameter which determines the algorithms used for sub-pixel motion estimation), `--merange` (an integer which governs the maximum search range for motion estimation), and `--ref` (which specifies the number of reference frames searched during motion estimation). `--subme` ranges from 1 to 7, `--merange` ranges from 1 to 16, and `--ref` ranges from 1 to 5. In all cases higher numbers correspond to higher quality encoded video and longer encoding times. The PARSEC native defaults for these are 7, 16, and 5, respectively.

Inputs: The native PARSEC input contains a single high-definition (1080p) video. We use this video and additional 1080p inputs from xiph.org [5].

QoS Metric: The QoS metric is the distortion of the peak signal to noise ratio (PSNR, as measured by the H.264 reference de-

Benchmark	Training Inputs	Production Inputs	Source
swaptions	64 swaptions	512 swaptions	PARSEC & randomly generated swaptions
x264	4 HD videos of 200+ frames	12 HD videos of 200+ frames	PARSEC & xiph.org [5]
bodytrack	sequence of 100 frames	sequence of 261 frames	PARSEC & additional input from PARSEC authors
swish++	2000 books	2000 books	Project Gutenberg [2]

Table 1: Summary of Training and Production Inputs for Each Benchmark

coder [22]) and bitrate (as measured by the size of the encoded video file), with the PSNR and bitrate weighted equally. This QoS metric captures the two most important attributes of encoded video: image quality and compression.

4.3 bodytrack

Description: This computer vision application uses an annealed particle filter and videos from multiple cameras to track a human's movement through a scene [15]. bodytrack produces two outputs: a text file containing a series of vectors representing the positions of body components (head, torso, arms, and legs) over time and a series of images graphically depicting the information in the vectors overlaid on the video frames from the cameras. In envisioned usage contexts [15], a range of vectors is acceptable as long as the vectors are reasonably accurately overlaid over the actual corresponding body components.

Knobs: bodytrack uses positional parameters, two of which we convert to knobs: `argv[5]`, which controls the number of annealing layers, and `argv[4]`, which controls the number of particles. The number of layers ranges from 1 to 5 (the PARSEC native default); the number of particles ranges from 100 to 4000 (the PARSEC native default) in increments of 100.

Inputs: bodytrack requires data collected from four carefully calibrated cameras. We use a sequence of 100 frames (obtained from the maintainers of PARSEC) as the training input and the PARSEC native input (a sequence of 261 frames) as the production input.

QoS Metric: The QoS metric is the distortion of the vectors that represent the position of the body parts. The weight of each vector component is proportional to its magnitude. Vector components which represent larger body components (such as the torso) therefore have a larger influence on the QoS metric than vectors that represent smaller body components (such as forearms).

4.4 swish++

Description: This search engine is used to index and search files on web sites. Given a query, it searches its index for documents that match the query and returns the documents in rank order. We configure this benchmark to run as a server — all queries originate from a remote location and search results must be returned to the appropriate location.

Knobs: We use the command line parameter `--max-results` (or `-m`, which controls the maximum number of returned search results) as the single dynamic knob. We use the values 5, 10, 25, 50, 75, and 100 (the default value).

Inputs: We use public domain books from Project Gutenberg [2] as our search documents. We use the methodology described by Middleton and Baeza-Yates [38] to generate queries for this corpus. Specifically, we construct a dictionary of all words present in the documents, excluding stop words, and select words at random following a power law distribution. We divide the documents randomly into equally-sized training and production sets.

QoS Metric: We use F-measure [36] (a standard information retrieval metric) as our QoS metric. F-measure is the harmonic mean of the *precision* and *recall*. Given a query, precision is the number of returned documents that are relevant to the query divided by the total number of returned documents. Recall is the number of relevant returned documents divided by the total number of relevant documents (returned or not). We examine precision and recall at different cutoff values, using typical notation **P @N**.

4.5 Discussion

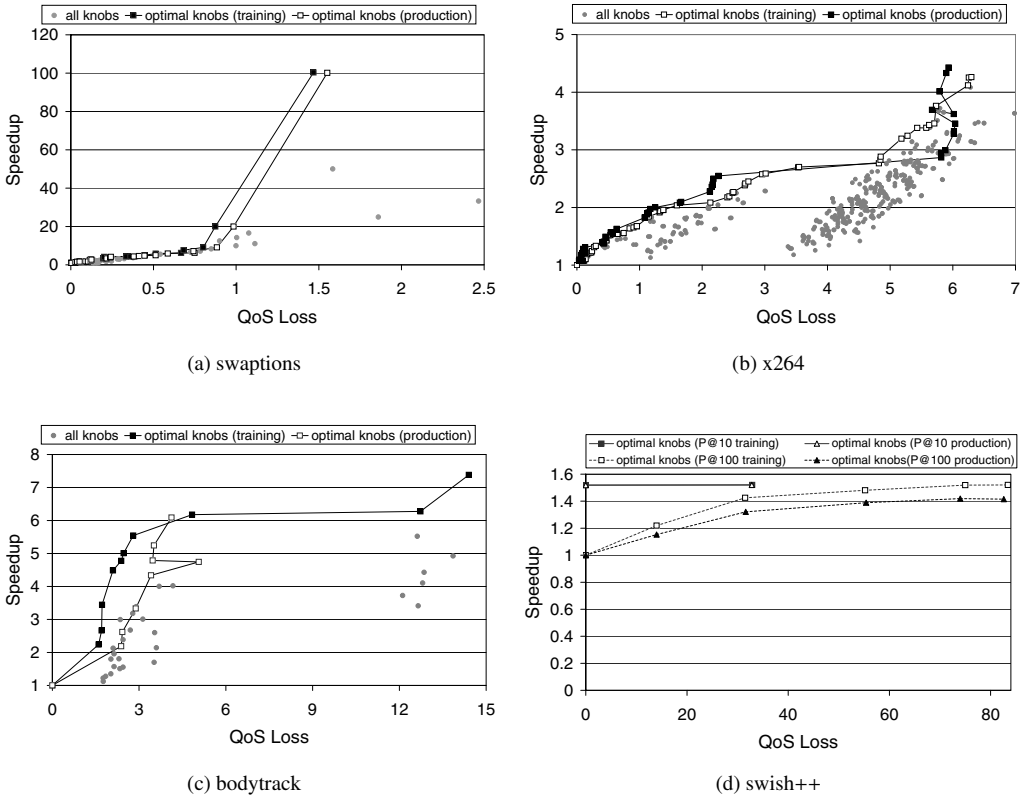
These applications are broadly representative of our target set of applications — they all have a performance versus quality of service trade-off and they all make that trade-off available via configuration parameters. Other examples of applications with appropriate trade-off spaces include most sensory applications (applications that process sensory data such as images, video, and audio), most machine learning applications, many financial analysis applications (especially applications designed for use in competitive high-frequency trading systems, where time is critically important), many scientific applications, and many Monte-Carlo simulations. Such applications (unlike more traditional applications such as compilers or databases) are typically inherently approximate computations that operate largely without a notion of hard logical correctness — for any given input, they instead have a range of acceptable outputs (with some outputs more accurate and therefore more desirable than others). This broad range of acceptable outputs, in combination with the fact that more accurate outputs are often more computationally expensive to compute, gives rise to the performance versus quality of service trade-offs that PowerDial enables the applications to dynamically navigate.

There are a variety of reasons such applications would be deployed in contexts that require responsive execution. Applications that process soft real-time data for human users (for example, video-conferencing systems) need to produce results responsively to deliver an acceptable user experience. Search and information retrieval applications must also present data responsively to human users (although with less stringent response requirements). Other scenarios involve automated interactions. Bodytrack and similar probabilistic analysis systems, for example, could be used in real-time surveillance and automated response systems. High-frequency trading systems are often better off trading on less accurate results that are available more quickly — because of competition with other automated trading systems, opportunities for lucrative trades disappear if the system does not produce timely results.

5. Experimental Evaluation

We next discuss the experimental platform and each of the experiments used to evaluate PowerDial. Our first set of experiments explores the performance versus QoS trade-off space for each of our benchmark applications. Next, we explore the (closely related) power versus QoS trade-off spaces. We then investigate how PowerDial enables applications to respond to the imposition of power caps using dynamic knobs. Finally, we investigate the use of PowerDial to reduce the number of machines required for servicing workloads with intermittent load spikes. This reduction can, in turn, reduce the cost of acquiring and operating the system.

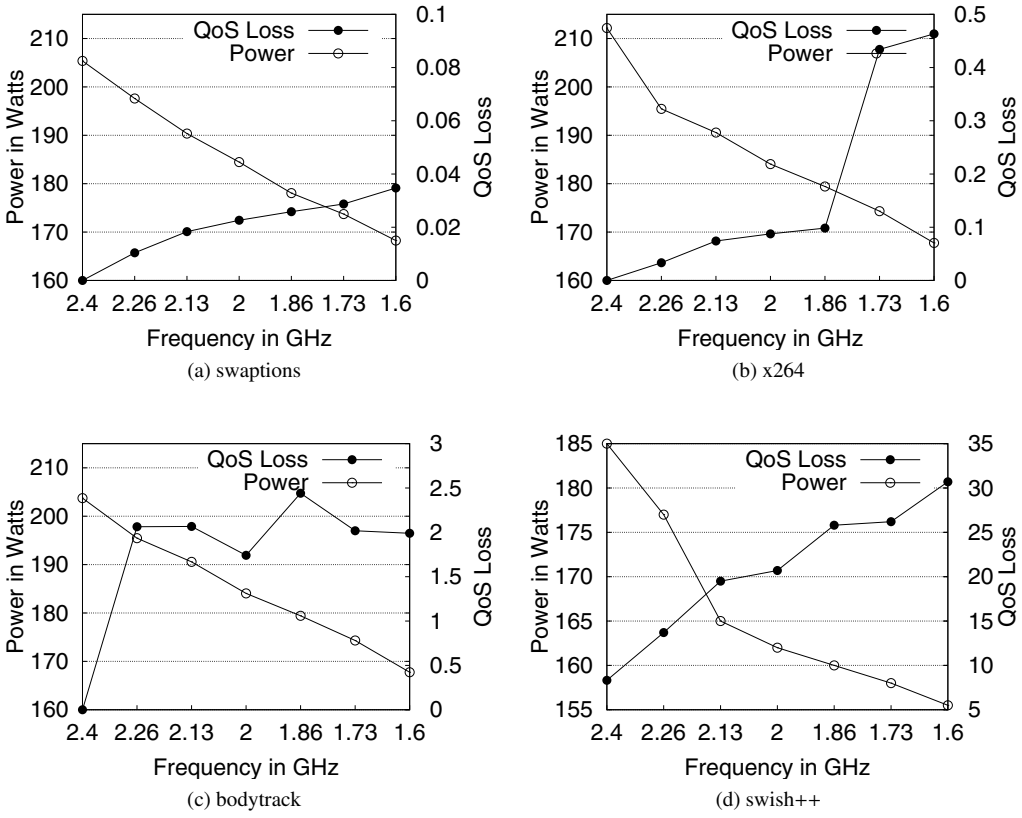
For each application, our experimental evaluation works with two data sets: a training data set used to characterize the application's performance versus QoS trade-off space, and a production data set used to evaluate how well the obtained characterization



(a) swaptions (b) x264

(c) bodytrack (d) swish++

Figure 5: QoS loss versus speedup for each benchmark.



(a) swaptions (b) x264

(c) bodytrack (d) swish++

Figure 6: Power versus QoS trade-offs for each benchmark.

generalizes to other inputs. We obtain a set of inputs for each application, then randomly divide this set of inputs into training and production sets.

5.1 Experimental Platform

We run all our experiments on a Dell PowerEdge R410 server with two quad-core Intel Xeon E5530 processors running Linux 2.6.26. The processors support seven power states with clock frequencies from 2.4 GHz to 1.6 GHz. The `cpufrequtils` package enables software control of the clock frequency (and thus the power state). We use a WattsUp device to sample and store the consumed power at 1 second intervals [4]. All benchmark applications run for significantly more than 1 second. The measured power ranges from 220 watts (at full load) to 80 watts (idle), with a typical idle power consumption of approximately 90 watts. The WattsUp device measures full system power and all results reported here are based on this measurement.

We measure the overhead of the PowerDial control system by comparing the performance of the benchmarks with and without the control system. The overhead of the PowerDial control system is insignificant and within the run-to-run variations in the execution times of the benchmarks executing without the control system.

5.2 Performance Versus QoS Trade-Offs

Dynamic knobs modulate power consumption by controlling the amount of computational work required to perform a given task. On a machine that delivers constant baseline performance (i.e., no clock frequency changes), changes in computational work correspond to changes in execution time.

Figures 5a–5d present the points that dynamic knobs make available in the speedup versus QoS trade-off space for each benchmark application. The points in the graphs plot the observed mean (across the training or production inputs as indicated) speedup as a function of the observed mean QoS loss for each dynamic knob setting. Gray dots plot results for the training inputs, with black squares (connected by a line) indicating Pareto-optimal dynamic knob settings. White squares (again connected by a line) plot the corresponding points for these Pareto-optimal dynamic knob settings for the production inputs. All speedups and QoS losses are calculated relative to the dynamic knob setting which delivers the highest QoS (and consequently the largest execution time). We observe the following facts:

- **Effective Trade-Offs:** Dynamic knobs provide access to operating points across a broad range of speedups (up to 100 for swaptions, 4.5 for x264, and 7 for bodytrack). Moreover, QoS losses are acceptably small for virtually all Pareto-optimal knob settings (up to only 1.5% for swaptions, 7% for x264, and, for speedups up to 6, 6% for bodytrack).
  For swish++, dynamic knobs enable a speedup of up to approximately a factor of 1.5. The QoS loss increases linearly with the dynamic knob setting. The effect of the dynamic knob is, however, very simple — it simply drops lower-priority search results. So, for example, at the fastest dynamic knob setting, swish++ returns the top five search results.
- **Close Correlation:** To compute how closely behavior on production inputs tracks behavior on training inputs, we take each metric (speedup and QoS loss), compute a linear least squares fit of training data to production data, and compute the correlation coefficient of each fit (see Table 2). The correlation coefficients are all close to 1, indicating that behavior on training inputs is an excellent predictor of behavior on production inputs.

Benchmark	Speedup	QoS Loss
x264	0.995	0.975
bodytrack	0.999	0.839
swaptions	1.000	0.999
swish++	0.996	0.999

Table 2: Correlation coefficient of observed values from training with measured values on production inputs.

5.3 Power Versus QoS Trade-offs

To characterize the power versus QoS trade-off space that dynamic knobs make available, we initially configure each application to run at its highest QoS point on a processor in its highest power state (2.4 GHz) and observe the performance (mean time between heartbeats). We then instruct the PowerDial control system to maintain the observed performance, use `cpufrequtils` to drop the clock frequency to each of the six lower-power states, run each application on all of the production inputs, and measure the resulting performance, QoS loss, and mean power consumption (the mean of the power samples over the execution of the application in the corresponding power state). We verify that, for all power states, PowerDial delivers performance within 5% of the target.

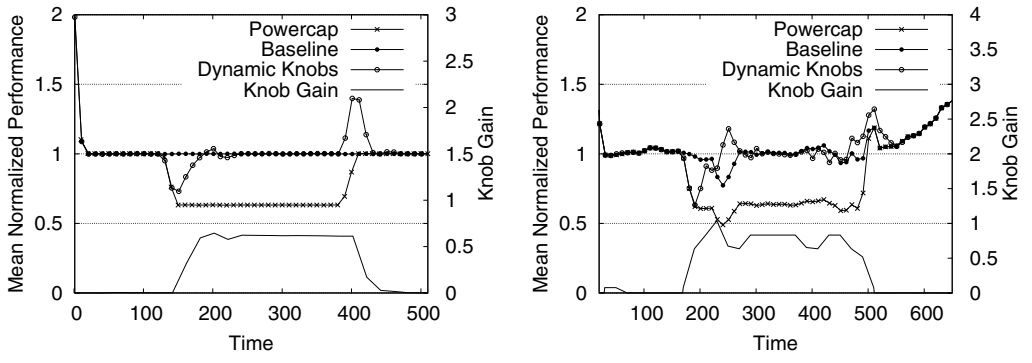
Figures 6a–6d plot the resulting QoS loss (right y axis, in percentages) and mean power (left y axis) as a function of the processor power state. For x264, the combination of dynamic knobs and frequency scaling can reduce system power by as much as 21% for less than 0.5% QoS loss. For bodytrack, we observe a 17% reduction in system power for less than 2.3% QoS loss. For swaptions, we observe an 18% reduction in system power for less than .05% QoS loss. Finally, for swish++ we observe power reductions of up to 16% for under 32% QoS loss. For swish++ the dynamic knob simply truncates the list of returned results — the top results are the same, but swish++ returns fewer total results.

The graphs show that x264, bodytrack, and swaptions all have suboptimal dynamic knob settings that are dominated by other, Pareto-optimal dynamic knob settings. The exploration of the trade-off space during training is therefore required to find good points in the trade-off space. The graphs also show that because the Pareto-optimal settings are reasonably consistent across the training and production inputs, the training exploration results appropriately generalize to the production inputs.

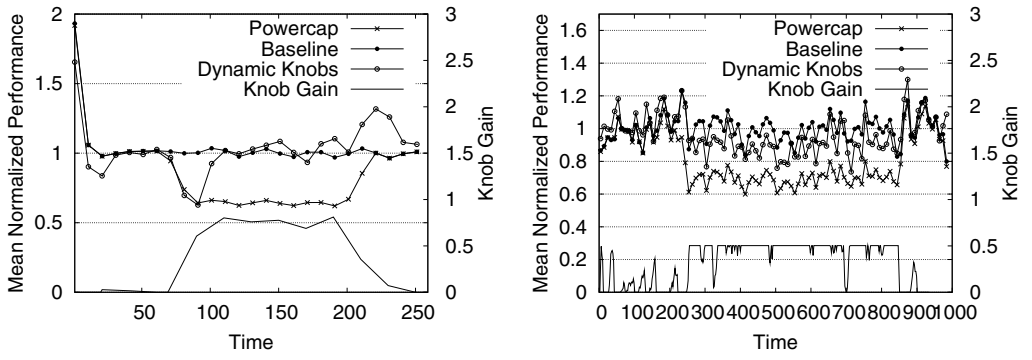
5.4 Elastic Response to Power Capping

The PowerDial system makes it possible to dynamically adapt application behavior to preserve performance (measured in heartbeats) in the face of *any* event that degrades the computational capacity of the underlying platform. We next investigate a specific scenario — the external imposition of a temporary power cap via a forced reduction in clock frequency. We first start the application running on a system with uncapped power in its highest power state (2.4 GHz). We instruct the PowerDial control system to maintain the observed performance (time between heartbeats). Approximately one quarter of the way through the computation we impose a power cap that drops the machine into its lowest power state (1.6 GHz). Approximately three quarters of the way through the computation we lift the power cap and place the system back into its highest power state (2.4 GHz).

Figures 7a–7d present the dynamic behavior of the benchmarks as they respond to the power cap and corresponding processor frequency changes. Each graph plots the observed performance (computed as the sliding mean of the last twenty times between heartbeats times normalized to the target heart rate of the application) of the application (left y axis) as a function of time. We present the

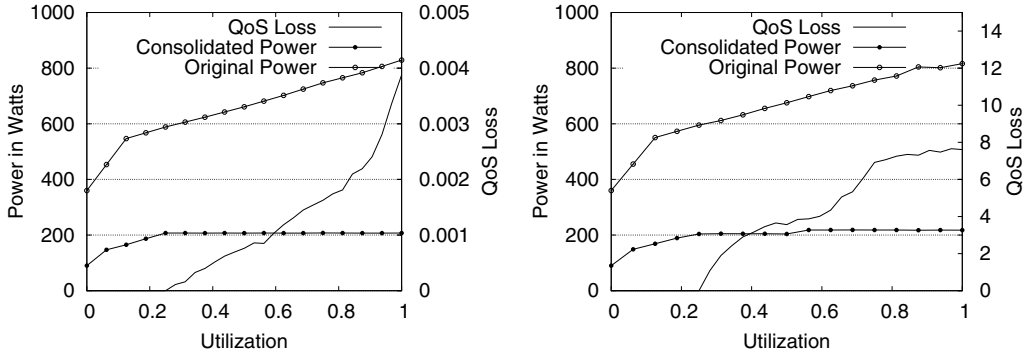


(a) swaptions (b) x264

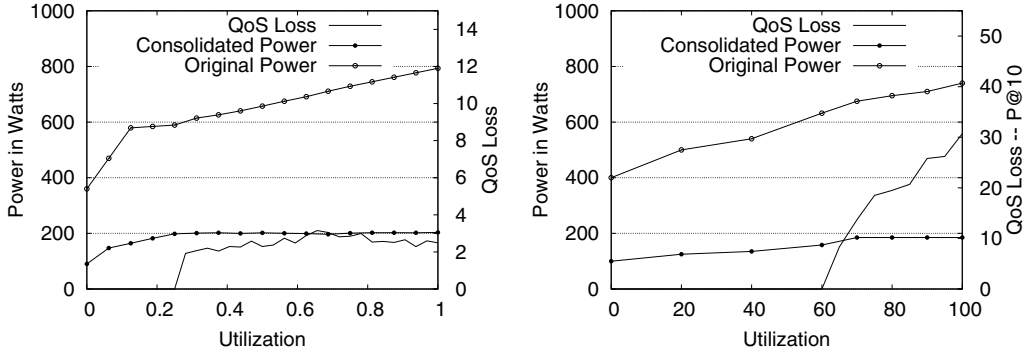


(c) bodytrack (d) swish++

Figure 7: Behavior of benchmarks with dynamic knobs in response to power cap.



(a) swaptions (b) x264



(c) bodytrack (d) swish++

Figure 8: Using dynamic knobs for system consolidation.

performance of three versions of the application: a version without dynamic knobs (marked with an ×), a baseline version running with no power cap in place (black points), and a version that uses dynamic knobs to preserve the performance despite the power cap (circles). We also present the knob "gain" or the instantaneous speedup achieved by the dynamic knob runtime (right y axis).

All applications exhibit the same general pattern. At the imposition of the power cap, PowerDial adjusts dynamic knobs, the gain increases (Knob Gain line), and the performance of the application first spikes down (circles), then returns back up to the baseline performance. When the power cap is lifted, the dynamic knobs adjust again, the gain decreases, and the application performance returns to the baseline after a brief upward spike. For most of the first and last quarters of the execution, the application executes with essentially no QoS loss. For the middle half of the execution, the application converges to the low power operating point plotted in Figures 6a–6d as a function of the 1.6 GHz processor frequency. Without dynamic knobs (marked with ×), application performance drops well below the baseline as soon as the power cap is imposed, then rises back up to the baseline only after the power cap is lifted.

Within this general pattern the applications exhibit varying degrees of noise in their response. Swaptions exhibits very predictable performance over time with little noise. swish++, on the other extreme, has relatively unpredictable performance over time with significant noise. x264 and bodytrack fall somewhere in between. Despite the differences in application characteristics, our dynamic adaptation mechanism makes it possible for the applications to largely satisfy their performance goals in the face of dynamically fluctuating power requirements.

5.5 Peak Load Provisioning

We next evaluate the use of dynamic knobs to reduce the number of machines required to service time-varying workloads with intermittent load spikes, thereby reducing the number of machines, power, and indirect costs (such as cooling costs) required to maintain responsive execution in the face of such spikes:

- **Target Performance:** We set the target performance to the performance achieved by running one instance of the application on an otherwise unloaded machine.
- **Baseline System:** We start by provisioning a system to deliver target performance for a specific peak load of the applications running the baseline (default command line) configuration. For the three PARSEC benchmarks we provision for a peak load of 32 (four eight-core machines) concurrent instances of the application. For swish++ we provision for a peak load of three concurrent instances, each with eight threads. This system load balances all jobs proportionally across available machines. Machines without jobs are idle but not powered off.
- **Consolidated System:** We impose a bound of either 5% (for the PARSEC benchmarks) or 30% (for swish++) QoS loss. We then use Equation 21 to provision the minimum number of machines required for PowerDial to provide baseline performance at peak load subject to the QoS loss bound. For the PARSEC benchmarks we provision a single machine. For swish++ we provision two machines.
- **Power Consumption Experiments:** We then vary the load from 0% utilization of the original baseline system (no load at all) to 100% utilization (the peak load). For each load, we measure the power consumption of the baseline system (which delivers baseline QoS at all utilizations) and the power consumption and QoS loss of the consolidated system (which uses PowerDial to deliver target performance. At low utilizations the consolidated system will configure the applications to deliver maximum QoS. As the utilization increases, PowerDial will progressively manipulate the dynamic knobs to maintain the target performance at the cost of some QoS loss.

Figures 8a–8d presents the results of these experiments. Each graph plots the mean power consumption of the original (circles) and consolidated (black dot) systems (left y axis) and the mean QoS loss (solid line, right y axis) as a function of system utilization (measured with respect to the original, fully provisioned system). These graphs show that using dynamic knobs to consolidate machines can provide considerable power savings across a range of system utilization. For each of the PARSEC benchmarks, at a system utilization of 25% consolidation can provide an average power savings of approximately 400 Watts, a reduction of 66%. For swish++ at 20% utilization, we see a power savings of approximately 125 Watts, a reduction of 25%. These power savings come from the elimination of machines that would be idle in the baseline system at these utilization levels.

Of course, it is not surprising that reducing the number of machines reduces power consumption. A key benefit of the dynamic knob elastic response mechanism is that even with the reduction in computational capacity, it enables the system to maintain the same performance at peak load while consuming significantly less power. For the PARSEC benchmarks at a system utilization of 100%, the consolidated systems consume approximately 75% less power than the original system while providing the same performance. For swish++ at 100% utilization, the consolidated system consumes 25% less power.

The consolidated systems save power by automatically reducing QoS to maintain performance. For swaptions, the maximum QoS loss required to meet peak load is 0.004%, for x264 it is 7.6%, and for bodytrack it is 2.5%. For swish++ with P@10, the QoS loss is 8% at a system utilization of 65%, rising to 30% at a system utilization of 100%. We note, however, that the majority of the QoS loss for swish++ is due to a reduction in recall — top results are generally preserved in order but fewer total results are returned. Precision is not affected by the change in dynamic knob unless the P@N is less than the current knob setting. As the lowest knob setting used by PowerDial is five, precision is always perfect for the top 5 results.

For common usage patterns characterized by predominantly low utilization punctuated by occasional high-utilization spikes [9], these results show that dynamic knobs can substantially reduce overall system cost, deliver the highest (or close to highest) QoS in predominant operating conditions, and preserve performance and acceptable QoS even when the system experiences intermittent load spikes. Note that system designers can use the equations in Section 3 to choose a consolidation appropriate for their envisioned usage pattern that minimizes costs yet still delivers acceptable QoS even under the maximum anticipated load spike.

6. Related Work

Adaptive, or self-aware, computing systems have the flexibility to meet multiple goals in changing computing environments. A number of adaptive techniques have been developed for both software [45] and hardware [6]. Adaptive hardware techniques are complementary to the Dynamic Knobs approach for developing adaptive applications. If such hardware chooses to save power by reducing computational capacity, Dynamic Knobs can enable the software to respond and maintain performance. This section focuses on related software techniques.

Trading accuracy of computation for other benefits is a well-known technique. It has been shown that one can trade off accuracy for performance [16, 32, 40, 43, 44], energy consumption [12, 13, 16, 20, 34, 43, 47] and fault tolerance [13, 43, 47]. The Dynamic Knobs system presented in this paper, along with loop

perforation [26, 39] and task skipping [43, 44], is unique in that it enables applications to adaptively trade accuracy for performance and does so without requiring a developer to change the application source code.

Autotuners explore a range of equally accurate implementation alternatives to find the alternative or combination of alternatives that deliver the best performance on the current computational platform [17, 52, 54]. Researchers have also developed APIs that an application can use to expose variables for external control (by, for example, the operating system) [29, 42, 49]. This paper presents a system (PowerDial) that transforms static configuration parameters into dynamic knobs and contains a control system that uses the dynamic knobs to maintain performance in the face of load fluctuations, power fluctuations, or any other event that may impair the ability of the application to successfully service its load with the given computational resources. It also presents experimental results that demonstrate the effectiveness of its approach in enabling server consolidation and effective execution through power reductions (imposed, for example, by power caps).

Researchers have developed several systems that allow programmers to provide multiple implementations for a given piece of functionality, with different implementations occupying different points in the performance versus accuracy trade-off space. Such systems include the *tunability interface* [14], Petabricks [7], Green [8], and Eon [46]. Chang and Karamcheti's tunability interface allows application developers to provide multiple configurations of an application (specified by the programmer through compiler directives). A tunable application is then modeled in a *virtual execution environment*, to determine which configuration is best suited for different system states. Petabricks is a parallel language and compiler that developers can use to provide alternate implementations of a given piece of functionality. Green also provides constructs that developers can use to specify alternate implementations. The alternatives typically exhibit different performance and QoS characteristics. PetaBricks and Green both contain algorithms that explore the trade-off space to find points with desirable performance and QoS characteristics. Eon [46] is a coordination language for power-aware computing that enables developers to adapt their algorithms to different energy contexts. In a similar vein, energy-aware adaptation for mobile applications [16], adapts to changing system demands by dynamically adjusting application input quality. For example, to save energy the system may switch to a lower quality video input to reduce the computation of the video decoder.

Each of these systems requires the developer to intervene directly in the source code to provide or specify multiple implementations of the same functionality. They can therefore increase the size and development cost of the application and require the presence of a developer who understands the internal structure of the implementation and can appropriately modify the implementation. These systems are therefore of little or no use when such a developer is unavailable, either because the original developers are no longer with the organization or are dedicated to other projects; the organization that originally developed the software no longer exists, is no longer developing or maintaining the application, or is simply unwilling to incorporate the functionality into their code base; or when the cost of performing the modifications is prohibitively expensive.

In contrast, PowerDial works directly on unmodified and unannotated applications. It automatically transforms existing configuration parameters into dynamic knobs and automatically inserts the appropriate Application Heartbeats API calls into the application. It can therefore enable third-party users to automatically augment their applications with desirable dynamic adaptation properties without the need to involve knowledgeable developers or the organization that originally developed the application.

None of Petabricks, Green, or Eon provides a control mechanism which can react to changes that affect performance. Petabricks does not have a dynamic control component. Green uses heuristic control to manage quality of service but does not control or even monitor performance. Similarly, Eon uses a heuristic control system to manage the energy consumption of the system, but does not directly control performance. Both control systems are completely heuristic, with no guaranteed convergence or predictability properties whatsoever. The Chang/Karamcheti approach does directly control performance using a heuristic decision mechanism. This controller monitors system state and attempts to select a configuration appropriate for the current state, but does not use direct feedback from the application.

In contrast, PowerDial uses a decision mechanism grounded in control science with provably good convergence and predictability properties [35]. By relying on a modeling phase to discover Pareto-optimal knob settings, the PowerDial control system is able to solve constrained optimization problems to dynamically maintain performance while minimizing quality loss. In addition, the PowerDial control system uses Heartbeats as its feedback mechanism. By using direct feedback from the application the control system is able to operate maintain performance without having to infer application performance from low measurements of system state.

Researchers have also explored the use of loop perforation (which automatically transforms loops to skip loop iterations) to augment applications with the ability to operate at different points in an induced performance versus quality of service trade-off space [26, 39]. The results show that loop perforation can help developers find computations that are suitable for further optimization [39] and enables applications to adapt to fluctuations in the delivered computational resources [26]. Task skipping [43, 44] has also been shown to automatically augment applications with the ability to trade off quality of service in return for increased performance. This paper presents a system that uses dynamic knobs instead of loop perforation, has a control system with guaranteed performance and predictability properties, and more fully demonstrates how to use dynamic knobs to solve power management issues.

Hellerstein et al [23] and Karamanolis et al [28] have both identified standard control theoretic techniques as a general solution for managing dynamic behavior in computing systems. Other authors have shown how control techniques can be generalized allowing software developers to incorporate them without having to develop expertise in control science [21, 25, 33, 35, 55]. The PowerDial control system furthers this idea, showing how standard control techniques can be automatically embedded into an application to dynamically manage performance. The control system presented in this paper uses Application Heartbeats as a feedback mechanism (or sensor) combined with a novel actuation strategy which converts static configuration options into dynamically tunable parameters. Control theory provides predictable behavior, making it a good match for applications with performance constraints.

7. Conclusion

The PowerDial system augments applications with dynamic knobs that the PowerDial control system can use to adapt the behavior of the application to execute successfully in the face of load spikes, power fluctuations, or (in general) any event that changes the balance between the computational demand and the resources available to meet that demand. Our results demonstrate that PowerDial can enable applications to maintain responsive execution in the face of power caps and load spikes (thereby reducing or even eliminating the over-provisioning otherwise required to service these spikes). We see PowerDial as an early example of an emerging class of management systems that will enable applications to oper-

ate successfully in complex modern computing environments characterized by fluctuations in power, load, and other key operating characteristics.

Acknowledgements

Henry Hoffmann and Anant Agarwal are grateful for support from DARPA, the NSF, and Quanta Computer. Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, and Martin Rinard are supported in part by the National Science Foundation under Grant Nos. 0937060 to the Computing Research Association for the CIFellows Project, Nos. CNS-0509415, CCF-0811397 and IIS-0835652, DARPA under Grant No. FA8750-06-2-0189 and Massachusetts Institute of Technology. We note our earlier technical reports on performance versus QoS trade-offs [26, 27].

References

[1] Intel Xeon Processor. http://www.intel.com/technology/Xeon.

[2] Project Gutenberg. http://www.gutenberg.org/.

[3] Intel Atom Processor. http://www.intel.com/technology/atom.

[4] Wattsup .net meter. http://www.wattsupmeters.com/.

[5] Xiph.org. http://xiph.org.

[6] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36:49–58, December 2003.

[7] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. Petabricks: A language and compiler for algorithmic choice. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Dublin, Ireland, June 2009.

[8] W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2010.

[9] L. Barroso and U. Holzle. The case for energy-proportional computing. *COMPUTER-IEEE COMPUTER SOCIETY-*, 40(12):33, 2007.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT-2008: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[11] M. Carbin and M. Rinard. Automatically Identifying Critical Input Regions and Code in Applications. In *Proceedings of the International Symposium on Software Testing and Analysis*, 2010.

[12] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee. Ultra-efficient (embedded) soc architectures based on probabilistic cmos (pcmos) technology. In *Proceedings of the conference on Design, automation and test in Europe*, DATE, pages 1110–1115, 2006.

[13] L. N. Chakrapani, K. K. Muntimadugu, A. Lingamneni, J. George, and K. V. Palem. Highly energy and performance efficient embedded computing through approximately correct arithmetic: a mathematical foundation and preliminary experimental validation. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, CASES, pages 187–196, 2008.

[14] F. Chang and V. Karamcheti. Automatic configuration and run-time adaptation of distributed applications. In *Proceedings of the International ACM Symposium on High Performance Parallel and Distributed Computing*, HPDC, pages 11–20, 2000.

[15] J. Deutscher and I. Reid. Articulated body motion capture by stochastic search. *International Journal of Computer Vision*, 61(2):185–205, 2005.

[16] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, page 63. ACM, 1999.

[17] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.

[18] A. Gandhi, M. Harchol-Balter, R. Das, C. Lefurgy, and J. Kephart. Power capping via forced idleness. In *Workshop on Energy-Efficient Design*, June 2009.

[19] V. Ganesh, T. Leek, and M. Rinard. Taint-based directed whitebox fuzzing. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 474–484. IEEE Computer Society, 2009.

[20] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, CASES, pages 158–168, 2006.

[21] A. Goel, D. Steere, C. Pu, and J. Walpole. Swift: A feedback control and dynamic reconfiguration toolkit. In *2nd USENIX Windows NT Symposium*, 1998.

[22] H.264 reference implementation. http://iphome.hhi.de/suehring/tml/download/.

[23] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[24] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments. In *7th International Conference on Autonomic Computing*, ICAC, 2010.

[25] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. SEEC: A Framework for Self-aware Computing. Technical Report MIT-CSAIL-TR-2010-049, CSAIL, MIT, October 2010.

[26] H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures . Technical Report MIT-CSAIL-TR-2009-042, CSAIL, MIT, September 2009.

[27] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Power-Aware Computing with Dynamic Knobs. Technical Report TR-2010-027, CSAIL, MIT, May 2010.

[28] C. Karamanolis, M. Karlsson, and X. Zhu. Designing controllable computer systems. In *Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 9–15, Berkeley, CA, USA, 2005. USENIX Association.

[29] P. J. Keleher, J. K. Hollingsworth, and D. Perkovic. Exposing application alternatives. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, ICDCS, page 384, Washington, DC, USA, 1999. IEEE Computer Society.

[30] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization* , CGO, Palo Alto, California, March 2004.

[31] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[32] J. Letchner, C. Re, M. Balazinska, and M. Philipose. Approximation trade-offs in markovian stream processing: An empirical study. In *2010 IEEE 26th International Conference on Data Engineering*, ICDE, pages 936 –939, 2010.

[33] B. Li and K. Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *Selected Areas in Communications, IEEE Journal on*, 17(9):1632 –1650, September 1999.

[34] S. Liu, K. P. amd Thomas Moscibroda, and B. G. Zorn. Flicker: Saving Refresh-Power in Mobile Devices through Critical Data Partitioning. Technical Report MSR-TR-2009-138, Microsoft Research, Oct. 2009.

[35] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Controlling software applications via resource allocation within the Heartbeats frame work. In *49th IEEE Conference on Decision and Control*, pages 3736 –3741, December 2010.

[36] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance measures for information extraction. In *Broadcast News Workshop'99 Proceedings*, page 249. Morgan Kaufmann Pub, 1999.

[37] D. Meisner, B. Gold, and T. Wenisch. PowerNap: eliminating server idle power. *ACM SIGPLAN Notices*, 44(3):205–216, 2009.

[38] C. Middleton and R. Baeza-Yates. A comparison of open source search engines. Technical report, Universitat Pompeu Fabra, Department of Technologies, October 2007.

[39] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, ICSE, pages 25–34. ACM, 2010.

[40] R. Narayanan, B. Ozisikyilmaz, G. Memik, A. Choudhary, and J. Zambreno. Quantization error and accuracy-performance tradeoffs for embedded data mining workloads. In *Proceedings of the 7th international conference on Computational Science*, ICCS, pages 734–741, Berlin, Heidelberg, 2007. Springer-Verlag.

[41] S. Pelley, D. Meisner, P. Zandevakili, T. Wenisch, and J. Underwood. Power routing: dynamic power provisioning in the data center. *ACM SIGPLAN Notices*, 45(3):231–242, 2010.

[42] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: adaptive control of distributed applications. In *High Performance Distributed Computing*, July 1998.

[43] M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 324–334. ACM New York, NY, USA, 2006.

[44] M. C. Rinard. Using early phase termination to eliminate load imbalances at barrier synchronization points. In *Proceedings of the 22nd annual ACM conference on Object-oriented programming systems and applications*, OOPSLA, pages 369–386, New York, NY, USA, 2007. ACM.

[45] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, 2009.

[46] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: a language and runtime system for perpetual systems. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys, New York, NY, USA, 2007. ACM.

[47] P. Stanley-Marbell, D. Dolech, A. Eindhoven, and D. Marculescu. Deviation-Tolerant Computation in Concurrent Failure-Prone Hardware. Technical Report ESR-2008-01, Eindhoven University of Technology, January 2008.

[48] SWISH++. http://swishplusplus.sourceforge.net/.

[49] C. Tapus, I. Chung, and J. Hollingsworth. Active harmony: Towards automated performance tuning. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[50] U.S. Environmental Protection Agency. EPA report to congress on server and data center energy efficiency, 2007.

[51] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. *Mobile Computing*, pages 449–471, 1996.

[52] R. Whaley and J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–27. IEEE Computer Society, 1998.

[53] x264. http://www.videolan.org/x264.html.

[54] J. Xiong, J. Johnson, R. W. Johnson, and D. Padua. SPL: A language and compiler for DSP algorithms. In *Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation*, PLDI, pages 298–308, 2001.

[55] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of the 22nd International conference on Distributed Computing Systems*. IEEE computer society, 2002.