Purdue University
Purdue e-Pubs

ECE Technical Reports

Electrical and Computer Engineering

3-21-2011

Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud

Yu-Ju Hong
School of Electrical and Computer Engineering, Purdue University, yujuhong@purdue.edu

Mithuna Thottethodi
School of Electrical and Computer Engineering, Purdue University, mithuna@purdue.edu

Jiachen Xue
School of Electrical and Computer Engineering, xuej@purdue.edu

Follow this and additional works at: http://docs.lib.purdue.edu/ecetr

Hong, Yu-Ju; Thottethodi, Mithuna; and Xue, Jiachen, "Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud" (2011). *ECE Technical Reports.* Paper 414.
http://docs.lib.purdue.edu/ecetr/414

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud

Yu-Ju Hong

Mithuna Thottethodi

Jiachen Xue

TR-ECE-11-08

March 21, 2011

School of Electrical and Computer Engineering

1285 Electrical Engineering Building

Purdue University

West Lafayette, IN 47907-1285

Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud

Yu-Ju Hong
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
yujuhong@purdue.edu

Jiachen Xue
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
xuej@purdue.edu

Mithuna Thottethodi
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
mithuna@purdue.edu

ABSTRACT

Cloud computing holds the exciting potential of elastically scaling computation to match time-varying demand, thus eliminating the need to provision for peak demand. However, the uncertainty of variable loads necessitate the use of "margins" – servers that must be held active to absorb unpredictable potential load surges – which can be a significant fraction of overall cost. Further, naively switching to an on-demand cloud model can actually degrade "true costs" (server costs that would be incurred even if margin costs disappeared) because of the fundamental economic rule wherein on-demand services/goods cost more compared to "reserved" goods/services where the user bears some commitment (i.e., on-demand customers must pay a premium in exchange for not undertaking the fixed-cost risk that committed customers undertake).

This paper addresses the twin challenges of minimizing margin costs and true costs in an Infrastructure-as-a-Service (IaaS) cloud. Our paper makes the following two contributions. To address the problem of margin costs, we make two key observations based on real Web server traces. First, rather than use a fixed margin, we observe that the margin may be load-dependent. For example, the margin required at low loads may be higher than the margin required at high loads. Second, we observe that the "tolerance" – the fraction of time when the response time target may be violated – need not be uniform across all load levels. For example, compared to a case where we satisfy requests within the target response time 95% of the time (for a tolerance of 5%) irrespective of load, one may achieve lower costs by satisfying the response time target 93% of the time at low loads and 97% of the time at high loads, while still achieving an overall 95% satisfaction ratio. We propose `ShrinkWrap-opt` which is a dynamic programming algorithm that exploits both the above observations to achieve optimal margin cost while achieving the desired (statistical) response time guarantees. To address true costs, we propose `commitment straddling` – the mixed use of reserved and on-demand machines – to achieve optimal true-cost. Simulations with real Web server load traces (including 3 months of traces from Wikimedia from Summer 2010) using the Amazon EC2 cost model reveal that our techniques save between 13% and 29% (21% on average) in cost while satisfying response-time targets.

1. INTRODUCTION

The cost/performance tradeoff faced by Web-service operators in the pre-cloud world was inelastic. One had to either incur the cost of provisioning for the peak-demand (or near-peak demand, if some modest dilution in server response time was acceptable [14]) *or* incur the cost of excessive degradation in response time. The emergence of commercially-available Infrastructure-as-a-Service (IaaS) cloud computing vendors such as Amazon EC2 has enabled a more elastic provisioning approach wherein on-demand computational resources can be "rented" at very short notice. Armbrust *et al.* provide an expanded overview of such tradeoffs in their white paper on cloud computing [3].

The cost-advantage of cloud-computing for *episodic* computation demands (e.g., one-time document digitization, hosting sites covering major sporting events) is well-understood; users with such one-time demands can avoid capital expenditure and instead utilize their financial resources solely for operational expenses. In contrast, the case for cloud computing for ongoing, day-to-day operations with long time horizons is less clear. There are many factors that may hinder cloud adoption, as described in [3]. This paper focuses on one such issue – costs incurred by the potential cloud user. The goal of this paper is to achieve significant cost savings for *normal* day-to-day computation demands and not for *episodic* computational demands.

Cost is a key concern that may limit cloud adoption. Specifically, there are two key factors that affect cost. First, because of the uncertainty of time-varying loads, operators are forced to maintain a margin – a pool of servers beyond the expected load. Minimizing such margin cost is important. Second, cloud vendors such as Amazon EC2 offer services at various commitment levels. For example, at the lowest commitment level, there are on-demand instances[1] in which machine instances are acquired on an hourly basis with no longer-term commitment at all. At higher levels, there are the "reserved instances" wherein the user may pay an upfront fixed cost to ensure discounted hourly pricing for various durations (e.g., 1 year, 3 years). Minimizing cost

[1]We ignore the "spot instances" which are similar to on-demand instances except that their availability is not guaranteed and they are dynamically priced. Such resources with no guarantees of availability at predictable pricing may be more appropriate for episodic computing demands.

by acquiring machine instances at the cost-optimal commitment level for time-varying loads is also an important challenge.

This paper makes two key contributions to reduce both the above costs for cloud users. Our first contribution is a technique to determine margins in such a way that margin costs are minimized under a given load volatility model. The technique has two innovations based on two observations we made in the request traces of real workloads. First, we observed that the margin requirements vary by load. Unlike traditional load-oblivious margin mechanisms which use some fixed arithmetic transformation on the load to compute margins (e.g., translation with a fixed offset for constant margins, scaling with a fixed ratio for linear margins), our `ShrinkWrap` technique uses a table-lookup to provide customized, load-dependent margins. `ShrinkWrap` reduces wasted margins by avoiding the one-size-fits-all approach. Our second observation was driven by the fact that systems typically have some "tolerance" – the fraction of time where response time targets may not be met. Our second observation was that the way in which the tolerance budget is expended affects cost because using the tolerance at some loads may result in more cost savings than at other loads. We develop a dynamic programming algorithm to optimally expend the tolerance budget to achieve maximum margin cost savings.

Our second contribution addresses the true costs (as distinct from margin costs described above) of serving requests by appropriately choosing commitment levels. We demonstrate that `commitment straddling` – the employment of both reserved and on-demand servers – is fundamentally necessary to minimize cost, while meeting performance requirements. To understand why such commitment straddling is cost-optimal, we may conceptually view time-varying loads as inducing varying utilization in a collection of servers with some servers being heavily loaded and others being lightly loaded. Combining such variation in utilization with the well-known notion that reserved instances are less expensive than on-demand instances when high utilization is expected (say, utilization beyond a break-even ratio), we can divide the servers into two classes – those with higher utilization than the break-even ratio and those with lower utilization than the break-even ratio. Naturally, the optimal cost configuration will employ reserved servers for the first class and on-demand servers for the second class. We show that cost-optimal commitment straddling can be computed if we know the load frequency distribution. Intuitively, one may think that commitment straddling is the equivalent of using reserved instances for the average load and on-demand instances for the peak load. However, our precise analysis provides a stronger result. For example, our results show that it takes a grossly underutilized workload (with more than 50% idle-time), for an all-on-demand configuration to be the optimal. Similarly, it takes a workload where the peak load is sustained for nearly 50% of the time for the all-reserved configuration to be cost optimal.

Both the above optimal cost techniques assume, in their proofs of optimality, that (1) workloads have known statistical behavior (frequency distributions), and that (2) the cloud model is ideal (fine-grained rental granularity). The first assumption is reasonable because (a) workloads behaviors indeed have stable statistical behavior, and (b) it is impossible to optimize for an unknown workload. Our second assumption is needed to simplify the analysis. However, evaluation using practical (i.e., non-ideal) conditions reveals significant cost reductions from each of the two techniques, individually and in combination. Our techniques can save between 12% and 29% in cost (21% on average) while satisfying response-time targets for a range of real server traces. Specifically, we show that a 14.5% cost saving is possible for one of the world's top ten Websites (Wikimedia).

In summary, the two primary contributions of this paper are:

- We develop `ShrinkWrap-opt`, which combines two new techniques to achieve optimal-margin-cost for a given statistical model of load-volatility. First, `ShrinkWrap` reduces wasted margin costs by using a load-dependent margins instead of fixed, load-oblivious margins. Second, our dynamic programming approach provides an optimal solution to the problem of exploiting quality-of-service tolerance to minimize costs in `ShrinkWrap-opt`.

- We show that `optimal commitment straddling` – the combined use of reserved machines to serve part of the load and on-demand machines to serve the remainder of the load so as to minimize cost – is possible if the load frequency distribution is known.

The rest of the paper is organized as follows. Section 2 defines terms used in the rest of the paper. Section 3 describes the margin savings via ShrinkWrap-opt. Section 4 discusses true-cost minimization via straddling. Section 5 describes our evaluation methodology. Section 6 discusses experimental results. Related work is described in Section 7. Finally, Section 8 concludes this paper.

2. TERMINOLOGY

We use the terms "machines" and "servers" interchangeably to mean a virtual machine instance in the cloud. We refer to machines where the user assumes the fixed-cost risk (by using reserved instances in the cloud) as *reserved machines*. We refer to on-demand machine instances where the user only pays for machine-hours that are used as *on-demand machines*. We use lower case c with the appropriate subscripts to denote hourly costs and upper case C with the appropriate subscript to denote aggregate costs over the duration of a workload.

The hourly costs of an active on-demand machine instance and an active reserved machine instance are c_{od} and c_{rs} respectively. Because the cloud vendor assumes underutilization risk for on-demand instances, the c_{od} is always higher than c_{rs}. We refer to the difference between the on-demand cost and reserved cost as the *on-demand premium* ($= c_{od} - c_{rs}$).

The existence of the on-demand premium does not imply that reserved machines are always better than on-demand machines because unlike c_{od}, which is charged only when machines are rented (and the machines are rented only when they are to be used), the fixed part of c_{rs} is incurred regardless of whether the machine is actively used or not. To incorporate the above notion, the hourly cost for an active reserved machine c_{rs} may be broken down to two components: hourly operational cost c_{op} and hourly fixed cost c_{fix} (i.e., $c_{rs} = c_{op} + c_{fix}$). The distinction between fixed costs and operational costs becomes relevant when we shut down a reserved instance when not in use. For such cases, we charge the fixed cost but not the operational cost. The terms and their meanings are summarized in Table 1. Further, Table 1 also includes the pricing values for Amazon EC2 for

Table 1: Notations

Symbol	Price	Description
c_{od}	0.680	hourly cost of an on-demand instance
c_{rs}	0.448	hourly cost of an active reserved instance; $c_{rs} = c_{fix} + c_{op}$
c_{fix}	0.208	hourly fixed cost of a reserved instance
c_{op}	0.240	hourly operational cost of a reserved instance
C_{rs}	-	aggregate cost of `all-reserved` configuration
C_{od}	-	aggregate cost of `all-on-demand` configuration
f	-	utilization ratio of a server
f_0	-	break-even utilization ratio

the various terms in dollars-per-hour for an extra-large machine instance, assuming a 1-year commitment on October 10th, 2010. The hourly fixed cost is computed by dividing the dollar-cost of the machine reservation by the number of hours in a year.

The configuration which uses only on-demand machines is referred to as the `all-on-demand` configuration and its aggregate cost is represented by C_{od}. Similarly, the configuration which uses only reserved-machines is referred to as the `all-reserved` configuration and its aggregate cost is represented by C_{rs}. We focus solely on computing costs because it remains a barrier to cloud adoption; disk and network bandwidth costs are already more attractive in the cloud [3].

Model of Operation.

We assume that requests must be satisfied within a target response time. The precise model we use to relate load, response time, and number of machines is described in detail in Section 5 and is not important for the exposition of our ideas. For now, it is sufficient to assume that the number of machines needed is monotonic with respect to load (i.e., heavier loads need more machines). Because there is some monotonic mapping from load to the number of machines needed to satisfy that load within its response time, we often specify loads in terms of machine units. For example, we use the phrase "a load of n machines" to mean the server load (in terms of requests per unit time) that can be served by n machines while satisfying the target response time.

3. MITIGATING MARGIN COSTS

This section addresses margin costs, which arise due to *uncertainty about load variation.* While nominal load prediction using various techniques have been proposed [6, 8], there is typically an error distribution around the predicted values because of the unpredictable nature of fine-grained load variations. Because of the possibility of underprediction, a common practice is to speculatively maintain a *margin* – a pool of servers beyond the predicted number of servers that are available to handle underprediction. An obvious tradeoff here is that larger margins maximize the probability of satisfying the target response time, but at a higher cost. The goal of our technique is to minimize the margin requirements.

To describe our margin-minimization techniques, we first describe the model of operation we assume, which includes the following three assumptions. First, we frame the problem of determining margins as the problem of determining the number of machines to keep active (a_{i+1}) at the beginning of the $(i+1)^{th}$ time interval given that we have some predicted load for the $(i+1)^{th}$ interval (say p_{i+1}). In general, p_{i+1} may depend on the prediction model used and may depend on model-specific parameters which may in turn be dependent on parameters such as prior observed loads (say m_i, m_{i-1} etc.), the time-of-day, and so on. While, our technique is orthogonal to the prediction mechanism, we use an auto-regressive moving average model similar to those used in prior literature for data-center load prediction [6]. Note, because the predicted number of servers must be active at the beginning of the $(i+1)^{th}$ interval, and because the prediction may require information from the end of i^{th} interval, our model implicitly assumes instantaneous machine startup. (We incorporate realistic startup time in our evaluation). Second, we assume that, for each predicted load-level, the distribution of prediction errors (i.e., differences of the actual load from the predicted load) is known. We refer to the distribution as the *error* distribution and it serves as our model for load-dependent volatility. Note, such an assumption is not unique to our method. Existing methods that use fixed margins implicitly assume knowledge of such error distributions as well. Finally, maintaining margins to satisfy *all possible* loads may be expensive and impractical. Just as it is undesirable to provision machines for peak loads, it is also undesirable to provision margin machines for peak volatility. Consequently, margin mechanisms typically provision margins to achieve statistical quality of service (e.g., response time targets must be met 95% or 99% of the time) under the assumed *error* distribution. We refer to the time intervals where the response time need not be satisfied as the *tolerance* of the system. We define the *satisfaction ratio* to be the fraction of time intervals where the response time target is met. For example, if the *satisfaction ratio* requirement is 99%, the *tolerance* is 1%.

Operationally, margin mechanisms serve two key functions. First, they determine the number of machines needed in the interval (a_i) based on the predicted load for that interval (p_i) and the expected volatility (i.e., *error* distribution).

The second function of margin mechanisms is to choose where the "tolerance budget" is spent (i.e., the choice of when response time targets may be violated), which is implicitly decided when a_i is inadequate to serve the tail of the *error* distribution. Recall our key observation that spending the tolerance budget uniformly may not yield the optimal margin cost.

Figure 1 illustrates how margin mechanisms achieve the two functionalities. Figure 1 plots the error distribution (Y-axis) at various predicted loads (X-axis) for a simple example. Note that the error distribution is plotted in terms of a distribution of absolute loads around the predicted load (rather than as a distribution of differences from predicted value). The error distribution for a given load is represented by the range of possible values the load may take (boxes stacked along the Y-axis that are populated with dots) and the frequency of those values (the number of dots in each box). The position of the dots within a box is not meaningful. For example, the figure highlights the error distribution for a predicted load of 2 machines. It shows that the probability that the actual load in the time interval being 2 machines (four dots) is twice that of the load being 1 machine (two dots).

To contrast our approach with prior approaches, consider that we want to minimize margin costs for the set of error distributions shown in Figure 1 with a tolerance of two

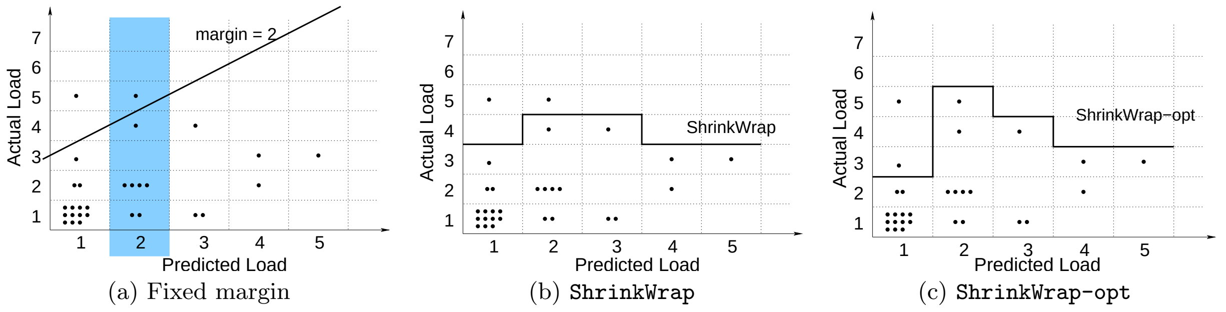


Figure 1: Visualizing margin costs

time intervals where we may violate the response time target. Prior work has proposed the use of constant margins [1] under the implicit assumption that the error distributions of all loads are the same. Such a fixed margin may be graphically interpreted as the line $a_i = p_i + c$ shown in Figure 1. The line achieves both the functions by (1) assigning a value for the margin ($c = 2$), and (2) omitting the two intervals that are to the top-left of the line.

The choice of using uniform margins can cause significant wastage when the volatility is non-uniform across loads. For example, in Figure 1, the margin of -1 machines is adequate when the predicted load is 4 machines because the actual loads all lie at or below 3 machines. However, the margin is forced to a higher value of 6 at a load value of 4 machines because of the error distribution at other load levels. We wish to avoid such a one-size-fits-all approach. Our ShrinkWrap eliminates such wastage by setting the load-dependent margin in an arbitrary, per-load manner by using a table lookup (i.e., $a_i = table[p_i]$) instead of using less flexible approaches such as translation (i.e., $a_i = p_i + c$ for constant margins) or other arithmetic transformation (e.g., scaling for linear margins $a_i = \alpha p_i$). ShrinkWrap's approach minimizes wasted margin costs since margins are customized for each predicted load which enables a contour-hugging margin curve (as shown in Figure 1(b). Note, in Figure 1(b), `ShrinkWrap` uses its tolerance budget in exactly the same way as the `FM` approach.

While the above example motivates the use of load-dependent margins using a toy example, the technique is driven by real world traces. Figure 2 illustrates the error distribution at two different load levels (load expressed as a range of machines, the two curves) for the Wikimedia Web traces (described in Section 5) with error on the X-axis and frequency on the Y-axis. As can be seen, the error distribution is not uniform across loads.

ShrinkWrap decouples the two functions of a margin mechanism. We may freely choose where we truncate the tail of the error distribution (to exploit tolerance) and wrap the margin curve around what remains. In the remainder of this section, we design a dynamic programming algorithm to obtain optimal margin costs under the ShrinkWrap approach for a given tolerance and for a given set of load-dependent error distributions. Our algorithm achieves optimal cost by using two mechanisms: (1) careful choice of where to expend the tolerance budget *and* (2) use of the ShrinkWrap approach. The use of two different mechanisms throws open an interesting question on the relative value of the two factors. Later, we answer this question by considering non-optimal heuristics (Section 6.4).

3.1 Optimal margin minimization

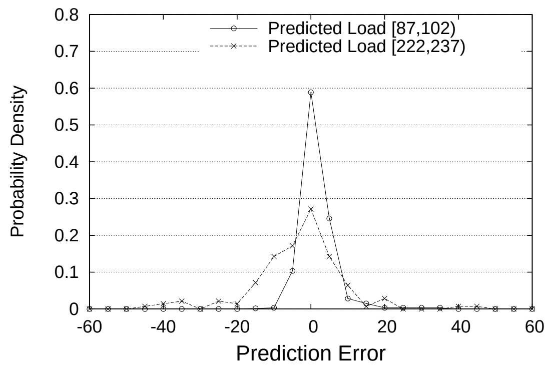


Figure 2: Dependence of margins on load level

Before we proceed to our margin minimization algorithm, we make two observations. First, consider the cost savings that accrue by using tolerance. Consider the error distribution of load $p_i = 2$ in Figure 1. To achieve 100% coverage, the number of machines a_i would have to be 5 machines to satisfy the maximum load possible. However, by choosing not to satisfy the response time for the extreme end-point in the distribution, we can set the a_i for $p_i = 2$ to be 4 (which is the next populated box). Thus, tolerance reduces the margin for this particular predicted load by 1 machine. Further, because the margin is used as many times as there are instances in the error distribution (eight, because there are eight dots in the shaded region), the total cost savings equals 1×8 machine-intervals (an occupancy metric similar to machine-hours). Machine-time is the metric our algorithm minimizes. True costs will differ by some scaling values depending on whether the saved machine-intervals were on-demand or reserved. With the above understanding of the cost savings from tolerance, we can now define our optimization problem. Figure 1(c) illustrates the margins that achieve the optimal cost. Compared to Figure 1(b), where the cost savings is 38 machine-intervals ($= 2 \times 15 + 1 \times 8$), the optimal margin reduces cost by 45 machine-intervals ($= 3 \times 15$).

We formally cast the margin minimization problem in terms of (a) the tolerance R and (b) a collection of error distributions (which are represented in $\mathbb{L}$ and the collection of L different $\mathbb{N}_i$s, as listed below). Note that the distributions are expressed using discrete counts rather than fractional probabilities.

$\mathbb{L}$: a vector to represent the unique load levels.
L: the length of the vector $\mathbb{L}$.
$\mathbb{N}_i$: the vector of actual loads for $\mathbb{L}(i)$. The vector $\mathbb{N}_i$ is sorted ascendingly.
k_i: the length of vector $\mathbb{N}_i$.
R: the tolerance (in number of intervals).

We define a matrix $\mathbb{P}$ (our dynamic programming matrix) of dimensions $L \times R$ where L and R are as defined above. We define $\mathbb{P}(i,j)$ as the choice of margins that maximizes the cost savings (compared to a zero-tolerance design) when considering the first i unique loads and using a tolerance of j time intervals.

With this definition, the bottom-up computation implicit in dynamic programming can be specified in terms of the initial conditions (to define the boundary conditions) and the recurrence (to bootstrap solutions to bigger problems in terms of solutions to smaller problems).

For initialization, consider the first row of the $\mathbb{P}$ matrix. Because it deals with a single error distribution, the cost savings for various tolerances is computed using a similar process as described earlier in this section. In general, a tolerance of j implies we can afford to lop off the top j elements in the next-load distribution of the first load (i.e., elements $\mathbb{N}_1(k_1)$ through $\mathbb{N}_1(k_1-j+1)$) and determine the number of machines based on what remains in the distribution. Recall that the number of machines is also scaled by the number of time intervals k_1 to count savings over all k_1 intervals.

The initializations corresponding to the above intuition are shown in Equations 1 and 2. There are two cases to handle the corner cases such as the tolerance exceeding the number of intervals in the error distribution (second choice in Equation 1) and vice versa (first choice in Equation 1 and all of Equation 2).

Initialization:
Case 1: $k_1 \leq R$

$$\mathbb{P}(1,j) = \begin{cases} [\mathbb{N}_1(k_1) - \mathbb{N}_1(k_1-j)] \times k_1 & \text{where } j < k_1 \\ \mathbb{N}_1(k_1) \times k_1 & \text{where } k_1 \leq j \leq R \end{cases} \tag{1}$$

Case 2: $k_1 > R$

$$\mathbb{P}(1,j) = [\mathbb{N}_1(k_1) - \mathbb{N}_1(k_1-j)] \times k_1 \tag{2}$$

To bootstrap solutions to bigger problem sizes, we note that the optimal solution for arbitrary $\mathbb{P}(i,j)$ must necessarily be one of the following exhaustive set of possibilities. The first possibility is that *none* of the tolerance budget is spent on $\mathbb{N}_i$, which implies that *all* of the tolerance is spent on the earlier error distributions (i.e., $\mathbb{P}(i-1,j)$). The second possibility is that *exactly one* interval of the tolerance budget is spent on $\mathbb{N}_i$, which implies that *all-but-one* of the tolerance is spent on the earlier next-load distributions (i.e., $\mathbb{P}(i-1,j-1)$). And so on. The set of possibilities terminate when either we run out of tolerance budget (i.e., j, as in Equation 3 or R as in Equation 5) or we run out of intervals in the error distribution (i.e., k_i, as in Equation 4). Because we can evaluate the cost savings from the known solutions for smaller problem sizes and from our knowledge of how tolerance affects cost savings in a single error distribution, the exhaustive set of choices may be compared to pick the optimal choice.

Bootstrapping, for $2 \leq i \leq L$:
Case 1: $k_i \leq R$
if $j < k_i$,

$$\mathbb{P}(i,j) = max \begin{cases} \mathbb{P}(i-1,j) \\ \mathbb{P}(i-1,j-1) + [\mathbb{N}_i(k_i) - \mathbb{N}_i(k_i-1)] \times k_i \\ \vdots & \vdots \\ [\mathbb{N}_i(k_i) - \mathbb{N}_i(k_i-j)] \times k_i \end{cases} \tag{3}$$

if $k_i \leq j \leq R$

$$\mathbb{P}(i,j) = max \begin{cases} \mathbb{P}(i-1,j) \\ \mathbb{P}(i-1,j-1) + [\mathbb{N}_i(k_i) - \mathbb{N}_i(k_i-1)] \times k_i \\ \vdots & \vdots \\ \mathbb{P}(i-1,j-k_i) + \mathbb{N}_i(k_i) \times k_i \end{cases} \tag{4}$$

Case 2: $k_i > R$

$$\mathbb{P}(i,j) = max \begin{cases} \mathbb{P}(i-1,j) \\ \mathbb{P}(i-1,j-1) + [\mathbb{N}_i(k_i) - \mathbb{N}_i(k_i-1)] \times k_i \\ \vdots & \vdots \\ [\mathbb{N}_i(k_i) - \mathbb{N}_i(k_i-j)] \times k_i \end{cases} \tag{5}$$

Once the recurrence computation is complete and the matrix $\mathbb{P}$ is fully populated, the entry $\mathbb{P}(L,R)$ provides the maximum cost savings possible. Since we are interested in the choice of time-intervals that are tolerated in the optimal cost configuration (and not only in the cost savings from such a configuration), we must introduce auxiliary data-structures to remember our optimal choices in the max operator. Because such auxiliary data structures can be added in a fairly mechanical manner, we omit details thereof.

Complexity.

The above algorithm must populate $L \times R$ elements performing a maximum of R computations per element. Therefore, the worst-case complexity is $O(LR^2)$. Note that the R term introduces a pseudopolynomial element because the complexity is expressed in terms of the value of R whereas the input R is provided in $logR$ bits. However, the practical values of R are small because R is typically 1% of time intervals over which the next-load distribution is expressed. In practice, the algorithm runs in seconds while analyzing fairly large distributions.

4. MITIGATING TRUE COSTS VIA COMMITMENT STRADDLING

Consider the well-known tradeoffs of reserved instances vs. on-demand instances. If expected utilization is low, reserved instances incur unnecessary fixed costs for the entire duration. However, on-demand achieves lower costs by paying a small premium to avoid the fixed costs. In contrast, at high utilization, the on-demand premium is unnecessarily incurred for the entire duration; making it more attractive to use reserved instances. There exists a break-even utilization ratio where the cost of a reserved instance equals the cost of an on-demand instance.

The above intuition can be quantified in the context of the utilization of a single server. Consider an ideal case where machine startup/shutdown in the cloud is instantaneous. For a server that is used for a fraction f of time, the aggregate cost of a reserved instance (C_{rs}) and an on-demand instance (C_{od}) are given by $C_{rs} = c_{fix} + c_{op} \times f$ and $C_{od} = c_{od} \times f$, respectively. Equating C_{rs} and C_{od}, we can solve for the break-even utilization ratio f_0 as $f_0 = c_{fix}/(c_{od} - c_{op})$. At utilization ratios higher (lower) than

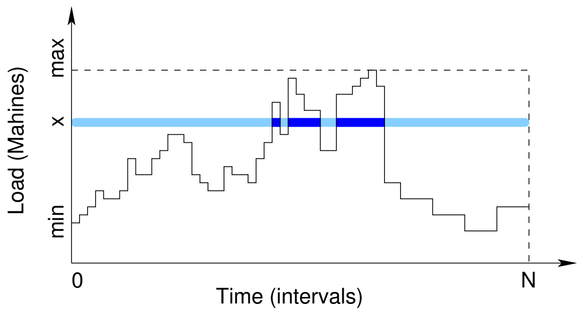


Figure 3: True Cost Savings

f_0, it is cheaper to use a reserved (on-demand) machine instance. Note, for the Amazon EC2 pricing structure shown in Table 1 f_0 is approximately 0.47.

Optimizing true costs.
Extending the above analysis to a collection of machines that are serving time-varying loads, we make two observations. First, under time-varying workloads, a collection of machines can be imagined to have varying utilization by using the *spatial variation* view described in Section 1. Consider an ordered collection of k machines $< m_1, m_2, m_3, \ldots m_k >$. Imagine that incoming server requests are routed in the specific machine order such that requests spill to the machine m_i only after all machines m_j $(j < i)$ are at capacity[2]. Figure 3 illustrates the application of the above model for an example load trace in which the curve plots the time-varying load (Y-axis) over N discrete time intervals (X-axis). The load-allocation model discussed above assumes that, in any interval, the machine i serves all requests corresponding to the load in the semi-closed interval $(i-1, i]$, because of which, the Y-values of the curve are all integers. As shown in Figure 3, the utilization ratio (f_x) of an arbitrary x^{th} machine is the ratio of the sum of widths of the dark-shaded areas to the width of entire duration of the trace (i.e., N). Note, the utilization of the i^{th} machine is less than the utilization of the j^{th} machine if $i > j$ because of the way the allocation model works; there cannot be a time interval where the i^{th} machine is utilized but j^{th} machine is not.

Second,the utilization of the i^{th} machine can also be interpreted as the fraction of time for which the load is *at least* i machines. Graphically, an equivalent statement would be to say that the height of the curve in the dark-shaded region in Figure 3 is at least i which is obviously true. Consequently, a cumulative distribution that plots load levels (on the X-axis) against the fraction of time intervals that meet or exceed that load level (on the Y-axis) is *equivalent* to a curve that plots the utilization ratio (Y-axis) of the i^{th} machine (X-axis) under our load-allocation model. Such a CDF will start with a value of 1 at a load of zero machines since obviously the entire duration has a load of zero or higher. Further the curve is monotonically decreasing with an eventual value of zero beyond the maximum load.

The above two observations directly lead to the design of our optimal-cost configuration, which we refer to as the `straddle` configuration. The first observation answers the question of *what* the optimal cost configuration is. If the utilization of the i^{th} machine is known, then the cost optimal configuration is to reserve n machines such that the utilization of the n^{th} machine is no less than f_0 (the break-even ratio) and the utilization of the $(n+1)^{th}$ machine is less than f_0. The load can then be served on reserved machine instances, to the extent possible and on on-demand machines for loads that exceed the capacity of the reserved machines.

[2]This is purely an academic exercise. We will not use such strict machine-by-machine ordered load allocation in practice.

The second observation tells us *how* such an optimal-cost `straddle` configuration can be constructed in a practical way. The example in Figure 3 assumed perfect knowledge of the future load levels to obtain the utilization curve. In contrast, the equivalence of the CDF to the machine-utilization curve implies that we only need to know the load frequency distribution of the future load to construct the utilization curve. We outline the two-step constructive method to develop the optimal-cost `straddle` configuration below.

- Using the known model of load frequency distribution, construct the cumulative distribution function that maps load X to fraction of time the load is expected to be at least X machines.

- Let the point where the above curve intersects the horizontal line defined by $y = f_0$ be (x_{opt}, f_0). Recall, f_0 is the break-even utilization ratio. A `commitment straddling` configuration that reserves x_{opt} machines and uses on-demand machines for the remainder is the optimal cost configuration. The proof is trivial from our above discussion because, by our method of construction, all machines beyond the x_{opt}^{th} machine have utilization lower than f_0 and all the reserved x_{opt} machines have a utilization of at least f_0.

One interesting observation that flows from the above analysis is the strong conclusions we can draw from it. The *only* way for an `all-reserved` configuration to be cost-optimal is if the machine with the lowest-utilization achieves higher utilization than the break even utilization ratio $f_0 = 0.47$. For Amazon EC2 cost parameters, this implies that the peak-load must be sustained for nearly half the time for the `all-reserved` configuration to be cost-optimal. Similarly, the *only* way for an `all-on-demand` configuration to be cost-optimal is if the machine with the highest-utilization (i.e., m_1 in our machine-by-machine load allocation strategy) has a utilization ratio lower than $f_0 = 0.47$. For Amazon EC2 cost parameters, this implies that a load must have more than 53% idle time for the `all-on-demand` configuration to be cost-optimal.

Finally, we note that once the `straddle` configuration is finalized, there is no need to allocate requests in a strict machine-by-machine order as assumed in the conceptual analysis. It is adequate if we make sure that incoming server requests are served on reserved machines before being farmed out to on-demand machines. Within the reserved machines, we may vary the load assignment for other considerations such as wear-leveling, load balance, and so on.

5. EXPERIMENTAL METHODOLOGY

We use an in-house trace-driven simulator that models a cloud vendor as seen by cloud clients. Our simulator assumes that on-demand machine instances can be started up in 10 minutes. This includes the queuing delay while a machine-startup request waits in the cloud vendor's request queue.

Table 2: Characteristics of the Web traces

Trace	Year	Length (days)	Avg-to-Peak Ratio	Num reserved
AnonUniv	2010	156	0.0690	146
Clarknet [10]	1995	14	0.5148	225
NASA [10]	1995	62	0.1551	240
UCBerkeley [10]	1996	18	0.5656	247
Wikimedia [15]	2010	92	0.4567	191

Further, we mimic Amazon EC2's minimum rental granularity of one hour (except where specifically modified to study the impact of rental granularity).

Our simulator models the costs for on-demand and reserved machines based on Amazon EC2 tariffs on October 10th, 2010. Specifically, we use on-demand instance costs directly. We use the costs of a "reserved instance" with a commitment period of 1-year for the reserved-machine. In both cases, we use numbers from the "Extra large instance". The reserved instance costs include an up-front fixed cost which we converted to an hourly cost as described earlier in Section 2. Based on the above decisions, our normalized cost/hr ratios for on-demand (c_{od}), active reserved ($c_{rs} = c_{fix}+c_{op}$), and inactive reserved machines (c_{fix}) were 1, 0.66, and 0.31, respectively (see Table 1). Our sensitivity studies (discussed briefly in Section 6.4) reveal that varying the ratios does not significantly alter our results.

We use the Allen-Cunneen approximation formula [2, 4] for the GI/G/m model to obtain the mean response time of the requests in each 10-minute interval of the traces. A GI/G/m queue models an m-server queuing system serving requests with general arrival and service time distributions. In the Allen-Cunneen approximation, the mean response time is the sum of the mean service time and average waiting time, detailed as follows:

$$\overline{W} = \frac{1}{\mu} + \frac{P_m}{\mu(1-\rho)} \cdot \frac{C_A^2 + C_S^2}{2m} \tag{6}$$

where
$\overline{W}$ is the mean response time,
μ is the mean service rate of a server,
λ is the mean request arrival rate,
$\rho = \frac{\lambda}{\mu m}$ is the average utilization of a server,
m is the number of servers available to serve the requests,
$P_m = \rho^{\frac{m+1}{2}}$ for $\rho \leq 0.7$ and $P_m = \frac{\rho^m+\rho}{2}$ for $\rho > 0.7$,
C_A and C_S are the coefficients of variation of request inter-arrival times and service times, respectively.

The service time varies upon different sizes of requested data. We assume service time of 3 ms for the first 100 KB of data, and then increase it linearly with the data size. We also use formula (6) to map loads to the number of servers required to satisfy a target response time. In our simulations, we assume the target response time ($\overline{W}$) is 6 ms, consistent with [1, 6].

Workloads.

We use five different traces (see Table 2) to drive our simulator. We use two newer traces: one obtained from a subset of server logs of an unnamed university's Website which includes the Web presence of 10+ academic departments. We also use load traces from Wikimedia group of Websites for 3 months from June 15th, midnight through September 15th, midnight, 2010. The Wikimedia trace is derived from publicly-posted request statistics [15]. The trace does not contain actual requests. Instead, it provides the aggregate load level (in requests per second) over intervals of 10 minutes each in graphical format. (We standardized all traces to the same granularity for uniformity.) Because Wikimedia request statistics are posted graphically and because our request for raw data went unanswered, we extracted the data from graphs using graph-data interpretation software (Engauge Digitizer v4.1 [7]) that interpolates values to datapoints based on user-driven calibration of axis values. Because of the relatively "zoomed-in" scale of the graphs and because of the fine resolution of points in the graphs, we estimate that errors are under 0.5% of true load levels. Since Wikimedia statistics reveal only the request rate, we model the coefficient of variation of inter-arrival times and the distribution of requested data size after the most recent trace we have (AnonUniv, which is also from 2010). The three remaining traces (Clarknet, UC Berkeley, NASA) are relatively old [10]. Because these older traces have very low load, we scaled the inter-arrival times uniformly to reach 60,000 requests per second, which is the average load from the Wikimedia trace. The relative load levels in the original traces are unaffected. The request sizes were assumed to have the same distribution as in the original trace. We include a graphical view of our traces in the appendix.

Key Comparisons.

Our techniques can be imagined to be two independent dimensions. On one dimension, we have margin-minimization policies. We compare our `ShrinkWrap-opt` against the `FM` base case. Recall that `FM` is similar to the Surgeguard mechanism [1]. Even though Surgeguard has a 2-tier mechanism, the net effect of Surgeguard is nearly identical to our `FM` because both policies have fine-grained replenishment of margin nodes. There are some second order differences (5-minute intervals vs. 10 minute intervals, shutdown at hour-boundaries without a minimum duration vs. shutdown with minimum rental period) which should not materially affect our results.

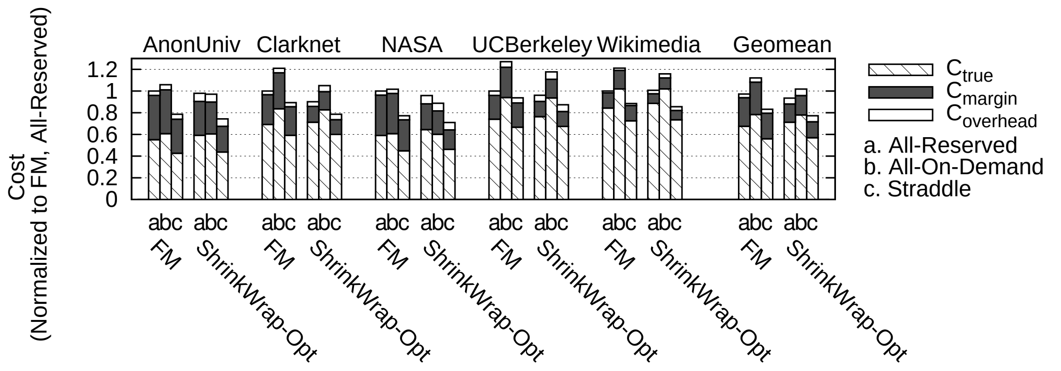
On the other dimension, the commitment policies affect true costs. We compare our **`straddle`** policy with the **`all-reserved`** and **`all-on-demand`** policies. For the **`straddle`** configurations we arrive at the number of reserved-machines by using an ideal model, even when using practical configurations with realistic startup times and rental granularity. Because the numbers obtained from ideal analysis may not be the best configuration under practical conditions, we perform a small search in the region indicated by the ideal analysis to fine-tune the number of reserved-machines. The number of machines reserved for each workload is shown in the right-most column of Table 2.

Load Prediction.

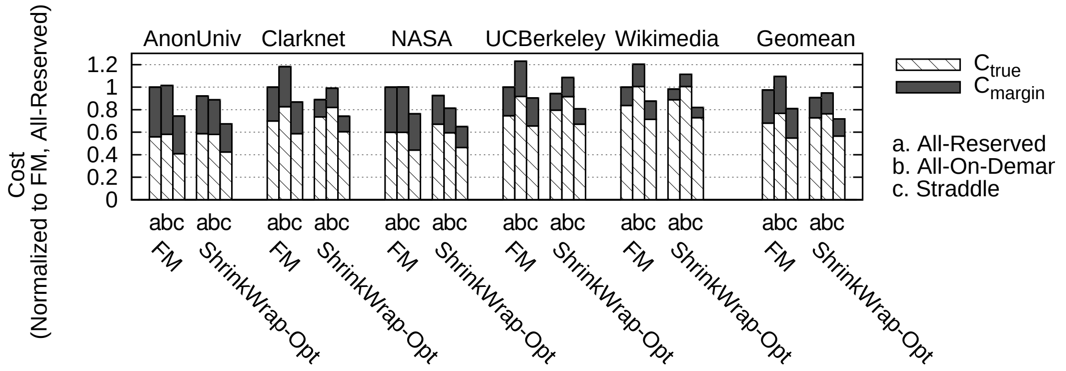
We use an autoregressive moving average model for workload prediction. The model is of the form:

$$Y_t = \sum_{i=1}^{p} a_i Y_{t-i} + \sum_{i=1}^{q} c_i \varepsilon_{t-i} + \varepsilon_t, \tag{7}$$

where $Y_{t-1}...Y_{t-p}$ are previous output values, $\varepsilon_t...\varepsilon_{t-q}$ are white noise disturbance values, and a_i and c_i are parameters obtained from training the model with the traces. The parameters p and q are the orders of autoregressive and the order of moving average terms, respectively. We use $p = 4$



(a) 1-hour rental granularity



(b) 10-minute rental granularity

Figure 4: Total costs; 1% tolerance

and $q = 2$ in the experiments. Higher orders result in significantly diminishing returns (i.e., little impact on prediction errors) for our traces.

6. RESULTS

The primary results of our evaluation are as follows.

1. **Margin-cost reduction:** `ShrinkWrap-opt` is the best practical margin minimization policy which achieves 38% lower margin costs.

2. **True cost savings:** The `straddle` configuration achieves, on average, 21% and 27% lower true cost than the `all-reserved` and `all-on-demand` configurations, respectively, while achieving the same (or better) satisfaction ratios.

3. Taken together, the two techniques yield cost-reductions between 13% and 29% (21% on average).

In addition to the above primary results, Section 6.4 presents additional results on sensitivity, cost-tolerance tradeoffs, and the impact of imperfect statistical models.

6.1 Total Cost Savings

In this section, we evaluate the total cost savings assuming 1% tolerance. Figure 4 plots the total cost (Y-axis) for each of traces (and the geometric mean) assuming 1 hour rental granularity (Figure 4(a)) and 10-minute rental granularity (Figure 4(b)). For each trace, we include two subgroups of bars (one for `FM` and another for `ShrinkWrap-opt`) with three bars in each subgroup (one each for the three commitment policies). Each bar is subdivided into subbars to indicate true cost (the cost incurred by the fraction of servers that were actively serving requests), margin cost (the cost of servers that were active, but did not have requests to serve) and overheads (the cost of machines beyond the margin which exist solely because they cannot be shutdown due to rental granularity).

The following four observations can be made from the graph. First, `ShrinkWrap-opt` provides total cost savings over `FM` across all machine acquisition policies because margin costs are always incurred (either as unnecessary operational costs in reserved machines or unnecessary rental costs in on-demand machines). However, in the remainder of this section, we focus on the `straddle` policy because it minimizes true costs (as shown later). Second, on average, `ShrinkWrap-opt` reduces the margin costs by 38% over `FM` in the practical case with 1 hour rental granularity. With 10-minute rental granularity, the margin cost reduction is 42%. Third, `ShrinkWrap-opt` increases the overhead in the 1 hour granularity configuration because it attempts to shut down machines more frequently than `FM`. When these attempts fail because of minimum rental granularity, the machines contribute to overhead costs by 7%. Effectively, the rental granularity prevents some of the efficiency of `ShrinkWrap-opt` from translating to cost reductions. Finally, the incremental total cost reduction of `ShrinkWrap-opt` over `FM` for `straddle` is 7%. The absolute total cost reduction of both `straddle` and `ShrinkWrap-opt` over the `all-reserved` configuration with `FM` is 21%.

We have also evaluated `ShrinkWrap-opt` with 5% tolerance and seen reduced benefits (10% margin reduction, 5% overall reduction; not shown). The reduction in improvement is an interplay of two different trends that we discuss later in Section 6.4.

6.2 Commitment Straddling

To focus on true costs, we assume that the loads can be perfectly predicted for all configurations, thus eliminating the need for margin costs.

Figure 5 plots the true cost (Y-axis) of each of the configurations (individual bars within groups) normalized to that of the `all-reserved` configuration for each of the Web traces (groups of bars on the X-axis) for the 1 hour rental granularity (Figure 5(a)) and the 10 minute rental granularity (Figure 5(b)) cases. Further, each bar illustrates the breakdown of on-demand costs and reserved costs (which is the sum of the two sub-bars – fixed costs and operational costs). Conservatively, we let the `all-reserved` configuration achieve a 99% satisfaction ratio whereas both `all-on-demand` and `straddle` achieve 100% coverage. In each graph, we include one additional group of bars for the geometric mean across all Web traces.

From Figure 5(a) and Figure 5(b), we observe that `straddle` uniformly achieves the least cost configuration across all traces and across the different rental granularities. With the 1-hr granularity, the true cost savings of the `straddle` configuration is 21% and 27% over `all-reserved` and `all-on-demand`, respectively. With the 10 minute rental granularity, the cost savings of `straddle` are 17% and 26% respectively. Note, in this case, the 1-hr granularity hurts the base cases (`all-reserved` and `all-on-demand`) more than our design. Consequently, our cost savings are higher with 1-hr granularity than with 10 minute granularity.

6.3 Imperfect statistical models

Our previous results assumed that the statistical models are known. This implies that the load predictor model was trained with the full trace and the frequency, and error distributions were also precisely known. Obtaining precise models is necessary but orthogonal to our contributions (which are about how such workload models may be used to reduce cost). However, we also evaluate the impact of imperfect statistical models in this section. We assume that our predictors and statistical models use data from a part of the trace and evaluate the benefits of our techniques on the remainder of the trace. Figure 6 plots the cost of the fixed margin configuration (normalized to that of the `ShrinkWrap-opt` configuration) for Wikimedia for the `all-on-demand` commitment policy. The X-axis plots various satisfaction ratios from 99% to 99.9%. At 99% the penalty arising from imperfect statistical models eliminate most of the cost savings degrading the difference to nearly 1%. However, at higher satisfaction ratios our techniques show nearly 5% cost reduction in spite of the imperfect models. That is not surprising because of the cost-savings opportunity is higher at higher satisfaction ratios (as we will show in later in Section 6.4.2).

6.4 Other results

While the basic question of cost savings were covered in previous sections, this section focuses on several auxiliary results that serve to enhance the understanding of our techniques. We use 10-minute rental granularities in this section.

6.4.1 *Deconstructing* `ShrinkWrap-opt`

Recall that `ShrinkWrap-opt` uses two independent techniques to reduce margin costs. The basic `ShrinkWrap` mechanism uses table-lookup to obtain per-load-level margins and

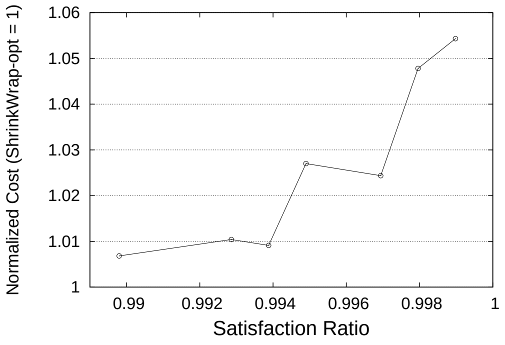


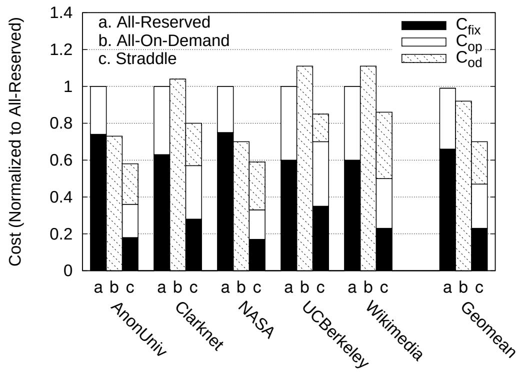
Figure 6: Impact of imperfect statistical models (Wikimedia, `all-on-demand`)

the dynamic programming algorithm selects the least-cost way to expend the tolerance budget. The two techniques can be decoupled by using arbitrary heuristics to decide where to expend the tolerance. Once decided, the original `ShrinkWrap` mechanism can be used to achieve the cost-savings of per-load-level margins.

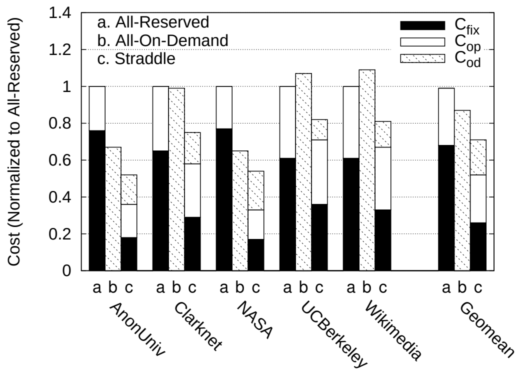
Note, the typical context of using heuristics is because a problem is computationally hard. In our context, the dynamic programming problem is not computationally hard (even though there is a pseudopolynomial term in the complexity equation) for typical problems. Rather, our purpose of examining heuristics is to isolate the relative benefits of the `ShrinkWrap` mechanism from the optimal dynamic programming solution.

We examine three different heuristics: (1) *random*, which randomly selects intervals where the tolerance budget may be spent, (2) *max-swing*, which selects the intervals which represent the largest relative increase in the actual-load-to-predicted-load ratio, and (3) *greedy*, which uses a greedy algorithm to successively choose time intervals which yield the maximum cost savings.

Figure 7 plots the cost increase relative to `ShrinkWrap-opt` (Y-axis) of the three heuristics (curves in figures) as we vary the tolerance (X-axis) for Clarknet (Figure 7(a)) and Wikimedia (Figure 7(b)). We use the `all-on-demand` configuration. As such, the cost on the Y-axis is directly proportional to the machine-intervals used by the heuristics. Consistently, `ShrinkWrap-opt` achieves the lowest cost (i.e., all the curves stay above the 1.0 value). The heuristics suffer significant cost increases when tolerance is high. But at low tolerances, the greedy heuristic has a marginal cost penalty. Intuitively, that makes sense because the greedy choices do have significant cost-savings. Further, with very little tolerance, the total cost impact of making a sub-optimal decision is limited because of an Amdahl's law effect. The fact that even the *random* and *max-swing* heuristics suffer less than 10% cost penalty indicates that at low tolerance, `ShrinkWrap`'s table-lookup based technique contributes more of `ShrinkWrap-opt`'s cost savings than the dynamic programming approach. However, there is no reason to use heuristics instead of the optimal approach because (a) the execution time to run the optimal algorithm is negligible and (b) the algorithm is not run frequently. Finally we note that, although the heuristics get worse at higher tolerance, *greedy* degrades more gracefully than the other two heuristics.



(a) 1-hr rental granularity



(b) 10-minute rental granularity

Figure 5: True costs with commitment straddling

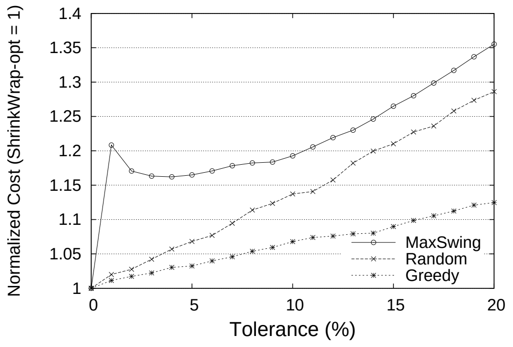
6.4.2 Tolerance vs. Cost tradeoffs

For ease of comparison, all earlier results in this section ensured that the satisfaction ratio was held constant (99%) while comparing costs. In this section, we examine two-way tolerance vs. cost tradeoffs to demonstrate how our techniques push the Pareto frontier in a wider space of parameters. Figure 8 shows the variation in cost (X-axis) and satisfaction ratio (Y-axis) for the Clarknet (Figure 8(a)) and Wikimedia (Figure 8(b)) traces. The costs (X-axis values) are relative costs within each graph and are not comparable across graphs. Each graph includes six point-clouds corresponding to the cross product of the machine-acquisition policies (`all-reserved`, `all-on-demand`, and `straddle`) and the margin mechanisms (`FM` and `ShrinkWrap-opt`). Points closer to the top and left of the graph are better because they are on the Pareto-frontier with respect to the tolerance/cost tradeoff. The point-clouds were obtained by varying parameters such as tolerances, margins, the number of reserved-machines in the straddle configuration.

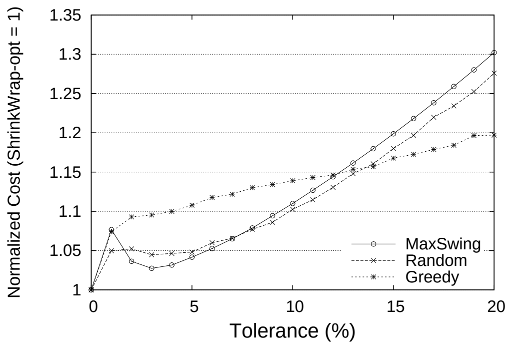
The following interesting trends are visible in the two graphs. First, the benefit of each technique is clear. The `straddle` configurations are always better than the corresponding `all-reserved` and `all-on-demand` configurations while keeping the margin policy the same. Similarly, `ShrinkWrap-opt` helps push the Pareto frontier for each of `all-reserved`, `all-on-demand` and `straddle` configurations.

Second, we observed one phenomenon rather consistently, which we call the *outlier* effect. In comparing the `FM` and `ShrinkWrap-opt` versions we observed that there was a significant gap in costs at high satisfaction ratios. That gap narrowed at around 95-97% satisfaction ratios only to widen again at satisfaction ratios lower than 94%. This can be explained by *outlier* load levels that are observed in a small fraction of points. The use of fixed margins to cover such outliers imposes a high cost at high satisfaction ratios. However, once the outlier effects are trimmed (i.e., at lower-satisfaction ratios) and the margins start coinciding with the dense part of the load, any further reduction in margins causes steep drops of satisfaction ratios, resulting in widening the gap with respect to `ShrinkWrap-opt`. `ShrinkWrap-opt` avoids the cost penalties of the outlier effect by improving satisfaction ratios without the steep increase in cost.

Finally, a comparison of Wikimedia and Clarknet shows that Wikimedia incurs relatively less margin costs. The difference is because Wikimedia has lower short-term variance in loads.



(a) Clarknet



(b) Wikimedia

Figure 7: Impact of heuristics

6.4.3 *Sensitivity studies*

The point-clouds seen earlier in Figure 8 already serve as sensitivity studies for some parameters since they vary various parameters such as reserved-machines in `straddle`, satisfaction ratios, and margins. In addition, we also conducted sensitivity studies varying other parameters such as the relative ratios of reserved-cost to on-demand cost. Because the results are qualitatively similar, we omit detailed results; instead observing that in the region of our operation (varying the ratio from 0.6 to 0.8) the trends were as expected. The gap with respect to `all-on-demand` decreases with increasing c_{rs}/c_{od} ratio because reserved costs approach on-demand costs. The converse was true for the gap with respect to `all-reserved`.

7. RELATED WORK

There has been some work in predicting the demand of enterprise applications [6, 8] by using various techniques including pattern recognition and feedback control theory. We use one such prediction mechanism in our base case. Because perfect load prediction is unlikely, the use of margins to handle prediction error is likely to remain. Also note that our technique to reduce true costs via commitment straddling will remain useful even in the unlikely event of perfect load prediction.

There are a number of techniques that target operational costs (with a focus on server power, cooling power or some combination of the two) in data centers [1, 5, 6, 11, 12, 13, 14]. At a high level, shutting idle servers down for power savings is similar in spirit to shutting down machine instances to save cloud-user cost. Our comparison with the fixed-margin configuration covers such techniques since it is similar to spatial subsetting [5] with the auto-regressive moving-average model based load prediction [6] and a fixed margin similar to SurgeGuard [1]. In addition, there is one key differences of our work with respect to the above body of work. Data center workloads may have different characteristics than an individual Web service because a data center represents an aggregate of several activities. Such aggregation has an effect of smoothing out the high-frequency surges, resulting in slow-changing workloads. No such claim can be made about individual Web services which do see significant load volatility (which results in margin cost).

Recent work by Hajjat *et al.* [9] examines ways to optimize the cost of vmigrating enterprise computation to the cloud while bounding performance degradation. Their model assumes that the cloud offers a costs-performance tradeoff wherein any cloud migration saves costs *and* reduces performance (because most users are assumed to be internal to the organization, and moving to a distant cloud reduces application responsiveness). Their migration strategy does not result in reduced machine utilization. In contrast, our work targets Web servers for which most users are external. Thus moving to the cloud has no performance impact. Further, we examine how to reduce costs and machine utilization (by reducing margins) while preserving performance.

Finally, while our techniques are applicable broadly to IaaS cloud vendors, who offer raw virtual machine based interfaces, there may be some limitations in alternate "platform" cloud models. For example, the Google AppEngine may not expose resource provisioning to the cloud user, thus making it impossible to use margin minimization. However, our techniques will remain useful at the cloud vendor end. For example, if the cloud vendor is committed by means of a service level agreement (SLA) to maintain a certain satisfaction ratio, they may internally use margin minimization to minimize resource usage. Another example could be that of a vendor that does not offer volume discounts for committed buyers may preclude the use of commitment straddling. However, we do not consider that to be a likely situation in view of the basic economic good-sense of volume discounts in all business activity.

8. CONCLUSIONS

Cost remains a significant barrier for adoption of cloud computing for ongoing computing operations (as opppsed to episodic computing demands). Web service operations incur two types of costs when serving variable workloads. They incur margin costs to handle uncertainty of load and also true costs to serve requests. This paper addresses both costs optimally, given statistical properties of the workload.

To address margin costs, we develop `ShrinkWrap-opt`, which combines two key innovations. First, based on our observation that margin requirements differ according to load, `ShrinkWrap` avoids the one-size-fits-all approach to margins and uses load-dependent margins, thus reducing wastage. Second, we develop a dynamic programming algorithm that optimally "spends" its tolerance budget to minimize margin costs. Our algorithm recognizes that cost savings from violating the response time targets is not the same across all intervals.

To address true costs, we exploit the various commitment levels offered by cloud vendors to show that the optimal cost configuration requires *commitment straddling* – deployment of both reserved and on-demand servers. The intuition behind such straddling is that any variable workload running on a collection of servers can be thought of as inducing variable utilization on each of those servers. Because the choice of on-demand vs. reserved instances is determined by a break-even utilization ratio, lower cost can be achieved by deploying all the high-utilization servers on reserved machines and the low-utilization servers on on-demand machines.

While the proof of optimality of both the above techniques are valid only under ideal conditions, the techniques do work well in practical conditions. Simulations using real workload traces and real cloud pricing models (Amazon EC2) reveal that combining the two techniques yields 21% cost savings (on average) compared to the baseline configurations. Specifically, our results show that as much as 14.5% cost reduction is possible for Wikimedia.

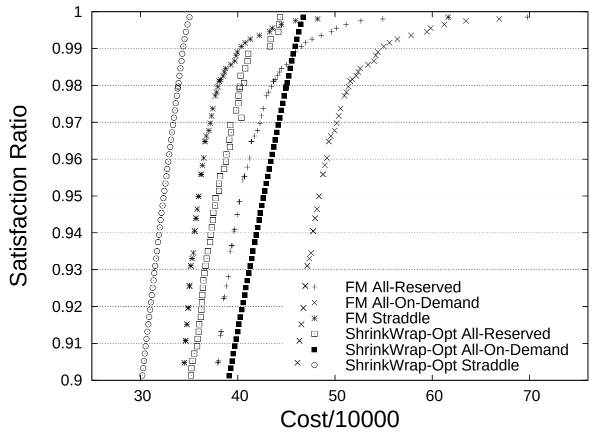
9. REFERENCES

[1] F. Ahmad and T. N. Vijaykumar. Joint optimization of idle and cooling power in data centers while maintaining response time. *SIGPLAN Not.*, 45(3):243–256, 2010.

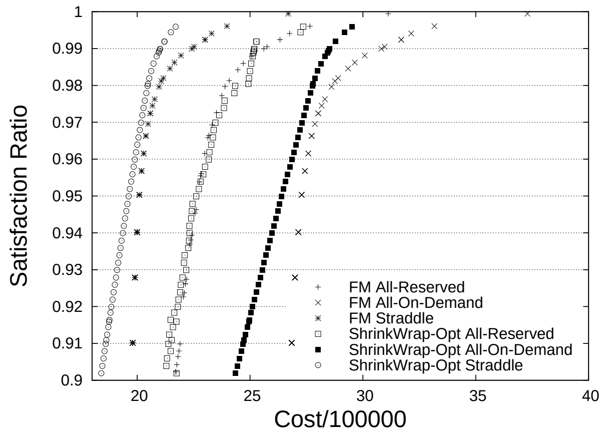
[2] A. O. Allen. *Probability, statistics, and queueing theory with computer science applications.* Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[3] M. Armbrust et al. Above the clouds: A berkeley view of cloud computing. In *Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley*, 2009.

[4] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science*



(a) Clarknet



(b) Wikimedia

Figure 8: Tolerance vs. Cost tradeoff

applications. Wiley-Interscience, New York, NY, USA, 1998.

[5] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116, New York, NY, USA, 2001. ACM.

[6] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. *SIGMETRICS*, 33(1):303–314, 2005.

[7] Engauge digitizer, v4.1. http://digitizer.sourceforge.net/.

[8] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *IISWC '07: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[9] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound:Âăplanning for beneficial migration of enterprise applications to the cloud. In *Applications, Technologies, Architectures, and Protocols for Computer Communication*, volume 40, pages 243–254, 2010.

[10] Traces in the Internet traffic archive. http://ita.ee.lbl.gov/html/traces.html.

[11] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44(3):205–216, 2009.

[12] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[13] R. K. Sharma, C. E. Bash, C. D. Patel, R. J. Friedrich, and J. S. Chase. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Computing*, 9(1):42–49, 2005.

[14] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. *SIGOPS Oper. Syst. Rev.*, 36(SI):239–254, 2002.

[15] Wikimedia statistics. http://meta.wikimedia.org/wiki/Statistics.

APPENDIX

We include a graphical view of the loads for each of the traces we used (see Table 2). Note these are unscaled loads. In our methodology, we scale up the loads to (approximately) Wikimedia-like load levels.

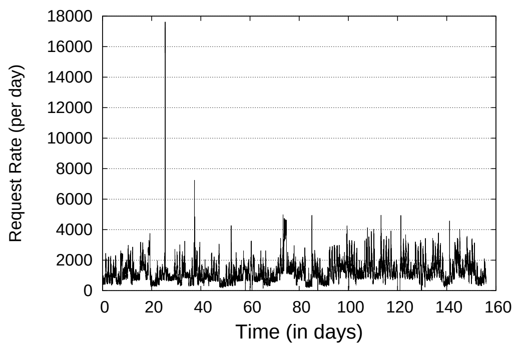


Figure 9: AnonUniv

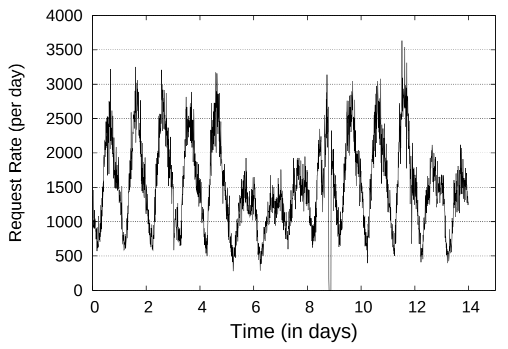


Figure 10: ClarkNet

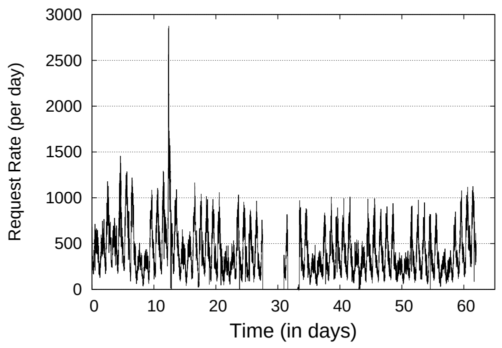


Figure 11: NASA

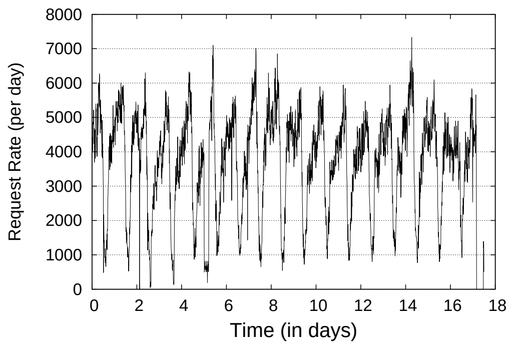


Figure 12: U. C. Berkeley

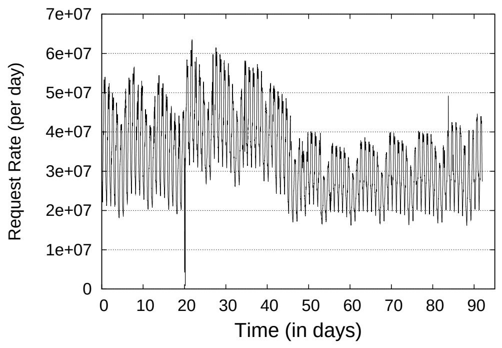


Figure 13: Wikimedia