Constructing Dynamic Policies for Paging Mode Selection

Jason Hiebel
Michigan Technological University
Houghton, Michigan
jshiebel@mtu.edu

Laura E. Brown
Michigan Technological University
Houghton, Michigan
lebrown@mtu.edu

Zhenlin Wang
Michigan Technological University
Houghton, Michigan
zlwang@mtu.edu

ABSTRACT

Virtualization technology is a key component for data center management which allows for multiple users and applications to share a single, physical machine. Modern virtual machine monitors utilize both software and hardware-assisted paging for memory virtualization, however neither paging mode is always preferable. Previous studies have shown that dynamic selection, which at runtime selects paging modes according to relevant performance metrics, can be effective in tailoring memory virtualization to program workload. However, these approaches require low-level manual analysis, or depend on prior knowledge of workload characteristics and phasing.

We map the problem of dynamic paging mode selection to the contextual bandit, a model for sequential decision making in environments with limited feedback. Utilizing random profiling, which executes a workload while regularly selecting paging modes at random, we construct a paging mode selection policy that dynamically optimizes workload performance given page fault and translation lookaside buffer miss counts. Our approach yields an effective policy, DSP-OFFSET, for the dynamic paging mode selection problem. When trained and evaluated on subsets of the SPEC CPU2006 benchmark suite, DSP-OFFSET achieves speedups up to 44% compared to static paging mode selections, which is equivalent to the performance of the state-of-the-art ASP-SVM model. In addition, DSP-OFFSET requires at most a tenth of the profiling time of ASP-SVM (2.5 hours compared to over 24 hours) to achieve equivalent performance.

CCS CONCEPTS

• **Computing methodologies** → *Supervised learning by classification*; *Cost-sensitive learning*; *Sequential decision making*; *Support vector machines*; • **Software and its engineering** → *Virtual machines*; *Virtual memory*;

KEYWORDS

Memory Management, Virtual Memory Paging, Contextual Bandits, Cost-Sensitive Learning, Support Vector Machines

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225082

ACM Reference Format:
Jason Hiebel, Laura E. Brown, and Zhenlin Wang. 2018. Constructing Dynamic Policies for Paging Mode Selection. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3225058.3225082

1 INTRODUCTION

Virtualization is an essential technology for cloud computing, providing an mechanism for performance isolation and resource utilization. A virtual machine monitor, such as Xen [4] or VMWare [30], presents guest operating systems with a virtual abstraction of a physical machine, while providing mappings between the virtual machine resources and actual hardware. The additional layer of abstraction can introduce performance overhead in many ways. For memory virtualization, there are two techniques taken by modern virtual machine managers: Hardware-Assisted Paging (HAP) and Shadow Paging (SP). Whether HAP or SP performs better depends on the memory access characteristics of a workload. Workloads with a large number of page faults will perform better using HAP. Memory intensive workloads will perform better using SP.

Previous work has proposed dynamic methods for selecting between HAP and SP at runtime depending on workload performance characteristics, using manual analysis and a hand-tuned model [31] or expensive enumerative profiling and machine learning [16]. Both cases show that dynamic selection can improve performance by matching, and in some cases beating, the performance of a static paging choice. While effective, both methods require time consuming data collection for model construction as well as manual intervention and/or domain expertise.

In this paper, we present a dynamic selection procedure, DSP-OFFSET, for the dynamic paging mode selection problem. We map the problem of selective paging to the contextual bandit, a model for sequential decision making under limited feedback. With a single, random profiling execution of each benchmark in the SPEC INT2006 suite, using the Binary-Offset algorithm [5], we construct an effective dynamic paging mode selection policy which is competitive with the state-of-the-art ASP-SVM [16] while requiring substantially less profiling time. Unlike previous work, our profiling requires no prior knowledge of workload structure or phasing, and does not require extensive domain expertise or manual tuning. In addition, our dynamic selection framework has the potential to be applied to other system configuration problems.

The paper is organized as follows. In Section 2, we review memory virtualization and summarize work related to dynamic paging mode selection. We also describe the contextual bandit and methods for constructing selection policies from logged random data. Section 3 describes our application of the contextual bandit to the dynamic paging mode selection problem. Section 4 presents our methodology, experimental results, and analysis. Section 5 discusses

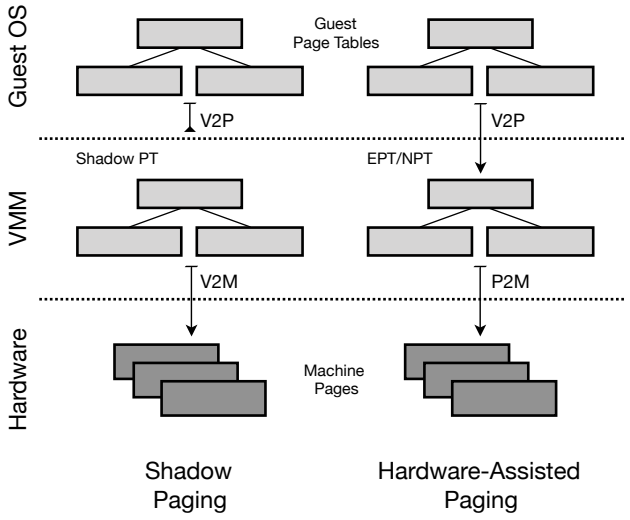

Figure 1: A comparison of Shadow Paging and Hardware-Assisted Paging using extended/nested page tables.

the general applicability of our method as well as possible future research directions. Section 6 summarizes our conclusions.

2 BACKGROUND

We first provide an overview of memory virtualization techniques, and describe prior work for selecting paging modes dynamically at runtime by observing workload characteristics. We then introduce the contextual bandit, which will serve as the underlying model for dynamic paging mode selection. Finally, we describe Binary-Offset and the Weighted Support Vector Machine, which we will use to construct our dynamic selection model.

2.1 Dynamic Paging Mode Selection

In virtualized systems, the virtual machine manager (VMM) is responsible for mapping virtual and physical memory addresses of guest operating systems to hardware addresses. Fully virtualized systems, which do not require modifications to guests, use either Shadow Paging (SP) or Hardware-Assisted Paging (HAP) for address translation. In SP, the VMM maintains a shadow page table in parallel with the page table maintained by the guest. The shadow page table maps virtual addresses in the guest directly to machine addresses (V2M), bypassing the virtual to physical address translation (V2P) of the guest all together. This requires updates to the guest page table to be reflected in the shadow page table, which results in expensive virtual machine (VM) exits and context switches in order to maintain the synchronization between the two tables. In HAP, an extended page table [11] (EPT) or nested page table [6] (NPT) is maintained by the VMM and maps a guest's physical addresses to machine addresses (P2M). An overview of the two methods is given in Figure 1. Page table updates in HAP do not require synchronization and expensive VM exits; however, address translation must access both the guest page table and the extended/nested page table, resulting in more memory accesses and longer latency.

The performance of either paging mode is dependent on workload, and both HAP and SP have cases in which they are preferable [7]. Adams and Agesen [1], Gillespie [11], Wang et al. [31] characterize the advantages of HAP and SP according to workload behavior. Workloads which contain a large number of page faults, and thus a large number of page table updates, will favor HAP, as hardware virtualization does not incur the penalty of page table synchronization. Workloads which are memory intensive will favor SP, as page walk overhead is substantially reduced. This suggests that VM exits, page faults, and translation lookaside buffer (TLB) misses are effective metrics for quantifying workload behavior with regards to memory virtualization.

To address these trade-offs, a number of dynamic paging mode selection schemes have been proposed. These methods choose to utilize hardware or software paging when appropriate based on runtime performance metrics for the current workload. Bae et al. [3] present a heuristic model for Palacios [17] which selects between hardware and software paging at routine intervals according to a pair of dynamic thresholds, for VM exits and for data TLB (DTLB) misses. Wang et al. [31] conducted an extensive manual analysis of page fault and DTLB miss counts for workloads executed using Xen [4], and present a set of hand-crafted and system-dependent thresholds for paging mode selection. However, both of these methods involve subjective construction by domain experts.

Kuang et al. [16] designed a procedure for labeling program phases according the performance gain associated with each paging mode, and utilize machine learning to construct a decision procedure. They enumerate over each phase of a program, comparing the performance of selecting HAP for that phase (and SP for the remaining phases) with the baseline performance of SP; similarly, they enumerate and compare SP with the baseline performance of HAP. This enumerative profiling approach is effective, but requires extensive computation. The authors suggest that the profiling required for the SPEC INT2006 [12] required over 24 hours.

2.2 Contextual Bandits

We will model the dynamic paging mode selection problem as a contextual bandit. The contextual bandit is a method for sequential decision making in environments which provide limited feedback [2, 5, 8, 18, 19, 28] . At each iteration, a contextual bandit observes some contextual information $\vec{x} \in X$ and uses $\vec{x}$ and existing knowledge about the environment in order to select an action $a \in A$. In response to taking action a, the bandit receives a reward r dependent on both the taken action and the associated context; the rewards for actions not taken remain unobserved. This is referred to as bandit feedback. The goal of the bandit is to learn some policy for action selection which maximizes the cumulative reward earned by the learner.

Classic approaches to the contextual bandit are online and dynamically adjust the selection of actions to adapt to both the estimates of each action's reward and the confidence of those reward estimates [2, 19]. These methods are said to balance exploration, selecting an action to improve the estimate of its reward, and exploitation, selecting the action believed to be optimal. However,

Algorithm 1 Binary-Offset [5]

Require: contextual bandit instances S
  $S' = \emptyset$
  for each $(\vec{x}, a, r) \in S$ **do**
    Add to S' the weighted classification instance

$$(\vec{x}, y, W) = (\vec{x}, \text{sign}\,(a \cdot r), |r|).$$

  end for
  return weighted classification instances S'

these methods are generally not amenable to low-level implementation, e.g., in the Xen virtual machine manager, because they require expensive numerical optimization, linear algebra, and statistical procedures.

Alternatively, offline evaluation and construction for contextual bandits can be performed using logged data [5, 8, 18, 28]. Here, exploration and exploitation are not interleaved; rather, exploration occurs for a fixed duration during a training phase, and the resulting logged data is used to construct a policy which is then exploited. The logged data can be obtained by selecting actions uniformly at random, or from carefully constructed deterministic action selections. These methods are also referred to as exploration scavenging [18], as they attempt to utilize the logged data gained from executing some other policy as a form of exploration.

Here, we focus on the Binary-Offset algorithm [5], given in Algorithm 1, which requires binary actions $A = \{-1, +1\}$. Binary-Offset is a method for transforming contextual bandit data $(\vec{x}, a, r)$ obtained from a random policy into weighted data $(\vec{x}, y, W)$, where the class y represents an estimate of the better performing action and the weight W represents the degree to which that action improves from the baseline. For paging mode selection, the context could take the form of relevant performance metrics measured over a sampling period and the action would indicate whether HAP or SP should be selected for the subsequent period. Workload throughput or speedup could both be considered as useful reward metrics.

The resulting weighted classification instances are amenable to a broad suite of machine learning techniques for feature selection, dimension reduction, and classifier construction. Classifiers which directly incorporate instance weights exist in the literature [9, 10, 22, 33]. Alternatively, using the 'Costing' method [34], weighted classification instances can be sampled in proportion to their weight in order to construct a standard, unweighted labeled data set.

We use the Weighted Support Vector Machine (WSVM) [33] to construct a dynamic selection model from the weighted classification instances generated by Binary-Offset. For a set of n weighted instances of the form $(\vec{x}_i, y_i, W_i)$, the (linear) WSVM attempts to find the classifier $f(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b)$ with the largest margin. This can be found using the constrained optimization problem

$$\begin{aligned} \underset{\vec{w}, b, \xi_i}{\arg\min} \quad & \frac{1}{2}\|\vec{w}\|^2 + C \sum_{i=1}^{N} W_i \xi_i \\ \text{subject to} \quad & y_i(\vec{w} \cdot \vec{x} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{1}$$

where C is a hyper-parameter indicating the relative importance of the margin size and the weighted misclassification error. This resulting linear classifier is simple to implement in a virtual machine manager.

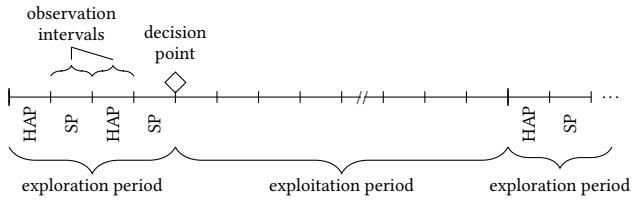

Figure 2: Design and parameters of DSP-SAMPLE.

3 DYNAMIC PAGING MODE SELECTION

We formulate dynamic paging mode selection as a contextual bandit, wherein the virtual machine monitor selects between Hardware-Assisted Paging (HAP) and Shadow Paging (SP) at regular intervals depending on relevant performance metrics in order to optimize workload performance. The contextual information will take the form of page fault and data translation lookaside buffer (DTLB) miss counts, as they characterize the performance of the two paging modes. The action space contains both HAP and SP. The reward will be a measure of workload performance, based on the number of instructions retired per cycle count (IPC) over an observation interval, for the selected paging mode.

Here we present two methods. The first is a simple, context-less bandit model, DSP-SAMPLE, which selects paging modes by comparing the IPC of HAP and SP directly at runtime without taking advantage of page faults, DTLB misses, or any other performance metrics. The second is a contextual bandit model, DSP-OFFSET, which exploits both page fault and DTLB miss counts in order to select paging modes which provide a speedup compared to a random baseline. However, unlike DSP-SAMPLE, DSP-OFFSET requires offline profiling and training.

3.1 Direct Sampling (DSP-SAMPLE)

DSP-SAMPLE is a simple, direct sampling approach which operates in two stages. The first stage alternates between selecting HAP and SP several times in order to discover which paging mode provides the highest IPC. The second stage selects the paging mode which was found to provide the best performance on average and utilizes that paging mode for a time. The two stages alternate, timed appropriately to balance constructing a confident estimate of performance, utilizing the best identified paging mode, and adapting to changing workload characteristics. This can be described as a method which balances exploration (sampling the performance of each paging mode), and exploitation (utilizing the best performing paging mode) — similar to the contextual bandit, but without contextual information. A similar model is used in Jiménez et al. [14] for dynamic hardware prefetcher configuration.

The design of DSP-SAMPLE is summarized in Figure 2. This method is parameterized by the length of the observation interval, as well as the number of intervals in both the exploration and exploitation periods. A longer exploration period can provide a better estimate of performance, which can lead to fewer poor exploitation period selections. However, a longer exploration period will

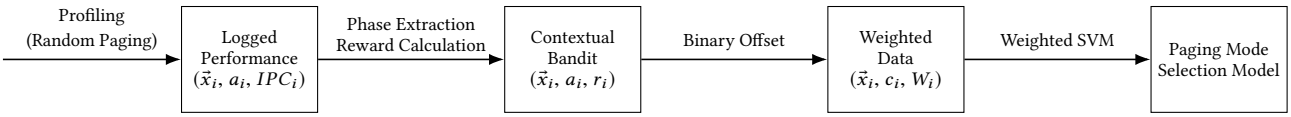

Figure 3: Workflow for constructing DSP-OFFSET models using random logged data.

also incur more overhead from paging mode switching. A longer exploitation period will reduce the frequency of exploration, but shifting workload characteristics can cause the selected paging mode to no longer be desirable. Parameter tuning is required for DSP-SAMPLE to be effective.

3.2 Contextual Bandit Model (DSP-OFFSET)

To construct the DSP-OFFSET model, we first must obtain logged data from random paging mode selections for workloads of interest. Next, the logged data must be converted into a form which is usable to Binary-Offset. This includes identifying phasing structure and defining a useful reward function. Finally, we transform, via Binary-Offset, the logged data into weighted data and use the WSVM in order to construct the DSP-OFFSET model. This construction is illustrated in Figure 3.

As with previous work [1, 16, 31], we rely on page faults and DTLB miss counts to characterize the relative performance of HAP and SP. The frequency of DTLB misses is correlated with the frequency of page walks, and the frequency of page faults is correlated with page table updates; therefore, we expect HAP to outperform SP during periods of frequent page faults and SP to outperform HAP during periods of frequent DTLB misses. However, effective switching requires determining the trade-off for workloads with mixed characteristics. As page faults and DTLB misses characterize the relative performance of HAP and SP, we assume that the relative performance of the two paging modes otherwise remains unchanged by other, unobserved performance characteristics, as well as from the historical behavior of both page faults and DTLB misses. As we find that the distribution of both page fault and DTLB miss counts over fixed sampling intervals are heavy tailed, we consider the binary logarithm of both counts instead of using the counts directly. This has the effect of leveling out the distribution of each metric and reducing the effect of outlier behavior.

Training data is obtained from workloads by executing a random paging mode policy. At regular sampling intervals, Xen measures relevant performance metrics, including page faults, DTLB misses, and IPC, and selects HAP or SP uniformly at random for use during the next sampling interval. If the system is already using the selected paging mode, no change happens. Otherwise, the system switches to the new paging mode, incurring the associated cost. We associate the performance characteristics used to make a selection (page fault and DTLB miss counts over the interval which just ended) with the performance resulting from that selection (IPC of the following interval).

The logged training data must now be transformed into contextual bandit data, i.e., context, action, and reward. We consider the speedup of a paging mode selection compared to average workload performance as a reward. However, many applications exhibit phasing behavior [24–26]. Shifting performance characteristics, either between workloads or between phases of a workload, can skew the weighting of our instances toward certain phases. Both `milc` and `xalancbmk` from the SPEC CPU2006 [12] suite skew our results if we consider IPC as a reward directly, as both contain small phases of high IPC that would be more strongly weighted despite providing little opportunity for improved performance. Any possible imbalance between HAP and SP due to random sampling during these phases can amplify the effect.

To account for these extraneous effects, we consider phases of the logged performance data. Using the change-point detection algorithm PELT [15], we partition each random profiling execution into a set of phases based on the sequence of IPC values. PELT is an efficient dynamic programming algorithm for identifying changes in the distribution of a time series, such as identifying changes to the mean and variance of a workload's IPC over time. PELT optimizes the number and position of the change points given an information criterion penalty. Given a set of change-points c_j, we segment our training data into phases $\left[c_j, c_{j+1}\right]$. These phases simply represent periods of consistent workload performance. An alternative would be to specify these phases manually, however we find that PELT is sufficient for identifying meaningful periods and does not rely on domain expertise.

The reward is calculated based on the logarithmic speedup of instance performance (IPC) against the average performance of the intervals (phase). Instances which cause a speedup in comparison to random are given a positive reward. Instances with no speedup or slowdown compared to the average performance of the random selections have a zero reward as they represent the baseline behavior. For Binary-Offset, instances which cause a slowdown compared to random should be treated as instances of the opposite paging mode with the reciprocal speedup. Therefore, for a instance $i \in \left[c_j, c_{j+1}\right]$, we calculate the reward as

$$r_i = \log \frac{IPC_i}{\overline{IPC}_{[c_j, c_{j+1}]}}, \tag{2}$$

where $\overline{IPC}_{[c_j, c_{j+1}]}$ is the average IPC for the phase containing instance i. Measuring speedup (per phase) avoids the problem of high IPC phases having a stronger weighting, as the weighting is now relative to the average performance of the phase. Figure 4 (top and middle) illustrates the transformation from IPC to reward.

Using the contextual information $\vec{x}_i$ (page fault and DTLB miss counts), actions a_i as HAP and SP (mapped to -1 and +1 respectively), and rewards r_i calculated according to Equation 2, we transform the contextual bandit data $(\vec{x}_i, a_i, r_i)$ into weighted data $(\vec{x}, y_i, W_i)$ using Algorithm 1. This transformation is illustrated in Figure 4 (middle and bottom). The weighted data describes, for some set of performance metrics $\vec{x}_i$, which paging mode y_i is expected to provide a speedup and how strongly it is expected, i.e., the weight W_i. We apply a linear WSVM (Equation 1) to the weighted instance

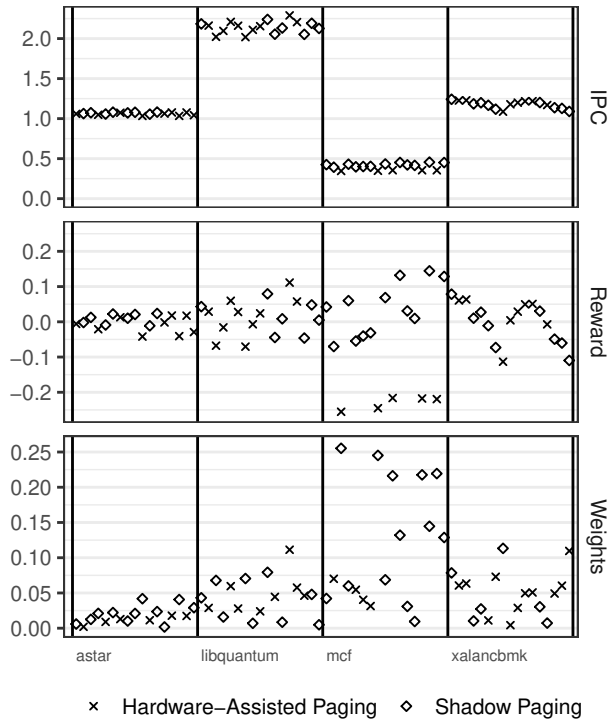

Figure 4: Top; traces of IPC and paging mode using a random selection policy for a subset of select workloads. Middle; IPC transformed to reward. Bottom; Binary-Offset transformation to weights.

data in order to construct a linear decision function which maps page fault and DTLB miss measurements to a paging mode selection. Other algorithms (e.g., weighted logistic regression, weighted sampling [34]) were considered but WSVM provided the best performance.

To prevent rapid switching between HAP and SP, a potential source of performance loss due to the switching overhead, we define a margin around the decision function. Any workload which is operating inside of the margin does not trigger a switch, as we assume that the potential performance advantage will not outweigh the cost of switching. We find that a quarter of the WSVM margin results in good performance:

$\vec{w} \cdot \vec{x} + b > +0.25$: if necessary, switch to SP,
$\vec{w} \cdot \vec{x} + b < -0.25$: if necessary, switch to HAP.

Thrashing behavior which occurs because a workload alternates between two extremes, and thus alternates outside of the margin, would not be prevented. However, this does not happen in practice for the workloads we investigated.

4 EVALUATION

This section describes our experimental methodology and presents our results. We evaluate the performance of both DSP-SAMPLE and DSP-OFFSET, and compare both models against the state-of-the-art ASP-SVM [16]. To conclude, we discuss the advantages of DSP-OFFSET with respect to profiling cost (Section 4.4).

Table 1: Hardware Configuration

CPU		2.8	GHz	
Memory		4	GB	
Cache	L1	64	KB	4-way
	L2	512	KB	8-way
	L3	8192	KB	16-way
DTLB	L1	64	entries	4-way
	L2	512	entries	4-way

4.1 Experimental Environment

Experiments are conducted on a 1st generation Intel Core i5 processor (Nehalem microarchitecture), running at 2.8 GHz, with Intel Turbo Boost and other adaptive clock cycle technology disabled. The hardware configuration is summarized in Table 1. A 64-bit host OS running Linux 2.6.18 (CentOS 5.4) is configured to run a modified version of Xen 3.3.1 which implements the paging mode selection mechanism for the Xen hypervisor as described in [31]. A 32-bit guest OS, also running Linux 2.6.18 (CentOS 5.4), is provided with 3 GB of memory and is constrained to a single core, for which it has sole affinity. Policies are evaluated using the SPEC CPU2006 [12] benchmark suite as the benchmarks show a variety of memory behavior. The benchmarks are compiled for the guest OS using GCC 4.1.

At regular intervals, Xen measures relevant performance metrics, including page faults and DTLB misses, and identifies if the system should utilize HAP or SP for the following interval according to the current policy. To measure page faults, a kernel module in the guest OS notifies the Xen hypervisor of a shared memory address in which the guest OS records the page fault count. To measure DTLB misses, instructions retired, and clock cycles, the Xen hypervisor configures and accesses the Performance Monitoring Unit [13] directly. A programmable counter is configured to measure DTLB misses (`DTLB_MISSES.WALK_COMPLETED`) and IPC is measured using the fixed-function counters for retired instructions and core clock cycles. The logarithmic page fault and DTLB miss counts are calculated using a simple fixed-point arithmetic binary logarithm; alternatively, these features could be approximated by identifying the number of leading zeros in the counts.

4.2 Experimental Design

We evaluate DSP-SAMPLE with a sampling rate (observation interval length) of 100 ms. For the exploration period, the algorithm measures the IPC of HAP and SP three times each (for a total of 0.6 s), and then selects the better performing paging mode to exploit for 50 observation intervals (for a total of 5 s). This is approximately a 1:10 exploration to exploitation ratio. We also attempted other possible parameter settings, but found no particular setting which was effective in all cases.

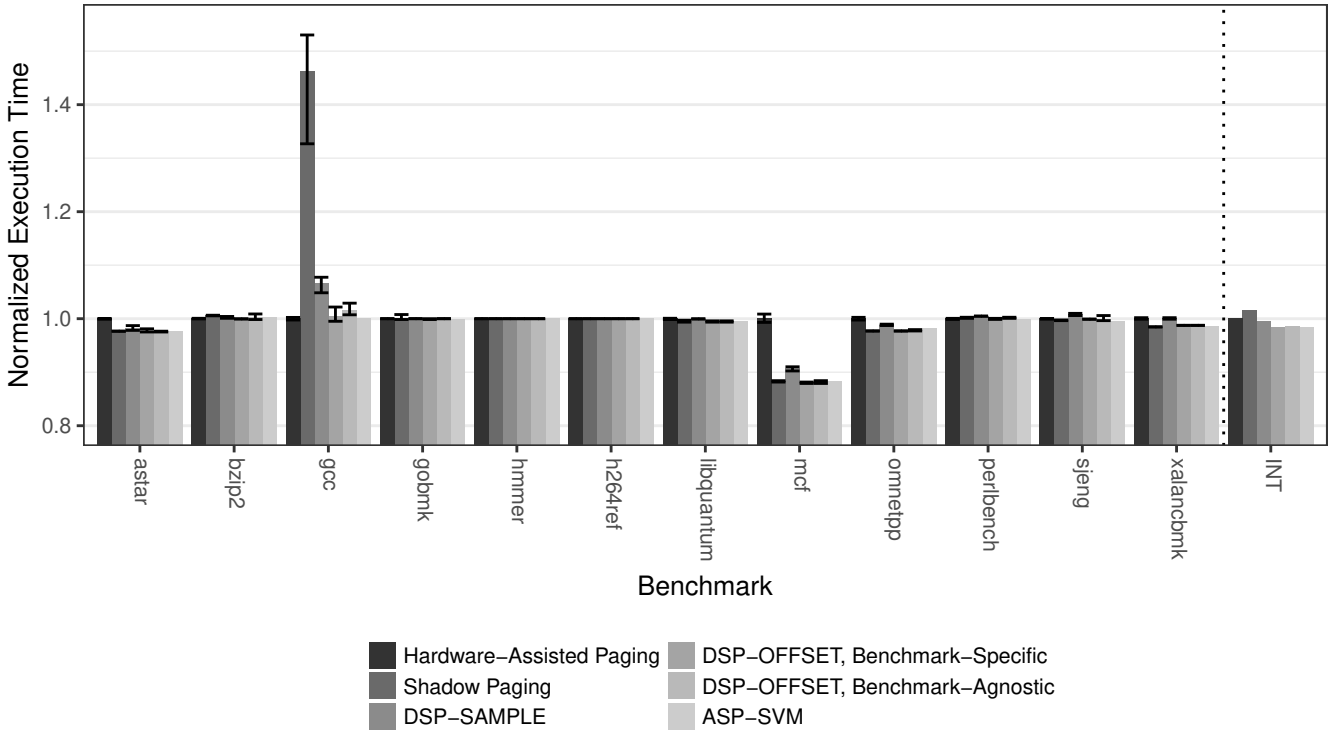

Figure 5: Mean normalized execution times for Hardware-Assisted Paging, Shadow Paging, DSP-SAMPLE, DSP-OFFSET (Benchmark-Specific), DSP-OFFSET (Benchmark-Agnostic), and ASP-SVM [16] on SPEC INT2006. Error bars indicate minimum and maximum normalized times.

We evaluate DSP-OFFSET for both a benchmark-specific and benchmark-agnostic setting. In the benchmark-specific case, we train a DSP-OFFSET model for each benchmark using a single random profiling execution from that benchmark. Each model is then evaluated on the benchmark for which it was trained. This evaluates the performance of DSP-OFFSET when constructed on a wide range of training data sizes with varying workload characteristics. In the benchmark-agnostic case, we construct a single DSP-OFFSET model by aggregating data from the SPEC INT2006 benchmarks. This evaluates the effectiveness of DSP-OFFSET to model a broad range of workload characteristics and to generalize to other workloads not included as part of the training data. In both cases, we use a sampling period of 1 s for both random profiling and evaluation, and we select the hyper-parameter C for the WSVM (Equation 1) using a simple grid search. We considered sampling periods of 2 s, 1 s, and 100 ms and found that the differences in the resulting policies and performance were small.

4.3 Results

Figures 5 and 6 summarize the mean execution times of HAP, SP, DSP-SAMPLE, DSP-OFFSET, and ASP-SVM, normalized to the mean execution time of HAP, for the SPEC INT2006 and FP2006 benchmark suites. For HAP, SP, DSP-SAMPLE, and DSP-OFFSET, we report the min, mean, and max ratios of three runs. For ASP-SVM, we report the mean of five runs. The results for `povray` are omitted for ASP-SVM, as they were not reported in [16].

We call out specific benchmarks where there is a notable difference between HAP and SP: `gcc` and `tonto` favor HAP (46%, 21% loss with SP, respectively); `cactusADM` and `mcf` favor SP (13%, 12% gain). On average, SP presents a performance loss of 1.6% compared to HAP (1.6% for SPEC INT2006 and 1.5% for FP2006), and many benchmarks show no difference in performance between the two static policies.

4.3.1 Direct Sampling. DSP-SAMPLE presents an overall performance loss of 0.2% compared to HAP (0.5% gain for SPEC INT2006 and 0.7% loss for PF2006). While the performance of DSP-SAMPLE can be similar to the performance of the best static policy, as is the case for `gcc`, `tonto`, `cactusADM`, and `mcf`, there are some cases for which the performance of the dynamic procedure is no better than the worst static policy. For `bwaves`, `milc`, and `wrf`, DSP-SAMPLE has roughly an equivalent average performance loss to SP (3.5%, 7.3%, 3.8% loss, respectively) and for `milc` and `wrf` there is significant variability in the performance across multiple runs. The performance of DSP-SAMPLE may be tailored, through careful parameter selection, to better suit certain types of workloads. However, this can in turn negatively affect other workloads.

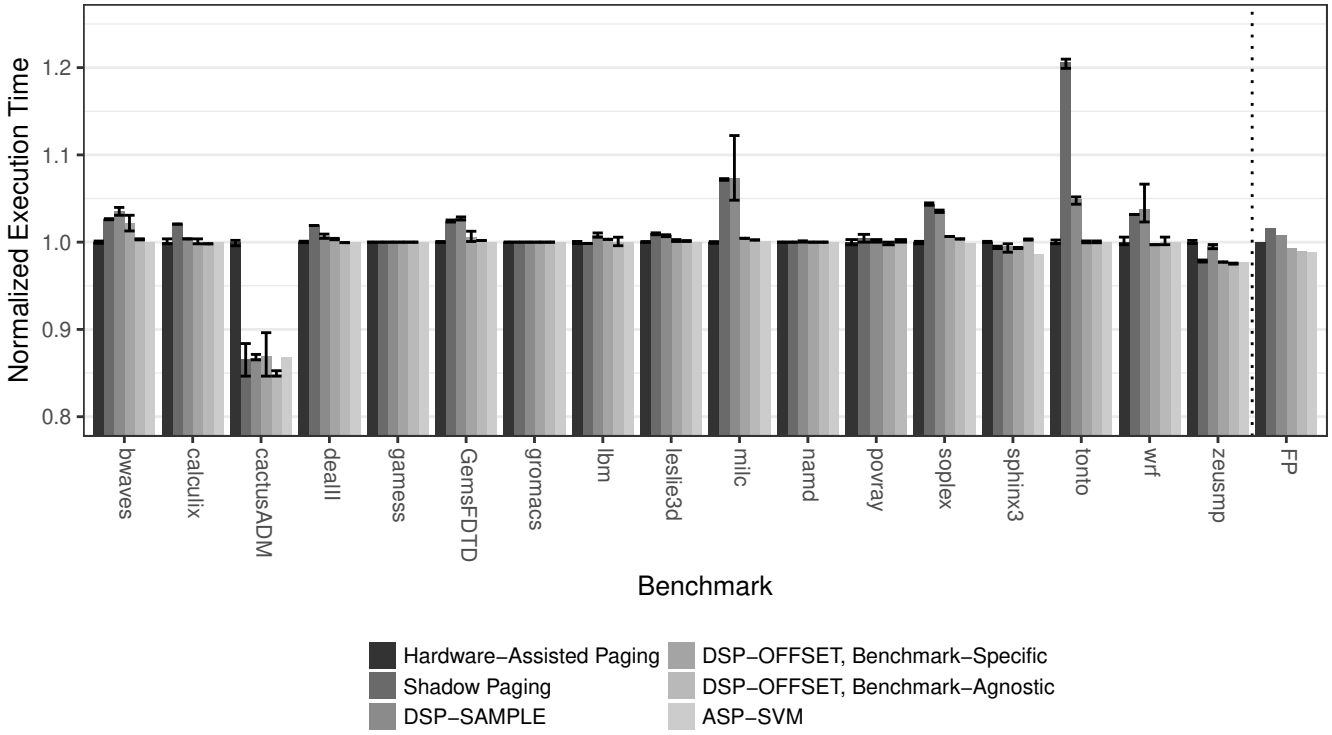

Figure 6: Mean normalized execution times for Hardware-Assisted Paging, Shadow Paging, DSP-SAMPLE, DSP-OFFSET (Benchmark-Specific), DSP-OFFSET (Benchmark-Agnostic), and ASP-SVM [16] on SPEC FP2006. Error bars indicate minimum and maximum normalized times.

4.3.2 Benchmark-Specific Models. As DSP-OFFSET utilizes the contextual information (performance metrics) available, we anticipate that each benchmark-specific model should provide effective performance on the workload in which it was trained. For nearly all benchmarks, the performance of the benchmark-specific DSP-OFFSET model constructed from a single random profiling execution matches the performance of the best static policy. The notable exception to the favorable performance of DSP-OFFSET is `bwaves`, which performs 2.1% worse than HAP but on average better than SP. On average, the DSP-OFFSET models present a 1.1% performance gain (1.6% for SPEC INT2006 and 0.8% for SPEC FP2006).

4.3.3 Benchmark-Agnostic Model. Whereas in the benchmark-specific we constructed separate models for each benchmark, here we construct a single benchmark-agnostic model for the full suite. The benchmark-agnostic DSP-OFFSET model presents a 1.2% performance gain compared to HAP (1.4% for SPEC INT2006 and 1.0% for FP2006). In comparison, ASP-SVM presents a 1.3% performance gain (1.6% for SPEC INT2006 and 1.1% for FP2006). Overall, both DSP-OFFSET and ASP-SVM have similar performance gains over the static policies. Again, we stress that the aggregate data used to train DSP-OFFSET contains only a single random execution for each integer benchmark.

The paging mode selections for the benchmark-agnostic DSP-OFFSET model are summarized in Figure 7, including periods in which the model would have triggered a switch but did not due to the margin. Workloads for most benchmarks cause a single paging mode to be selected almost always during the course of the benchmark's execution. For benchmarks which execute primarily in SP, we observed periods at the beginning and end of the profiling run in which HAP was utilized. These periods coincide with the initialization and tear-down of the SPEC tools as well as with the start and end of program execution. A larger than average number of page faults are to be expected during these periods, and thus these periods would favor HAP as hardware paging avoids the cost of page table synchronization. The margin only affects `bwaves` and `zeusmp`. For `zeusmp`, the margin prevents thrashing behavior that would otherwise cause the model to switch between HAP and SP every two or three seconds. For `bwaves`, we observe that the benchmark's workload is predominantly inside of the margin.

4.4 Profiling Cost

Collection of the training data using random selection is no more expensive than running the benchmarks using the worst of their static paging modes. Moreover, a single random evaluation for each benchmark is sufficient to obtain performance equivalent to ASP-SVM. In contrast, ASP-SVM requires an average of 6 executions of each benchmark in the collection of the training data. The reported data collection time for ASP-SVM was over 24 hours; in comparison, random profiling for SPEC INT2006 requires less than 2.5 hours for

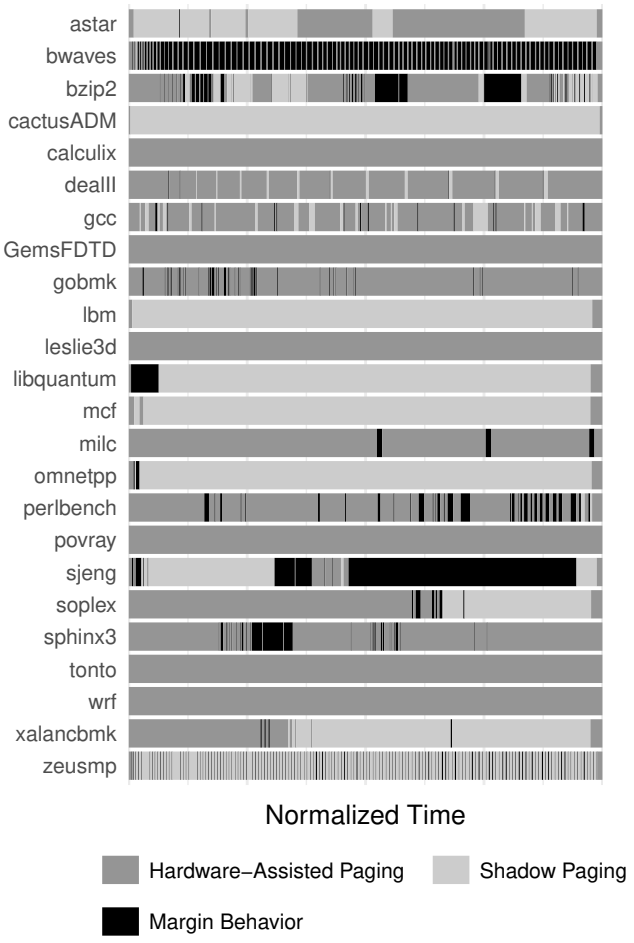

Figure 7: Paging modes selected over time for SPEC CPU06 benchmarks using the benchmark-agnostic DSP-OFFSET constructed on SPEC INT06.

DSP-OFFSET, and the full SPEC CPU2006 suite requires less than 6.5 hours.

While our profiling time is reduced in comparison to ASP-SVM, the dataset for DSP-OFFSET is several orders of magnitude larger. For DSP-OFFSET, with a 1 s sampling period, there are approximately 25000 data samples across the twelve integer benchmark executions (one data sample per sampling period); for ASP-SVM there are 60–67 samples. This is noisy data, both due to variable workload characteristics as well as the random selection of paging modes. There are periods of a benchmark's execution which will be under-sampled. In some cases, random selection may also lead to periods where one paging mode is sampled almost always. This leads to outliers in the contextual measurements (page fault and DTLB miss counts) as well as in the labels and weights we eventually generate using Binary-Offset. The enumerative profiling approach taken in [16] encodes knowledge and assumptions regarding workload structure, which is a significant source of their profiling cost. We instead compute this structure after the fact using the random logged data.

5 DISCUSSION AND FUTURE DIRECTIONS

While we chose to apply our method specifically to paging mode selection, the framework we present is generally applicable to a range of dynamic configuration problems for computer systems. One particular example is that of hardware prefetching. Modern Intel systems are equipped with four hardware prefetchers which can be enabled or disabled at runtime [29]. IBM POWER7 systems are equipped with a highly configurable prefetch engine that allows prefetchers to be parameterized (e.g., prefetching depth and stride) [27]. Liao et al. [20], Rahman et al. [23] propose prefetcher configuration recommendation methods; however, these are static, and not dynamic approaches. A single, fixed configuration is selected for a given program after a window of profiling. Jiménez et al. [14] propose a direct sampling method, similar to the DSP-SAMPLE approach given in Section 3.1, without using contextual information to guide their selection.

Paging mode selection can be seen as a small and well understood instance of a dynamic configuration problem. Performance can be described by a small number of features identified by domain knowledge (page faults, DTLB misses), with only two configurations (Hardware-Assisted Paging, Shadow Paging). Hardware prefetching is an interesting application as it presents the challenge of larger action sets (16 in total for Intel systems) and action sets which are combinatorial in nature (4 independent hardware prefetchers). The Binary-Offset method can be expanded into an Offset-Tree [5], providing for larger action spaces. Hardware prefetching can also present the opportunity to expand the contextual information used to include additional performance metrics (our framework has no explicit limit on the number of attributes).

While our application of Binary-Offset substantially reduces profiling time for training, validation of the resulting dynamic selection procedures still requires execution of the model in situ. Methods for evaluating deterministic policies, using random or deterministic data, are available for the contextual bandit [8, 18], and may be amenable to the problem setting. We hope to apply these methods in order to provide offline evaluation, in addition to offline model construction.

Finally, we note that the application of Binary-Offset still required careful attention in order to address problems such as label noise. Standard convex-loss methods are sensitive to label outliers in the data [21, 32]. We hope to investigate the use of more robust machine learning methods which are capable of addressing this problem.

6 CONCLUSIONS

In this paper, we present DSP-OFFSET, an effective procedure for dynamic paging mode selection which utilizes a simple, random profiling method. Dynamic paging mode selection policies are capable of balancing the trade-off between Hardware-Assisted Paging and Shadow Paging at runtime by dynamically switching the paging mode at runtime according to performance metrics. We evaluate our approach on the SPEC CPU2006 benchmark suite and compare our approach with an existing machine learning method. DSP-OFFSET

achieves speedups up to 44% compared to static paging mode selections and matches state-of-the-art performance. In addition, our method requires substantially less profiling, an 90% reduction in profiling time.

ACKNOWLEDGMENTS

This research is supported in part by the National Science Foundation under Grant No. CSR1422342 and CSR1618384, the National Science Foundation of China under Grant No. 61232008, 61472008, 61672053 and U1611461, Shenzhen Key Research Project under Grant No. JCYJ20170412150946024, and the 863 Program of China under Grant No. 2015AA015305. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Keith Adams and Ole Agesen. 2006. A Comparison of Software and Hardware Techniques for x86 Virtualization. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*. 2–13.

[2] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. 2003. The Nonstochastic Multiarmed Bandit Problem. *SIAM J. Comput.* 32, 1 (2003), 48–77.

[3] Chang S. Bae, John R. Lange, and Peter A. Dinda. 2011. Enhancing Virtualized Application Performance Through Dynamic Adaptive Paging Mode Selection. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11)*. 255–264.

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. *SIGOPS Operating Systems Review* 37, 5 (2003), 164–177.

[5] Alina Beygelzimer and John Langford. 2009. The Offset Tree for Learning with Partial Labels. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. 129–138.

[6] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. 2008. Accelerating Two-Dimensional Page Walks for Virtualized Systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII)*. 26–35.

[7] Nikhil Bhatia. 2009. *Performance Evaluation of Intel EPT Hardware Assist.* Technical Report. VMWare.

[8] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML '11)*. 1097–1104.

[9] Charles Elkan. 1997. *Boosting And Naive Bayesian Learning*. Technical Report. University of California, San Diego.

[10] Yoav Freund and Robert E. Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139.

[11] Matthew Gillespie. 2009. *Best Practices for Paravirtualization Enhancements from Intel Virtualization Technology: EPT and VT-d.* Technical Report. Intel.

[12] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Computer Architecture News* 34, 4 (September 2006), 1–17.

[13] Intel 2016. *Intel 64 and IA-32 Architectures Developer's Manual: Volume 3C.* Intel.

[14] Víctor Jiménez, Roberto Gioiosa, Francisco J. Cazorla, Alper Buyuktosunoglu, Pradip Bose, and Francis P. O'Connell. 2012. Making Data Prefetch Smarter: Adaptive Prefetching on POWER7. In *21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 137–146.

[15] Rebecca Killick, Paul Fearnhead, and I.A. Eckley. 2012. Optimal Detection of Changepoints With a Linear Computational Cost. 107 (2012), 1590–1598.

[16] Wei Kuang, Laura E. Brown, and Zhenlin Wang. 2015. Selective switching mechanism in virtual machines via support vector machines and transfer learning. *Machine Learning* 101, 1 (2015), 137–161.

[17] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell. 2010. Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS '10)*. 1–12.

[18] John Langford, Alexander Strehl, and Jennifer Wortman. 2008. Exploration Scavenging. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. 528–535.

[19] John Langford and Tong Zhang. 2007. The Epoch-Greedy Algorithm for Contextual Multi-armed Bandits. In *Advances in Neural Information Processing Systems 20 (NIPS)*. 817–824.

[20] Shih-wei Liao, Tzu-Han Hung, Donald Nguyen, Chinyen Chou, Chiaheng Tu, and Hucheng Zhou. 2009. Machine Learning-based Prefetch Optimization for Data Center Applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. 1–10.

[21] Tan T. Nguyen and Scott Sanner. 2013. Algorithms for Direct 0-1 Loss Optimization in Binary Classification. In *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML '13)*. 1085–1093.

[22] J. R. Quinlan. 1996. Bagging, Boosting, and C4.S. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96)*. 725–730.

[23] Saami Rahman, Martin Burtscher, Ziliang Zong, and Apan Qasem. 2015. Maximizing Hardware Prefetch Effectiveness with Machine Learning. In *Proceedings of the 17th International Conference on High Performance Computing and Communications*. 383–389.

[24] Xipeng Shen, Yutao Zhong, and Chen Ding. 2004. Locality Phase Prediction. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI)*. 165–176.

[25] Timothy Sherwood, Erez Perelman, and Brad Calder. 2001. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT '01)*. 3–14.

[26] Timothy Sherwood, Suleyman Sair, and Brad Calder. 2003. Phase Tracking and Prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA '03)*. 336–349.

[27] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams. 2011. IBM POWER7 Multicore Server Processor. *IBM Journal of Research and Development* (2011), 191–219.

[28] Alexander L. Strehl, John Langford, Lihong Li, and Sham M. Kakade. 2010. Learning from Logged Implicit Exploration Data. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems (NIPS '10)*. 2217–2225.

[29] Vish Viswanathan. 2014. *Disclosure of H/W Prefetcher Control on some Intel Processors.* Technical Report. Intel.

[30] Carl A. Waldspurger. 2002. Memory Resource Management in VMware ESX Server. *SIGOPS Operating Systems Review* 36, SI (2002), 181–194.

[31] Xiaolin Wang, Jiarui Zang, Zhenlin Wang, Yingwei Luo, and Xiaoming Li. 2011. Selective Hardware/Software Memory Virtualization. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*. 217–226.

[32] Min Yang, Linli Xu, Martha White, Dale Schuurmans, and Yao liang Yu. 2010. Relaxed Clipping: A Global Training Method for Robust Regression and Classification. In *Advances in Neural Information Processing Systems 23 (NIPS)*. 2532–2540.

[33] Xulei Yang, Qing Song, and Aize Cao. 2005. Weighted Support Vector Machine for Data Classification. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks*, Vol. 2. 859–864.

[34] Bianca Zadrozny, John Langford, and Naoki Abe. 2003. Cost-Sensitive Learning by Cost-Proportionate Example Weighting. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM '03)*. 435–442.