Autonomic Power Management Schemes for Internet Servers and Data Centers

Lykomidis Mastroleon, Nicholas Bambos, Christos Kozyrakis and Dimitris Economou
Department of Electrical Engineering
Stanford University
350 Serra Mall, Stanford, CA 94305–9505
Phone: 650-725-5525, Fax: 650-723-4107
Email: {lmastro,bambos,christos,dimeco}@stanford.edu

***Abstract*— We investigate autonomic power control policies for internet servers and data centers. In particular, by monitoring the system load and thermal status, we decide how to vary the utilized processing resources to achieve acceptable delay and power performance. We formulate the problem using a Dynamic Programming approach that captures the power-performance tradeoff. We study the structural properties of the optimal solution and develop low-complexity justified heuristics, which achieve significant performance gains over standard benchmarks. The performance gains are higher when the load exhibits stronger temporal variations. We also demonstrate that the heuristics are very efficient, in the sense that they perform very close to the optimal solution obtained via dynamic programming.**

I. Introduction

Data center infrastructures are a major power consumer with rapidly increasing demand [3], [7]. Scientists at Lawrence Berkeley National Laboratory estimated that Web sites and other servers consumed over 7 terawatt-hours (TWh) of electricity in the US in 1999, more than the internet switching infrastructure itself [4]. Modern data centers contain tightly placed hardware that requires a large amount of power to operate; moreover, a significant amount of power is also required for cooling such installations [2], [6]. The resulting power cost sums up to a significant fraction of the total investment (over its lifetime). This becomes larger every year as hardware speeds and power consumption increase.

Data centers experience large periods of low utilization, during which power management can drastically reduce power consumption with minimal performance impact. Indeed, the load of large data centers exhibits strong temporal variations [1], [5] over multiple time scales, which provide opportunities for power control of high impact. Ideally, an optimal control scheme would power off hardware resources in a judicious manner without compromising service performance. This would also reduce the thermal stress of the systems and decrease the cooling power.

Internet servers are key building blocks of data centers. Such servers usually contain multiple CPUs and sophisticated cooling systems. Server loads can exhibit strong temporal variations. In fact, at an appropriately high level of modeling abstraction, an internet server can be viewed as a 'caricature' of a data center, in the sense that it comprises a rich environment of power-managed resources. Of course, the scale is orders of magnitude larger in a data center. However, for the purposes of our analysis below, the internet server paradigm is enough and can appropriately scale to larger environments of controlled resources.

This paper develops efficient autonomic power control schemes that can be applied both in single internet servers or data centers. In both cases, the schemes result in (1) efficient utilization of available resources and (2) adaptive scaling of power demand.

The rest of the paper is organized as follows. In section II, we present a simple model that captures the key performance tradeoff(s) and provides a framework for computing efficient dynamic power control policies. This formulation is very flexible in terms of the performance costs it can incorporate. We leverage the Dynamic Programming (DP) methodology to obtain optimal power control policies. In section III, we analyze the structural properties of the optimal policies to design justified, near-optimal, practical (low-complexity) heuristics. In section IV, we evaluate the proposed heuristics and show that they achieve substantial performance gains over conventional benchmarks. Finally, in section V we discuss various extensions and future research.

II. Basic Model and Problem Formulation

In this section we introduce the basic model, reflecting our problem formulation and capturing the performance tradeoffs and control issues. We embed the problem within a Markov decision process framework and use Dynamic Programming to compute the optimal control.

We consider a server processing job requests. The server is equipped with M CPUs and a finite buffer for storing jobs. Job requests received by the server are queued up in the buffer, if it is not full, or dropped otherwise. The job requests stored in the buffer retain their positions until they are completed. Time is slotted and indexed by $t = 0, 1, 2, 3, ...$ In each time slot, depending on the number of pending requests in the buffer, the server allocates a number of CPUs to it. The number of CPUs allocated to the buffer is limited by the number of CPUs allocated for other tasks that the server has to complete (*e.g.*operating systems tasks) and vice versa. Increasing the number of CPUs assigned to the job buffer decreases the overall processing time for the jobs in it (but also decreases

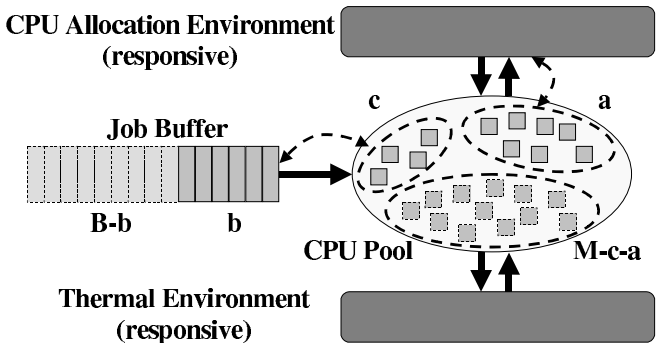


Fig. 1. The Model.

the number of CPUs available for other tasks) and increases the service cost and power consumption associated with it.

We focus on the simple model depicted in Fig. 1, taking the local perspective of an arbitrarily chosen buffer. The model is comprised of a single job buffer of size $\mathcal{B}$, a CPU pool, a CPU allocation environment and a thermal environment. We assume that the buffer initially (at time 0) contains b_0 jobs of a given type (no job arrivals occur in this simple model). At the beginning of each time slot, a decision regarding the number of CPUs to be used during the slot, is taken. The decision is based on the state of the CPU allocation environment and the state of the thermal environment of the model, as explained later.

The state of the CPU allocation environment a, is the number of CPUs that are available for use by the buffer at the beginning of each slot. It takes values in the finite set $\mathcal{A}$ of all attainable CPU environment states. The buffer may actually not choose to use all the CPUs available to it in the slot. When a decision is made (regarding how many CPUs the buffer will use in the current slot), the difference between the number c of CPUs that were used in the previous slot and the number u of CPUs to be used in this slot will affect a. Specifically, the state after the decision will be $a^* = a + c - u$ (available CPUs + CPUs that were in use before - CPUs to be used next). It is assumed that a^* remains invariant within each time slot once the decision is made. However, the CPU allocation environment state in the following time slot will switch according to a time-homogeneous Markov chain. More specifically, the CPU allocation environment changes from $a^* \in \mathcal{A}$ in a time slot (after decisions) to $a' \in \mathcal{A}$ in the next time slot (before decisions) with probability

$$p_{a^*a'|u}$$

which depends on the control u. Recall that u is the number of CPUs the buffer has decided to engage and use in the current time slot to serve its jobs. The Markov chain is irreducible and aperiodic. This random CPU environment captures the interactions of the buffer under consideration with the multitude of other tasks that engage and release CPUs all the time.

When the decision is made to use u CPUs in a slot, the head-of-line job in the buffer completes service and leaves at the end of the slot with probability $s(u)$, which is increasing in u. Thus, the buffered job population drops by 1 with probability $s(u)$ and remains constant with probability $1 - s(u)$. Service completion events are statistically independent in consecutive slots and also independent from CPU availability switching events (conditioned on u).

There is also a thermal environment state t which fluctuates according to a time-homogeneous Markov chain, taking values in a finite set $\mathcal{T}$ of all attainable states. It is assumed that t remains invariant within each time slot. Given the number of CPUs available in the beginning of a time slot a and the decision (number of CPUs to be used) u, the thermal environment changes from $t \in \mathcal{T}$ in the current time slot to $t' \in \mathcal{T}$ in the next one with probability

$$q_{tt'|(c,a,u)}$$

which depends on the the vector (c, a, u). The Markov chain is assumed to be irreducible and aperiodic, and statistically independent of the CPU availability environment (conditioned on (c, a, u)) and service completion events. Note that (c, a, u) alone does not fully capture the decisions and actions of other tasks in the system. However, the thermal environment t could reflect the thermal influence of the multitude of other tasks that engage and release CPUs all the time.

A. The Cost Structure

Let us now consider the costs the system incurs in each time slot. Assuming that at the beginning of a slot there are b jobs remaining in the buffer, the state of the thermal environment is t and it is decided to use u CPUs (instead of c that were used in the previous slot), we have the following 3 types of costs:

1) Backlog cost $C_b(b)$, increasing in the number of jobs b in the buffer.
2) Power cost (and thermal stress) $C_{ut}(u, t)$, increasing in the number u of CPUs used and the state (temperature) t of the thermal environment.
3) Reconfiguration cost $C_{uc}(u, c)$, capturing the overhead of changing the number of CPUs from c to u. We expect it to depend on the difference between u and c.

Other than the previously stated general structural properties of the various cost functions, we make no further assumptions on their particular forms.

B. Optimal Control

The objective is to serve all the jobs initially stored in the buffer (b_0 in number), while minimizing the overall cost incurred in the process. The system state to be tracked at each time slot is (b, t, c, a), that is the current job backlog b in the buffer, the current state of the thermal environment t, the number of CPUs that were used in the previous slot c, and the current state of the CPU allocation environment a. There is only one decision to be made at the beginning of each slot and that is the number u of CPUs to be used.

Given this formulation, the system becomes a controlled Markov decision process and, hence, we can develop a DP recursion to compute the optimal control. Let $J(b,t,c,a)$ be the cost-to-go that is, the total average cost incurred under optimal control to empty the buffer, given that the system starts from initial state (b,t,c,a). Then we can write the DP equation:

$$\begin{aligned} J(b,t,c,a) = \min_u \{ & C_b(b) + C_{ut}(u,t) + C_{uc}(u,c) + \\ & + s(u) \sum_{t',a'} q_{tt'|(c,a,u)} p_{a^* a'|u} J(b-1,t',u,a') + \\ & + [1-s(u)] \sum_{t',a'} q_{tt'|(c,a,u)} p_{a^* a'|u} J(b,t',u,a') \} \end{aligned} \quad (1)$$

where:

1) $a^* = a + c - u$
2) $b \in \{0,1,...,B\}$
3) $t \in \mathcal{T}$
4) $c \in \{0,1,...,M\}$
5) $a \in \mathcal{A} = \{0,1,...,M-c\}$
6) u is optimized over the range $0 \leq u \leq c+a$
7) There is no more cost once we finish all the jobs:

$$J(0,t,c,a) = 0, \qquad \forall c,t,a \quad (2)$$

Equation (1) can be interpreted as follows. Starting at state (b,t,c,a) and exercising optimal control u, the minimum cost-to-go is comprised of the current cost $(C_b(b) + C_{ut}(u,t) + C_{uc}(u,c))$ plus the expected future cost. The future cost is:

1) $J(b,t',u,a')$ with probability

$$s(u) q_{tt'|(c,a,u)} p_{a^* a'|u}$$

if the head-of-line job does not complete service.

2) $J(b-1,t',u,a')$ with probability

$$(1-s(u)) q_{tt'(c,a,u)} p_{a^* a'|u}$$

if the job completes service in this slot and leaves.

We must then sum the above terms over all possible thermal environment states $t' \in \mathcal{T}$ and all possible allocation environment states $a' \in \mathcal{A}$. This explains (1).

C. Computing the Optimal Control

The solution of the DP (1), (2) results in the optimal decision u given that the system is in state (b,t,c,a). Notice that this decision does not depend on the specific time slot but only on the specific state of the system. Therefore, a stationary optimal policy can be constructed by using the same state-based decision rule for each time slot until the buffer empties. We implicitly assume that when the buffer empties (state $(0,t,c,a)$), the system reaches a zero cost exit state.

The solution of (1), (2) can easily be obtained using the *policy iteration* method [8].

D. The Power vs. Delay Tradeoff

It is natural to expect that the average queueing delay (including processing time) for a job in the buffer will decrease as the average utilized power is increased. The issue is how to quantify this tradeoff and this is what the above model achieves.

Notice that the queuing delay stress is reflected in the buffering cost $C_b(b)$. As this increases, the system will try to minimize the overall cost by emptying the buffer faster. The sum of the power cost $C_{ut}(u,t)$ plus the overhead cost $C_{uc}(u,c)$ affects the overall utilized power. The more valued low power is, the more the system will attempt to trade higher delay for lower power. In order to explore the power-delay tradeoff with this model, we simply have to consider cost combinations of the type

$$[\xi C_b(b) + (1-\xi)(C_{ut}(u,t) + C_{uc}(u,c))]$$

for various values of $\xi \in (0,1]$. For $\xi = 1$, the DP ignores any power cost. This will obviously lead to using the maximum number of available CPUs in every slot, as this increases the chances of completing a job (and is cost-free in this case), achieving the minimum possible queuing delay. If we allowed $\xi = 0$, the DP would chose no CPU and cause the average queuing delay to become infinite. Ranging ξ in $(0,1]$ we can achieve any average queuing delay from infinite to the minimum possible traded off against the corresponding power and thermal stress of the system.

E. Extensions to the model

It is easy to extend the model to include Markov modulated (bursty) Bernoulli arrivals. This, however, does not add significant intuition. In the following sections where we analyze systems with job arrivals, we simply use heuristics extracted from the previous no-arrival case and we see that they perform very well.

III. Design of Efficient Power Control Heuristics

Solving the DP recursion of section II for an actual system can be a computationally intensive task, depending on the size of the action space. It is therefore desirable to use the structural properties of the optimal solution and design *justified*, low-complexity heuristics that are efficient and have near-optimal performance. We make the following observations, based on the model of the previous section, which we should incorporate into efficient heuristics:

- If the buffer is empty, no CPU should be used ($u = 0$ if $b = 0$).
- The maximum number of CPUs to be used can not exceed the ones available $u \leq (c+a)$. However, the state t of the thermal environment can limit this number. Therefore, we will denote the maximum number of CPUs that can be used as $h(c+a,t)$. This function is system specific.
- At least one CPU should be used, if available, when the buffer is not empty ($u \geq 1 \; if \; (c+a) > 1, b > 1$). Note that this is not necessarily optimal since when the environment is in a very poor state it may be optimal to

wait for it to improve. Nevertheless, this is a reasonable approximation in most cases.

- The number of CPUs to use should increase as the buffer size increases, assuming that the other state parameters remain the same $(u(b+1,t,c,a) \geq u(b,t,c,a))$.
- When many CPUs are available, the engagement could be less aggressive $(u(b,t,c,a) \geq u(b,t,c,a')\ if\ a < a')$. It is reasonable to anticipate that most of them will very likely also be available in the next time slot to use.
- A heuristic should incorporate the parameter ξ used to 'position' the power-delay tradeoff. As ξ increases the emphasis on power cost decreases and we expect the control to be more aggressive in activating CPUs.

A heuristic control that incorporates the above concerns is given by the following formula:

$$u(b,t,c,a) = min\{h(c+a,t), \\ (b>0)(1+round(f(\xi)h(c+a,t)\frac{b}{\mathcal{B}} - \frac{a}{M-c+1}))\} \quad (3)$$

where: $f(\xi) = 2^{3\xi-1}(f([0.0\ 0.5\ 1.0]) = [0.5\ 1.4\ 4.0])$
This function induces high CPUs usage when ξ is close to one (even when the buffer is fairly empty), while it suppresses it as ξ approaches zero.

IV. Performance Evaluation

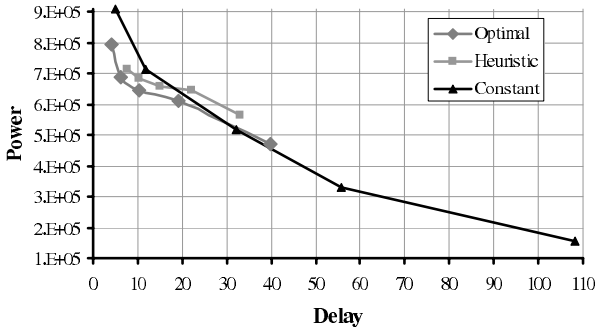
In this section we evaluate how well the optimal control and the heuristic one, developed in sections II and III respectively, work in an actual system environment. Our evaluation is based on simulations performed using OMNeT++ [9].

We consider a system with $B = 20$, $\mathcal{T} = \{1,2,3,4,5\}$, $M = 8$. We will assume that $s(u) = 1 - e^{-0.2u}$, $C_b(b) = b$, $C_{ut}(u,t) = u\sqrt{t}$ and $C_{u,c}(u,c) = 0.2(u-c)^+ + 0.1(u-c)^-$. We omit the details about the transition probabilities for the thermal and the CPU environment. We briefly say that (1) the thermal state can only change to adjacent ones (higher or lower) and (2) the probability of moving to a higher state increases as fewer CPUs are available (after decisions). Moreover, the CPU availability state switches to any of the other possible states with equal probability, smaller than the probability of remaining the same. Finally, for this system, we choose $h(c+a,t) = c+a$.

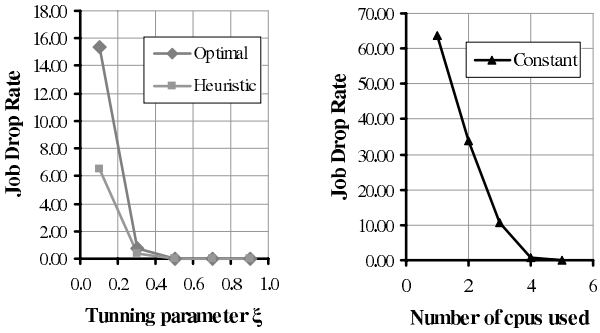
Since in actual systems we do have job arrivals, this is the case we study here. We consider two types of arrivals:

- Constant rate - The interarrival time is exponentially distributed with mean 2 time slots.
- Bursty - In this case, there are 2 kinds of alternating cycles. The length of each cycle is exponentially distributed with average length 100 time slots. In the first cycle type, the arrivals have constant rate and the interarrival time is exponentially distributed with mean 2 time slots. In the second cycle type, there are no arrivals.

In order to benchmark our work, we will further consider the case of a constant CPU usage policy. This is what usually holds in practice. Specifically, each queue is equipped with a constant number of CPUs that are powered-up at all times.



(a) Comparison of the power-cost and average-queuing-delay trade-off for the cases of the optimal policy ($\xi = \{0.1, 0.3, 0.5, 0.7, 0.9\}$), the heuristic ($\xi = \{0.1, 0.3, 0.5, 0.7, 0.9\}$) and constant CPU usage (1-5 CPUs)



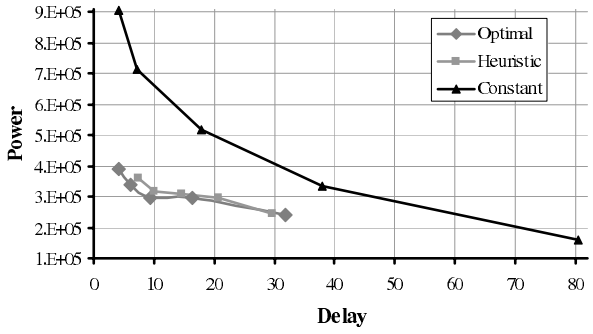
(b) The left plot presents the job-drop-rate versus ξ for the optimal policy and the heuristic. The right plot presents the job-drop-rate when a constant number of CPUs is used.

Fig. 2. Results for the case of constant arrival rate.

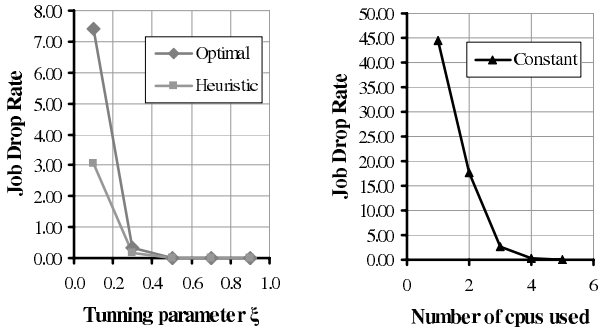
Fig.2(a) and 2(b) present our results in the case with constant arrival rate. As we see in Fig.2(a) the optimal policy slightly outperforms the heuristic in terms of the power-delay tradeoff. We also see that the constant CPU usage policy appears to outperform all others when when the number of CPUs used is less than 3 while it appears to perform worse in the opposite case. These results do not, at first, appear to meet our expectations. However, examining Fig.2(b) we discover that the job-drop-rate (overflow rate) is a lot more significant in the case of constant CPU usage. Specifically, we see that even when 3 CPUs are used, the dropped jobs exceed one tenth of the total jobs received. Naturally, this is not acceptable in a real system application.

Fig.3(a) and 3(b) present our results in the case of bursty arrivals. These results are a lot more dramatic as anticipated. Indeed in Fig.3(a) we see that in terms of power cost and power vs. delay tradeoff, the optimal policy and the heuristic perform similarly, and far better than the constant CPU usage policy. Fig.3(b) supports a similar conclusion regarding the job-drop-rate.

An important observation needs to be made here. Roughly speaking, when the arrival rate is nearly constant, there are power savings obtained by the control schemes, but they are



(a) Comparison of the power-cost and average-queuing-delay trade-off for the cases of the optimal policy ($\xi = \{0.1, 0.3, 0.5, 0.7, 0.9\}$), the heuristic ($\xi = \{0.1, 0.3, 0.5, 0.7, 0.9\}$) and constant CPU usage (1-5 CPUs)



(b) The left plot presents the job-drop-rate versus ξ for the optimal policy and the heuristic. The right plot presents the job-drop-rate when a constant number of CPUs is used.

Fig. 3. Results for the case of bursty arrival rate.

somewhat limited. This is because the buffer smooths out the load on the CPUs and, by using the proper fixed number of CPUs matching the average load, we can achieve relatively high performance. However, in real systems the arrival rate demonstrates intense variations. Real system traffic can be approximated by operating cycles within which the arrival rate may be rather constant. This is why the power control schemes we presented can achieve substantial benefits over standard approaches.

V. Conclusion

In this paper, we examined the important problem of autonomic power control for internet servers and data centers. We formulated this problem within a Dynamic Programming framework that captures the power-delay tradeoff. In addition to the solution of the DP, we designed and evaluated a practical, low-complexity heuristic algorithm. In both cases, we demonstrated significant performance gain compared to benchmark systems. The gain is greater in the regime where job arrivals exhibit substantial temporal variations. Our approach can be applied to a large range of scenarios and does not depend on the statistical characteristics of the job arrivals. However, future work will investigate ways to achieve even higher performance efficiency by utilizing proper estimation techniques for the jobs arrivals.

References

[1] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers", *Symposium on Operating Systems Principles*, October 2001.

[2] E. Pinheiro, R. Bianchini, E.V. Carrera and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems", *Technical Report DCSTR440, Department of Computer Science, Rutgers University,*, May 2001.

[3] J.D. Mitchell-Jackson, "Energy Needs in an Internet Economy: A Closer Look at Data Centers", *Masters thesis, Energy and Resources Group, University of California at Berkeley,* July 2001.

[4] J.S. Chase and R.P. Doyle, "Balance of Power: Energy Management for Server Clusters", *available at: http://www.cs.duke.edu/ari/publications/publications.html*, January 2001.

[5] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell and R. Rajamony, "The Case For Power Management In Web Servers", *in Power-Aware Computing (R. Graybill and R. Melhem, editors), Kluwer Academic Publishers in Computer Science,* January 2002.

[6] R. Bianchini and R. Rajamony, "Power and Energy Management for Server Systems", *Technical Report DCSTR528, Department of Computer Science, Rutgers University,* June 2003

[7] E. Pinheiro, R. Bianchini, E.V. Carrera and T. Heath, "Dynamic Cluster Reconfiguration For Power And Performance", *in Compilers and Operating Systems for Low Power (L. Benini, M. Kandemir, and J. Ramanujam, editors), Kluwer Academic Publishers,* September 2003.

[8] D. Bertsekas, "Dynamic Programming and Optimal Control", Athena Scientific, 1995.

[9] OMNeT++ Discrete Evemt Simulation System, publicly available at: *http://www.omnetpp.org/.*