Selective switching mechanism in virtual machines via support vector machines and transfer learning

Wei Kuang · Laura E. Brown · Zhenlin Wang

Received: 31 December 2013 / Accepted: 17 May 2014 / Published online: 4 July 2014
© The Author(s) 2014

Abstract Virtualization is an essential technology in data centers allowing for a single machine to be used for multiple applications or users. With memory virtualization, two approaches, shadow paging (SP) and hardware-assisted paging (HAP), are taken by modern virtual machine memory managers. Neither memory mode is always preferred; previous studies have proposed to exploit the advantages of both modes by dynamically switching between these two paging modes based on the on-the-fly system behavior. However, the existing scheme makes the switching decision based on manual rules summarized for a specific architecture. This paper employs a machine learning approach that learns a decision model automatically and thus can adapt to different systems. Experimental results show that the performance of our switching mechanism can match or outperform either SP or HAP alone. Also, the results demonstrate that a machine learning-based decision model can match the performance of the hand-tuned model. Moreover, we further show that different hardware/software settings can affect on-the-fly system behavior and thus demand different decision models. Our scheme yields two effective decision models on two different machines. Additionally, transfer learning was used in order to efficiently train a model when faced with a new hardware configuration with only a limited number of training samples from the new machine.

Keywords Memory virtualization · Support vector machines · Transfer learning

Editors: Vadim Strijov, Richard Weber, Gerhard-Wilhelm Weber, and Süreyya Ozogur Akyüz.

W. Kuang (✉) · L. E. Brown · Z. Wang
Department of Computer Science, Michigan Technological University, Houghton, MI, USA
e-mail: wkuang@mtu.edu

L. E. Brown
e-mail: lebrown@mtu.edu

Z. Wang
e-mail: zlwang@mtu.edu

1 Introduction

Virtualization has become a key supporting technology for data centers in cloud computing with its advantages in server consolidation, resource utilization and performance isolation. A virtual machine monitor (VMM), such as XEN ([Barham et al. 2003](#)) and VMWare ([Waldspurger 2002](#)), manages the actual hardware while providing the guest operating systems with an illusion of a physical machine, called virtual machine (VM). This additional layer of abstraction often introduces performance overhead. One area causing the introduction of significant overhead is memory virtualization. Two approaches, shadow paging (SP) and hardware assistant paging (HAP), are taken by modern VMMs for memory virtualization.

It is easy to observe that an application can perform better in HAP mode than SP mode or vice versa, depending upon its memory access behavior, specifically, the frequency of page table updates and the page walks that result from translation lookaside buffer (TLB) misses. Previous work proposes selective paging which dynamically switches between HAP and SP based on application behavior ([Wang et al. 2011](#)). However, the decision model is hand-tuned and only works for the specific architecture it is designed for.

In a data center environment, there are thousands of machines with a variety of hardware and software configurations and myriad applications tasked to run in VMs. It is prohibitive to collect training data and work out a decision model manually. We thus propose a machine learning-based approach which can learn a decision model automatically. We validate our approach by training and learning a decision model on two different architectures. The results show that dynamic switching based on the automatically learned model can achieve the performance of the hand-tuned model.

These models are effective when applied to new applications on the same physical machine (source domain) but results are poor when applied to a new hardware configuration (target domain). Individual models could be created for each hardware configuration, however, this is time consuming with the collection of training data repeated for each machine. We use transfer learning to expedite the learning process. The results show using knowledge (data samples) from the source domain (Intel machine) and only limited training data collected from the target domain (AMD machine) results in a more accurate model with fewer target data samples needed for training.

The rest of the paper is organized as follows. In Sect. 2 we briefly review virtualization, specifically memory virtualization, and discuss related research on the improvement of HAP mode as well as SP modes. We also give a brief introduction to support vector machines (SVMs), the machine learning method used in the selective switching models, and transfer learning. Section 3 describes our methodology to train a SVM-based decision model. Section 4 presents experiment results and analyses. Section 5 summarizes the paper and discusses possible future research directions.

2 Background and related work

2.1 Virtualized machines

A VMM or hypervisor is a layer of software installed on a physical machine or a host system, the native architecture/system, that provides an illusion of a physical machine by abstracting a VM for each guest operating system (OS). A VM is transparent to the guest OS and end users who think they are interacting with a physical machine. The concept of virtualized systems dates back to the 1960s and has been revived with the popularity of cloud

computing. When running a VMM as an additional layer coordinating the communication between the guest operating system and physical hardware and managing system resources, overheads are introduced, with a significant overhead stemming from memory virtualization.

In most of modern operating systems, virtual memory is supported so that the application running above it perceives a contiguous, private memory space. A memory access in a user application uses a virtual address which is translated to a physical address by the hardware and operating system. The modern architecture, such as Intel, relies on the hardware memory management unit to perform this translation which walks through the page tables (page walk) managed by the operating system. Modern architectures also put in one or two levels of hardware TLB to cache recent address translations. A TLB hit will yield a physical address directly without expensive page walk.

In a virtualized system, the memory is also virtualized and thus the physical addresses are no longer machine addresses. Now, there are three address spaces in a virtualized machine, virtual addresses, guest physical addresses and machine addresses. The guest operating system is responsible for the translation from virtual addresses to physical addresses, known as v2p translation. The virtual machine monitor has to map physical addresses to actual machine addresses, known as p2m translation. Several approaches have been proposed to assist this two levels of address translation. In a para-virtualization system, the guest operating system is updated to directly translate virtual addresses to machine addresses with the assistance of the VMM. Para-virtualization requires modifying an OS and thus is typically not applicable to a data center.

We focus on a full-virtualization system where the guest operating system is intact. A full-virtualization system uses either (SP) or HAP for address translation as shown in Fig. 1 (Devin et al. 1998). In SP mode, the guest operating system still maintains a page table that translates virtual addresses to physical addresses. The VMM maintains a page table, called the shadow page table, that translates virtual memory addresses directly to hardware addresses. Note that a shadow page table has the same structure complexity as the guest page table and thus the address translation cost remains unchanged compared to the native system. However, to maintain the consistency between the guest page table and the shadow page table, whenever there is a guest page table update, the VMM must perform a corresponding update in the shadow page table which requires expensive context switch, called the exit, from the VM to the VMM. To avoid this overhead, Intel (Gillespie 2009) introduced its HAP mode, which is called extended page table (EPT). AMD also proposed a similar mechanism, called nested page table (NPT) (Bhargava et al. 2008). NPT or EPT extends the guest page tables by adding tables for physical to machine address mapping. Now, page table updates can be applied without VM exits. However, the page walk in HAP mode is several times more expensive than that of SP due to nested address translation.

2.2 Memory virtualization optimization techniques

There are several studies on the improvement of memory virtualization. Gillespie (2009) discussed the relationship and differences between SP and HAP without experimental comparison. He offered software vendors, enterprises, as well as individual users a general guideline to determine which mode is better towards certain tasks. VMware, one of the major virtualization software providers, published their performance study in 2008 on HAP and SP and concluded that there is no absolute winner for both modes (VMware 2009). In the same year, VMware also proposed to use super pages in a VM to reduce the TLB misses and thus improve the performance of HAP mode (VMware 2008). Their results showed that, with super pages, the L1 Data TLB misses can be significantly reduced, which results in a

Fig. 1 Shadow paging versus hardware-assisted paging (EPT/NPT)

performance improvement of 8–10 %. However, they did not discuss how super pages can affect the performance of SP mode which is more tolerant to TLB misses. Bhargava et al. presented the detail implementation of AMD's two dimensional (NPT) structure (Bhargava et al. 2008). To reduce page walk penalty, they proposed to cache nested page translations. Their method improved NPT by 3–7 %. Moreover, they also employed the idea of super pages and achieved an improvement up to 22 %. The aforementioned studies suggest that either SP or HAP has its advantages and disadvantages toward various applications. Therefore, a few studies have proposed to select a more suitable mode when dealing with a specific application. Adams et al. conducted an extensive study on software and hardware approaches for x86 architecture virtualization (Adams and Agesen 2006). They showed that software-based virtualization performs better than hardware-assisted virtualization when a workload frequently performs I/O operations, processes creation and contexts switching, while, on the other hand, hardware-assisted virtualization has an edge when a workload contains a large number of system calls. Moreover, they suggested that TLB miss and page fault counts can be used as a criterion to quantify the workload behavior. Following this direction, Wang et al. proposed a dynamic paging switching mechanism by monitoring TLB misses and page fault counts and some other runtime system data (Wang et al. 2011). They proposed a hand-tuned decision model based on manual analyses of the behavior of SPEC CPU2006 Integer benchmarks (SPEC 2006). Independently, Bae et al. proposed a similar dynamic switching mechanism (Bae et al. 2011) that runs on a different hypervisor. Both studies have demonstrated that adaptive paging is promising given that different workloads may exhibit different system behavior. However, their decision models are generalized by hand that require heavy human labor. The models are subjective and specific to a system for which they are designed. This paper employs machine learning approaches to learn an adaptive switching model.

2.3 Support vector machines

SVMs have been widely used since being introduced by Vapnik and his colleagues (1992, 1995). For binary classification, a SVM tries to find a hyper-plane that separates the two

classes so that the margin defined by the distance between hyper-plane and closest point (min) from both classes is maximized (max). The SVM decision function and hyper-plane will be used interchangeably in this paper. SVMs have been successfully employed for various real world applications (Joachims 1998, 2002; Guyon et al. 2002; Collobert and Bengio 2001). However, the application of SVM in system research is very limited. One notable study is by Liao et al. who developed a tuning framework for dynamical configuring the hardware prefetcher in a data center environment based on a SVM model (Liao et al. 2009). A short review of the soft-margin SVM follows.

Let there be l sample points of the form $(\mathbf{x}_i, y_i)$. We call $\mathbf{x}_i$, which has d dimensions, $\mathbf{x}_i \in \mathbb{R}^d$, the features of the sample point and $y_i \in \{-1, 1\}$, the label of the sample point. The linear decision surface (hyper-plane) generated from this training set can be defined as $f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$. Assume a Mercer kernel is available, $K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$ for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$. Then, the weight vector $\mathbf{w}$ and offset b of the decision function are found by solving the following constrained optimization problem:

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{k=1}^l \xi_k, \quad (1)$$

subject to the constraints $y_k(\mathbf{w} \cdot \Phi(\mathbf{x}_k) + b) \geq 1 - \xi_k$ and $\xi_k \geq 0$ for $k = 1, \dots, l$.

2.4 Transfer learning

Transfer learning has been applied in many domains such as document/sentiment classification (Dai et al. 2007; Blitzer et al. 2007), video concept detection (Jiang et al. 2008; Duan et al. 2009), NLP domain adaption (Jiang and Zhai 2007; Dai et al. 2007) and WiFi localization (Pan et al. 2007, 2008; Zheng et al. 2008). We formalize the transfer learning processes as follows using the notation of Pan and Yang (2010). A *domain*, $\mathcal{D}$, is defined as a pair consisting of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathcal{X}$. A *task* $\mathcal{T}$ also has two components, a label space $\mathcal{Y}$ and a function $f: \mathcal{X} \rightarrow \mathcal{Y}$. In traditional machine learning, a task is to be learned for a given domain, that is to learn the best estimate $\hat{f}: \mathcal{X} \rightarrow \mathcal{Y}$ from the training data $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$. Different types of supervised learning problems result from changes to the label space: binary classification: $\mathcal{Y} = \{+1, -1\}$, multi-class classification $\mathcal{Y} = \{1, \dots, L\}$, or regression $\mathcal{Y} \in \mathbb{R}$.

Within transfer learning, let us define a *source* domain $\mathcal{D}_S = \{\mathcal{X}_S, P_S(X)\}$, source task $\mathcal{T}_S = \{\mathcal{Y}_S, f_S\}$, and source training data D_S . Also, let us define a *target* domain $\mathcal{D}_T = \{\mathcal{X}_T, P_T(X)\}$, target task $\mathcal{T}_T \in \{\mathcal{Y}_T, f_T\}$, and target training data D_T . The goal of transfer learning is to improve the learning of the target function f_T by using knowledge from the source domain, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. Transfer learning can be generalized to cases with multiple sources or targets. If there is no difference between the source and target domains, $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$, the problem becomes a traditional inductive learning problem.

In inductive transfer learning, the inductive bias of the target task is affected by knowledge of the source task. The transfer learning methods can be characterized by what and how the knowledge is transferred. There are four main categories of approaches: (i) instance-based approaches, which reuse parts of the data from the source domain in the target domain (Dai et al. 2007; Yao and Doretto 2010); (ii) feature-representation-transfer approaches, where feature representations are identified to improve model performance (Dai et al. 2008), (iii) parameter-transfer approaches, which share parameters and prior distributions of the models

of the source and target (Bonilla et al. 2008; Yao and Doretto 2010), and (iv) relational-knowledge-transfer approaches, where similar relationships exist across source and target domains.

We use an instance-based transfer learning method TrAdaBoost (Dai et al. 2007). TrAdaBoost is short for Transfer learning Adaptive Boosting. Freund and Schapire (1997) proposed that one can adaptively update the weight of misclassified training samples iteratively to improve a weak learner's model performance. Dai et al. (2007) extended this idea to TrAdaBoost, a boosting-based algorithm allowing the user to utilize a small amount of the target domain training data to leverage the source domain data to construct a high-quality model.

The TrAdaBoost method considers learning in the following situation, the training data set, T consisting of labeled training data from the source, T_S , and target domains, T_T , with n_S and n_T samples respectively. Additional test data consists of unlabeled data from the target domain Te_T . Initially, the weight of each training sample is set to 1, $\mathbf{w}^1 = (w_1^1, w_2^1, \dots, w_{n_S+n_T}^1)$. The weight of each data point is updated iteratively by examining if its corresponding training point is misclassified. Since the training samples are from both source domain and target domain, we update the weights in two ways depending on domain information. The weights of misclassified points from target domain grow larger and contribute more to the creation of SVM model. Eventually, the model accuracy improves and converges as the weights of such training points converge. The TrAdaBoost methods is described in Algorithm 1.

Algorithm 1 TrAdaBoost Method

procedure TRADABOOST(T_S, T_T, Te_T, N)

Inputs: T_S, n_S source labeled training samples, T_T, n_T target labeled training samples
 Te_T , unlabeled target test samples, N , number of iterations

% Initialize weight vector,
 $\mathbf{w}^1 = (w_1^1, w_2^1, \dots, w_{n_S+n_T}^1)$

for $t = 1, \dots, N$ **do**

1. Let $\mathbf{p}^t = \mathbf{w}^t / (\sum_{i=1}^{n_S+n_T} w_i^t)$

2. Train SVM model on $T = T_S \cup T_T$ and weights $\mathbf{p}^t$, to get the model $f_t(\mathbf{x})$

3. Calculate the error of f_t on T_T , $\epsilon_t = \sum_{i=n_S+1}^{n_S+n_T} \frac{w_i^t |f_t(x_i) - y_i|}{\sum_{i=n_S+1}^{n_S+n_T} w_i^t}$

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$ and $\beta = 1 / (1 + \sqrt{2 \ln n_S / N})$

5. Update the weight vector, $w_i^{t+1} = \begin{cases} w_i^t \beta^{|f_t(x_i) - y_i|}, & 1 \leq i \leq n_S \\ w_i^t \beta_t^{-|f_t(x_i) - y_i|}, & n_S + 1 \leq i \leq n_S + n_T \end{cases}$

end for

return model, $f_f = \begin{cases} 1, & \prod_{t=\lceil N/2 \rceil}^N \beta_t^{-f_t(x)} \geq \prod_{t=\lceil N/2 \rceil}^N \beta_t^{-\frac{1}{2}} \\ 0, & \text{otherwise} \end{cases}$

end procedure

3 SVM-based decision model

As discussed in Sect. 2.1, the choice between HAP and SP is dependent upon the program runtime behavior, specifically, the frequencies of page walks and page table updates. Based on this observation, we train a decision model off-line to help the hypervisor make dynamic

Fig. 2 Phasing behavior of TLB misses on (a) *mcf* and (b) *milc* SPEC CPU2006 benchmarks

mode switching decisions based on runtime application behavior. This section details our approach on training data collection, labeling, SVM-based model generation, and validation analyses on the generated models.

3.1 Data collection

Note that the frequency of page walks is determined by the frequency of TLB misses and the page table updates are closely correlated to the frequency of page faults. The previous manual decision model by Wang et al. relies on TLB miss count, page fault count, and number of instructions executed in a fixed time interval (Wang et al. 2011). They experimented with a time interval of 2, 5 or 10 s and showed that a 5-s interval performs the best. Since a data point was collected every several seconds and a program usually runs in thousands of seconds or more, thousands of sample points were collected. They scanned through these data points for a set of benchmarks and summarizes a decision tree-like model. This process is tedious even it is just for one system. For a different system, the whole process needs to be repeated.

We automate this process based on a phase-based training data collection and use an SVM to automatically generate a decision model. Note, most applications demonstrate phasing behavior: the application repeats its behavior during execution (Shen et al. 2004; Sherwood et al. 2001, 2003). Figure 2 shows the TLB miss count at every two second interval versus time for *mcf*, a SPEC CPU2006 integer (INT) program, and *milc*, a SPEC CPU2006 floating point (FP) program (SPEC 2006). Taking *mcf* for example, the program demonstrates four phases: A, B, C and D with the rest of the program merely repeating this pattern. For SPEC CPU2006 integer type programs, 11 out of 12 programs exhibit obvious phasing behavior. Most of the SPEC CPU2006 FP programs perform well-structured scientific computing; therefore, they also show phasing behavior. Phase detection can be automated either online or off-line (Shen et al. 2004; Sherwood et al. 2001, 2003; Zhao et al. 2011). We collect training data on the 12 SPEC CPU2006 integer programs. The training data samples are labeled based on the execution time of a phase in HAP mode versus SP mode. For each hardware configuration, a training set was collected running in each mode. There are about five phases in each program resulting in 60–67 training data samples (variation is dependent on the hardware). Each program usually runs in several hundred seconds and the total time for collecting the training data over all the benchmarks is greater than 24 hours.

In addition to the data collected running in only SP or HAP mode, we generate two data sets to model HAP to SP switching and SP to HAP switching. The way to collect these two training sets are similar. Take HAP to SP switching as an example. We start a program in HAP mode. Given the transition point of each phase, we trigger a switch from HAP to SP mode when the program enters the phase. We then keep the program running in SP mode for the same as time span of the phase when in HAP mode, and then switch back to HAP mode. For example, let the program have 5 phases: A, B, C, D and E. We record the running time of this program running under HAP only mode. Then we conduct a mode switch, running one phase A in SP mode and all others including repeated A's in HAP mode and record the run time. The two runs are compared to determine the training sample label. The same method should be applied to B, C, D and E, respectively. Therefore, five sample points and their corresponding labels can be collected.

For labeling a sample point, we take two different approaches. For the first approach, if the mode switching helps a program run in less time, we assign the data point a label +1 indicating a mode switch should be made. Otherwise, a “no switch” label −1 is given. We call this approach *time-gain labeling*. For the second approach, we compute the ratio between the time that is saved due to switching into a different mode and the time spent in the new mode. For example, if a program runs 400 s in HAP only mode. We conduct a mode switching from HAP to SP and runs under SP for 100 s, then switch back to HAP and the total run time becomes 330 s. Compared to HAP only mode, 70 s is saved and 70 over 100 is 0.7. We set the threshold to 0.05, above which a “switch” label is used; conceptually, this can be thought of as only switching if at least a 5 % gain in performance is found in the phase. This approach is called *time-gain-ratio labeling*.

3.2 SVM parameter selection and feature selection

The features we collect as the input for SVM include TLB miss count and page fault count, the number of instructions retired, and historical data such as average TLB miss and page fault count in the past three intervals. A cross-validation procedure is used to select which combination of features, labeling methods, and SVM parameters lead to a model with the best generalization performance (a grid search procedure is performed (Chang and Lin 2011)).

We perform the training on one Intel core i5 machine, one Intel core i7 machine and one AMD machine, whose configurations are listed in Table 1. The number of training samples are 66, 67, and 60 for the i5, i7, and AMD machines respectively. The following subsets of features were considered:

Table 1 Hardware configurations

Hardware configuration	Intel i5 760 processor	Intel i7 920 processor	AMD A8-3850 processor
CPU clock (GHz)	2.8	2.67	2.9
Cores per CPU	4	4	4
L1 DTLB entries	64	64	48
L2 DTL B entries	512	512	1,024
L1 Cache size (KB)	64	128	256
L2 Cache size (KB)	512	1,024	4,096
L3 Cache size (KB)	8,192	8,192	None
Memory size for DOM0 (GB)	0.8	4	4
Memory size for DOMU (GB)	3	4	4

Table 2 Performance of models across sets of features, labeling methods, and SVM parameters

	HW	Feat.	linear			d2poly			d3poly		
			Sen%	Spec%	AUC	Sen%	Spec%	AUC	Sen%	Spec%	AUC
Time-gain	i5	tp	48.0	90.0	69.0	28.0	96.7	62.3	80.0	86.7	83.3
		rtp	28.0	100.0	64.0	28.0	96.7	62.3	64.0	90.0	77.0
		ratp	32.0	100.0	66.0	32.0	96.7	64.3	32.0	100.0	66.0
	i7	tp	79.4	97.0	88.2	82.4	97.0	89.7	97.1	97.0	97.0
		rtp	64.7	100.0	82.4	76.5	97.0	86.7	79.4	97.0	88.2
		ratp	64.7	100.0	82.4	82.4	90.9	86.6	79.4	97.0	88.2
Time-gain-ratio	i5	tp	81.5	100.0	90.7	81.5	97.4	89.5	92.6	94.9	93.7
	i7	tp	86.5	100.0	93.2	100.0	96.0	98.0	100.0	98.0	99.0
	amd	tp	100.0	100.0	100.0	93.1	93.0	91.7	96.6	100.0	98.3
	amd	rtp	93.1	93.5	93.3	93.1	96.8	95.0	100.0	100.0	100.0
	amd	ratp	93.1	96.8	94.9	96.6	96.8	96.7	89.7	96.8	93.2

tp - TLB misses and page faults,

rtp - instruction retired to scale TLB misses and page faults,

ratp - instruction retired to scale TLB misses and page faults, and average TLB misses and page fault of the last three intervals.

The performance of various models estimated from 5-fold cross validation using the best SVM parameters are shown in Table 2. The sensitivity (Sen), specificity (Spec), and area under the ROC curve (AUC) are reported for the best model found using a linear (linear), polynomial kernel of degree 2 (d2poly), and polynomial kernel of degree 3 (d3poly).

First, the best combination of features (Feat.) was considered using the time-gain labeling method on the i5 and i7 machines (top rows of Table 2). The simplest feature combination of TLB miss and page fault counts, *tp*, results in the models with the best performance in terms of AUC for all kernel choices on both the i5 and i7 machine, except that the degree 2 polynomial kernel on the i5 machine where including the number of instruction retired and historical data (*ratp*) improves performance. In general, using the number of instruction retired and historical data does not result in significantly better performing models at the added cost of higher computational complexity and runtime monitoring overhead, therefore the feature set *tp* TLB miss and page fault was selected.

Next, the *time-gain-ratio* labeling method was compared to the results above. The performance of the time-gain-ratio model is much better than the time-gain model. Therefore, all future results will use *tp* feature set with time-gain-ratio labeling.

3.3 Implementation of SVM model

Once the decision model has been learned, then the model must be embedded into the dynamic switching system in such a way as to minimize additional overhead. As described in Sect. 2.3, SVM implicitly maps raw data points from a lower dimensional space to a higher dimensional space by using kernel function. The equation of the SVM hyper-plane is $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + b$, and the SVM decision function is a sign function of $f(\mathbf{x})$. We can calculate $f(\mathbf{x})$ and embed it into the VMM and feed the on-the-fly data into the formula to make decision, where $\mathbf{w} = \sum_s \alpha_s y_s \Phi(\mathbf{x}_s)$, $\mathbf{x}_s$ are support vectors, and α_s are corresponding

Lagrange multipliers. If linear function is selected as the kernel function, data points need not to be mapped into a higher dimension space thus the above formula become $\mathbf{w} = \sum_s \alpha_s y_s \mathbf{x}_s$.

Likewise, if a degree 2 or degree 3 polynomial kernel is selected, we can also compute parameters corresponding to each feature or parameters for the combination of features (monomials) (Brown et al. 2012). For example, assume we have a kernel function mapping:

$$\Phi : R^2 \rightarrow R^3 : (x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2). \quad (2)$$

So long as we have this prior knowledge about the kernel function mapping, we can easily compute the values such as square of feature and inner product of multiple features (the standard heterogeneous polynomial kernel and mapping is considered). Then, we can compute the normal of the hyper-plane $\mathbf{w}$ as discussed before. The decision functions using linear SVM model, degree 2 polynomial SVM model and degree 3 polynomial SVM model for the Intel i7 machine are as follows, where T and P are normalized TLB miss count and page fault count, respectively.

Linear SVM formula:

$$7.11T - 29.95P - 24.70 = 0$$

Degree 2 polynomial SVM formula:

$$9.43 \times 10^2 T - 1.45 \times 10^3 P - 6.13 \times 10^{-5} TP - 9.75 \times 10^{-5} T^2 \\ + 2.19 \times 10^{-2} P^2 + 5720 = 0$$

Degree 3 polynomial SVM formula:

$$1.51 \times 10^{-4} T - 1.11 \times 10^{-4} P - 4.80 \times 10^{-5} T^2 + 2.22 \times 10^{-4} P^2 - 1.03 \times 10^{-4} TP \\ + 3.21 \times 10^{-5} T^3 - 1.91 \times 10^{-4} P^3 - 3.72 \times 10^{-6} TP^2 \\ - 6.03 \times 10^{-6} T^2 P - 49.0 = 0$$

By examining the formulas of these models, the degree 2 and degree 3 polynomial formulas have more terms, need more calculations, and thus require more execution cycles for runtime decision. Moreover, based on the results in Table 2, the higher degree formulas do not bring a significant improvement on model accuracy. Considering runtime efficiency, we select the linear SVM model as the decision function of our dynamic switching mechanism.

While examining the distribution of our training data, we notice that some of the data points that are close to the decision line can trigger a mode transition but the switching will not bring a significant improvement with respect to the overall program running time. In addition, mode switching will result in a slight, though often negligible, runtime overhead. To avoid unnecessary switching, we introduce a grey-area threshold: if a data point is located inside the grey area, e.g., close to the decision line, we do not perform mode switching. We calculate the distance between a sample point and the linear function and set a threshold for the distance. The value of grey-area threshold is determined empirically. The experimental results are discussed in Sect. 4.1.4.

All the input training data are scaled to $[-1, 1]$ before passed to the SVM model. The scaling formulas are as follows:

$$T' = -1 + 2 \times \frac{T - T_{min}}{T_{max} - T_{min}} \\ P' = -1 + 2 \times \frac{P}{P_{max}}, \quad (3)$$

where T' and P' are scaled TLB miss count and page fault count, respectively, T and P are original TLB miss and page fault counts, and T_{max} and T_{min} are the maximum and minimum TLB miss values among all training sets. Finally, we encode the SVM formula in the XEN hypervisor (Barham et al. 2003). However, XEN does not support FP calculation in its kernel. Therefore, we transform the equation and amplify the parameters of the SVM model to make all parameters and calculations integer.

4 Experimental results and analyses

4.1 SVM-based selective switching

In order to validate the effectiveness of our SVM models, we implement the linear model in the XEN hypervisor and use it in online dynamic switching. When a VM is running, we monitor page faults and TLB misses on the fly and feed them into the SVM model. The hypervisor conducts mode switching if the model suggests so. We conduct several experiments under different architectures, different hardware settings, as well as different benchmark programs. The hardware configurations are shown in Table 1.

For the base experiment, we compare our automated SVM model with the hand-tuned manual model. We run experiments on the Intel i5 processor. We set 800 MB of memory for domain0 and 3GB for domainU. We pin only one core to domainU. The host operation system is CentOS 5.4 with Linux kernel 2.6.18. The guest operating system runs CentOS 5.4 with 32-bit Linux kernel 2.6.18. We imported the manual switching model by Wang et al. (2011) which is designed for a similar system. Both models make their own switching decision every 5 s. We run SPEC CPU2006 Integer programs which are also used to train the models.

We compare the manual model (manual-ASP), our SVM-based mode (SVM-ASP) with HAP-only or SP-only mode. Each benchmark is run five times and the mean running time is reported in in Table 3. Figure 3 shows the execution times normalized to the HAP-only mode with 95 % confidence intervals. For many of the benchmarks, the execution time of

Table 3 Intel i5 32-bit SPEC CPU2006 INT average execution times

Benchmark program	HAP (s)	SP (s)	manual-ASP (s)	SVM-ASP (s)
<i>astar</i>	670.6	654.0	655.6	654.4
<i>bzip2</i>	869.6	911.0	872.4	871.2
<i>gcc</i>	413.2	597.8	420.8	413.8
<i>gobmk</i>	642.0	640.6	640.6	641.6
<i>hmmmer</i>	1,250.0	1,250.0	1,250.0	1,250.0
<i>h264ref</i>	1,050.0	1,050.0	1,050.0	1,050.0
<i>libquantum</i>	828.8	826.4	826.6	825.4
<i>mcf</i>	384.6	338.6	341.4	339.6
<i>omnetpp</i>	389.2	381.6	383.6	382.2
<i>perlbench</i>	493.0	495.0	493.0	492.6
<i>sjeng</i>	733.4	730.0	730.6	730.0
<i>xalancbmk</i>	346.0	340.2	342.2	341.0

Fig. 3 Intel i5 32-bit SPEC CPU2006 INT normalized execution times

Table 4 Intel i5 64-bit SPEC CPU2006 INT average execution times under different guest OS

Benchmark program	HAP (s)	SP (s)	manual-ASP (s)	SVM-ASP (s)
<i>astar</i>	628.0	603.0	607.0	605.0
<i>bzip2</i>	694.2	690.0	692.2	690.4
<i>gcc</i>	400.0	400.0	400.0	400.0
<i>gobmk</i>	580.6	577.4	575.4	577.0
<i>hmmer</i>	966.0	966.0	966.0	966.0
<i>h264ref</i>	799.0	799.0	799.0	799.0
<i>libquantum</i>	667.0	665.0	666.0	667.0
<i>mcf</i>	540.4	432.6	436.2	432.6
<i>omnetpp</i>	402.0	381.8	384.0	383.4
<i>perlbench</i>	455.0	455.0	455.0	455.0
<i>sjeng</i>	667.2	661.2	661.0	661.0
<i>xalancbmk</i>	305.4	290.2	295.4	292.6

HAP and SP are similar. The most notable perform differences are a +40 % for *gcc* with SP and −15 % for *mcf* with all non-HAP modes. There are around 5 % differences for several other benchmarks. Both manual-ASP and SVM-ASP are able to match or even outperform the performance of HAP or SP. The mean time of SVM-ASP is even slightly lower than that of manual-ASP, which demonstrates that the automated model can perform as well as the manual model. To further validate the models, we apply them for a different guest OS, a different set of benchmarks, and a different hardware machine.

4.1.1 Verification with different guest OS

We use the same settings except that the guest operating system is CentOS 5.4 with 64-bit Linux kernel 2.6.18. We also use the same linear equations for adaptive switching. Table 4 show the results. Note, when running under 64-bit guest operating system, SP always outper-

Table 5 Intel i5 32-bit SPEC CPU2006 FP average execution times

Benchmark program	HAP (s)	SP (s)	manual-ASP (s)	SVM-ASP (s)
<i>bwaves</i>	779.4	808.2	782.4	779.6
<i>milc</i>	538.4	750.0	538.8	539.0
<i>zeusmp</i>	849.0	829.0	829.0	829.0
<i>cactusADM</i>	1,590.0	1,380.0	1,380.0	1,380.0
<i>leslie3d</i>	899.6	910.6	900.0	899.6
<i>dealII</i>	611.6	626.6	613.2	611.6
<i>soplex</i>	361.6	378.2	364.0	361.0
<i>GemsFDTD</i>	852.6	875.0	853.4	852.3
<i>tonto</i>	950.4	1,140.0	950.6	950.3
<i>lbm</i>	409.0	408.0	408.0	409.0
<i>wrf</i>	1,160.0	1,330.0	1,160.0	1,160.0
<i>sphinx3</i>	722.4	712.6	713.0	712.6
<i>gameess</i>	1,440.0	1,440.0	1,440.0	1,440.0
<i>gromacs</i>	1,060.0	1,060.0	1,060.0	1,060.0
<i>Named</i>	644.0	644.0	644.0	644.0
<i>Calculix</i>	1,770.0	1,800.0	1,770.0	1,770.0

forms HAP. Manual-ASP and SVM-based ASP are able to make right decisions by switching to SP immediately. The results also show that we do not need retrain the models if the guest operating system changes.

4.1.2 Verification with different benchmarks (Testing Set)

We now switch back to the base setting but running a different set of benchmarks. We run SPEC CPU2006 FP benchmarks. The experiment results are shown in Table 5 and Fig. 4. Note that the models are trained based on the SPEC CPU2006 integer programs. Therefore, these floating point programs are regarded as an completely unseen testing set to the SVM models and the manual model. Both selective switching models match the best of the HAP or SP mode results. And for certain benchmark program, SVM-ASP performs slightly better than Manual-ASP, such as *bwaves*, *dealII*, *soplex*. SVM-ASP models are able to pick up the significant performance gaps for *milc*, *tonto*, *wrf*, and *cactusADM*. The results suggest that the SVM-based adaptive switching mechanism works for different workloads and has good generalization ability.

4.1.3 Verification with different switching intervals

In our previous experiments, we set the switching interval to 5 s. Wang et al. (2011) propose that a 5-s interval is appropriate for triggering next decision. In order to verify if 5 s is most suitable for the SVM-ASP mechanism, we test our model using different decision intervals, so as to trigger the decision mechanism for every 2, 5, and 10 s. As shown in Fig. 5 and Table 6, with a 5-s interval, SVM-ASP has a clear edge over that with a 10-s interval in a few benchmarks, namely *astar*, *bzip2*, *gcc*, *omnetpp*, *perlbench* and *xanlancbmk*. The gap between the 2-s interval and the 5-s interval is less significant. However, both *gcc* and

Fig. 4 Intel i5 32-bit SPEC CPU2006 FP normalized execution times

Fig. 5 Intel i5 SPEC CPU2006 INT normalized execution times with different switching intervals

perlbench are in favor of the 5-s interval. We observe a similar trend in the FP programs as shown in Fig. 6 and Table 7. Our results are therefore consistent with the observation by Wang et al. (2011).

4.1.4 Impact of different grey area thresholds

The linear SVM model helps determine when to perform mode switching. However, the model itself does not consider the runtime switching cost. After we obtain the linear SVM model, we calculate the distance between the decision line and each of the training data points. We observe that more than half of the training samples are located near the decision line within a range from 0 to 0.2. When a sample point is too close to the decision line, it might not be profitable to perform mode switching indicated. We introduce a grey area threshold to consider this overhead as discussed in Sect. 3.3. In our experiments, we set up

Table 6 Intel i5 SPEC CPU2006 INT execution times with different switching intervals

Benchmark program	HAP (s)	SP (s)	SVM-ASP switch interval (s)		
			2 s	5 s	10 s
<i>astar</i>	670.6	654.0	654.3	654.4	658.6
<i>bzip2</i>	869.6	911.0	869.3	871.2	875.0
<i>gcc</i>	413.2	597.8	416.3	413.8	424.0
<i>gobmk</i>	642.0	640.6	640.3	641.6	641.0
<i>hmmmer</i>	1,250.0	1,250.0	1,250.0	1,250.0	1,250.0
<i>h264ref</i>	1,050.0	1,050.0	1,050.0	1,050.0	1,050.0
<i>libquantum</i>	828.8	826.4	826.6	825.4	828.3
<i>mcf</i>	384.6	338.6	338.3	339.6	341.3
<i>omnetpp</i>	389.2	381.6	381.6	382.2	388.0
<i>perlbench</i>	493.0	495.0	495.3	492.6	496.0
<i>sjeng</i>	733.4	730.0	731.6	730.0	732.6
<i>xalancbmk</i>	346.0	340.2	341.3	341.0	345.0

Fig. 6 Intel i5 SPEC CPU2006 FP normalized execution times with different switching intervals

a threshold from 0, which means no constraint, to 0.2. The smaller this threshold value is, the more closely the SVM model is followed. A right choice of the threshold is a balance between the overhead of mode switching and the performance improvement gained through running the program under a superior mode. With a threshold beyond 0.2, the SVM model cannot efficiently make decision for most cases.

As can be seen in Fig. 7 and Table 8, a threshold of 0.2 will degrade the performance of *astar*, *omnetpp*, *sjeng*, and *xalancbmk* among the SPEC INT programs with a small percentage when compared to no threshold. These programs run faster under SP mode and have large TLB misses values and small page fault values. However, these data points are located near the decision line within 0.2. A threshold of 0.1 is comparable to no threshold with a slight gain in *gcc*, *libquantum*, and *mcf*. A similar trend in behavior is confirmed

Table 7 Intel i5 SPEC CPU2006 FP execution times with different switching intervals

Benchmark program	HAP (s)	SP (s)	SVM-ASP switch interval (s)		
			2 s	5 s	10 s
<i>bwave</i>	779.4	808.2	780.3	779.6	780.0
<i>milc</i>	538.4	750.0	539.6	539.0	549.6
<i>zeusmp</i>	849.0	829.0	829.0	829.0	830.6
<i>cactusADM</i>	1,590.0	1,380.0	1,380.0	1,380.0	1,382.3
<i>leslie3d</i>	899.6	910.6	900.6	899.6	899.3
<i>dealII</i>	611.6	626.6	612.0	611.6	611.6
<i>soplex</i>	361.6	378.2	361.3	361.0	361.0
<i>GemsFDTD</i>	852.6	875.0	852.3	852.3	859.3
<i>tonto</i>	950.4	1,140.0	950.6	950.3	950.3
<i>lbm</i>	409.0	408.0	409.0	409.0	409.0
<i>wrf</i>	1,160.0	1,330.0	1,160.0	1,160.0	1,160.0
<i>sphinx3</i>	722.4	712.6	713.6	712.6	712.6
<i>gamess</i>	1,440.0	1,440.0	1,440.0	1,440.0	1,440.0
<i>gromacs</i>	1,060.0	1,060.0	1,060.0	1,060.0	1,060.0
<i>Named</i>	644.0	644.0	644.0	644.0	644.0
<i>Calculix</i>	1,770.0	1,800.0	1,770.0	1,770.0	1,770.0

Fig. 7 Intel i5 SPEC CPU2006 INT normalized execution times with different *grey-area* thresholds

on the FP programs as show in Fig. 8 and Table 9. We thus choose 0.1 as the grey area threshold.

4.1.5 Verification with different hardware configuration

We further conduct experiments using a different desktop computer, an Intel i7 machine configured as in Table 1. We allocate 4 GB of memory to domain0 and 4 GB for domainU.

Table 8 Intel i5 SPEC CPU2006 INT execution time with different grey-area thresholds

Benchmark program	HAP (s)	SP (s)	Threshold (s)		
			0	0.1	0.2
<i>astar</i>	670.6	654.0	654.3	654.4	658.3
<i>bzip2</i>	869.6	911.0	870.3	871.2	870.3
<i>gcc</i>	413.2	597.8	421.3	413.8	421.3
<i>gobmk</i>	642.0	640.6	641.3	641.6	640.0
<i>hmmer</i>	1,250.0	1,250.0	1,250.0	1,250.0	1,250.0
<i>h264ref</i>	1,050.0	1,050.0	1,050.0	1,050.0	1,050.0
<i>libquantum</i>	828.8	826.4	827.0	825.4	828
<i>mcf</i>	384.6	338.6	341.3	339.6	339.3
<i>omnetpp</i>	389.2	381.6	382.3	382.2	389.6
<i>perlbench</i>	493.0	495.0	493.0	492.6	493.0
<i>sjeng</i>	733.4	730.0	730.6	730.0	733.6
<i>xalancbmk</i>	346.0	340.2	341.3	341.0	344.6

Fig. 8 Intel i5 SPEC CPU2006 FP normalized execution times with different thresholds

The host OS and the guest OS are the same as the base setting. Using the i5 model on the i7 machine, the experiment results are shown in Table 10 and Fig. 9.

The Intel i5 and i7 have similar hardware configuration. As can be seen from Table 10 and Fig. 9, SVM-ASP performs well in a machine with similar hardware setting. Intel i7 CPU frequency is 2.67 GHz while Intel i5 CPU frequency is 2.8 GHz. The data TLB sizes are the same for both chips. Though we sampled and trained a SVM model independently for the two machines, either model can be applied to both machines with almost the same performance.

Based on the above experiments, we can conclude that a model trained on one machine can be used in the same machine for different guest systems and different benchmarks. It can also be used in a different machine with similar architecture and hardware configuration. Now the question is if the model can be used for a different hardware architecture. We pick

Table 9 Intel i5 SPEC CPU2006 FP execution time with different grey-area thresholds

Benchmark program	HAP (s)	SP (s)	Threshold (s)		
			0	0.1	0.2
<i>bwaves</i>	779.4	808.2	780.3	779.6	780.0
<i>milc</i>	538.4	750.0	540.6	539.0	539.3
<i>zeusmp</i>	849.0	829.0	829.0	829.0	829.6
<i>cactusADM</i>	1,590.0	1,380.0	1,380.0	1,380.0	1,380.0
<i>leslie3d</i>	899.6	910.6	899.6	899.6	900.0
<i>dealII</i>	611.6	626.6	615.6	611.6	611.6
<i>soplex</i>	361.6	378.2	361.3	361.0	362.0
<i>GemsFDTD</i>	852.6	875.0	853.0	852.3	852.6
<i>tonto</i>	950.4	1,140.0	952.6	950.3	950.6
<i>lbm</i>	409.0	408.0	409.6	409.0	408.6
<i>wrf</i>	1,160.0	1,330.0	1,160.0	1,160.0	1,160.0
<i>sphinx3</i>	722.4	712.6	713.0	712.6	721.6
<i>gamess</i>	1,440.0	1,440.0	1,440.0	1,440.0	1,440.0
<i>gromacs</i>	1,060.0	1,060.0	1,060.0	1,060.0	1,060.0
<i>Namd</i>	644.0	644.0	644.0	644.0	644.0
<i>Calculix</i>	1,770.0	1,800.0	1,770.0	1,770.0	1,770.0

Table 10 Intel i7 SPEC CPU2006 INT execution times with i5 model

Benchmark program	HAP (s)	SP (s)	manual-ASP (s)	SVM-ASP (s)
<i>astar</i>	647.4	631.4	640.6	632
<i>bzip2</i>	842.4	847.4	846.2	844.6
<i>gcc</i>	428.2	609.0	424.4	424.4
<i>gobmk</i>	652.0	651.4	652.0	652.0
<i>hmmer</i>	1,260.0	1,260.0	1,260.0	1,260.0
<i>h264ref</i>	1,044.0	1,040.0	1,046.0	1,042.0
<i>libquantum</i>	869.8	869.0	868.6	870.0
<i>mcf</i>	399.0	352.6	351.0	350.6
<i>omnetpp</i>	404.0	394.8	399.0	396.4
<i>perlbench</i>	500.8	502.2	500.8	500.4
<i>sjeng</i>	753.2	750.0	749.8	751.4
<i>xalancbmk</i>	362.6	358.0	359.8	359.0

the AMD machine as listed in Table 1 for the next set of experiments. The AMD processor and the two Intel processors have different hardware architectures and AMD has its own HAP support (called NPT) rather than Intel's EPT. Moreover, the Level 2 data TLB size of the AMD processor is twice as the Intel machines. The TLB miss count is much lower for most of benchmarks in the AMD machine. When directly applying the model obtained from the Intel machines, we observe that the execution time of manual-ASP or SVM-ASP is

Fig. 9 Intel i7 SPEC CPU2006 INT normalized execution times with i5 model

Table 11 AMD SPEC CPU2006 INT execution times with i5 model and AMD model

Benchmark program	HAP (s)	SP (s)	i5 SVM-ASP (s)	AMD SVM-ASP (s)
<i>astar</i>	832.0	798.0	810.0	797.6
<i>bzip2</i>	1,070.0	1,090.0	1,070.0	1,070.0
<i>gcc</i>	618.6	821.4	621.0	619.6
<i>gobmk</i>	651.0	650.2	650.6	651.0
<i>hmmer</i>	980.8	980.4	980.6	980.3
<i>h264ref</i>	1,142.0	1,142.0	1,142.0	1,142.0
<i>libquantum</i>	1,192.0	1,192.0	1,192.0	1,192.0
<i>mcf</i>	811.4	763.8	811.6	764.0
<i>omnetpp</i>	640.8	627.4	640.3	623.6
<i>perlbench</i>	523.0	531.6	523.3	523.6
<i>sjeng</i>	826.0	828.2	826.3	826.0
<i>xalancbmk</i>	532.0	524.0	531.6	524.0

sometimes in-between the execution time of HAP and SP only mode and often close to the slower one as shown in Table 11 and Fig. 10.

To obtain an accurate model for the AMD machine, a new training set was obtained for a new SVM model. Table 11 and Fig. 10 shows that the new model is able to always match the better performance of HAP and SP, which suggests that our current SVM-based approach works for a new architecture but it requires retraining the model. Repeating the data collection and training process is time consuming and even impractical for machines with different hardware configuration in a data center environment. In this study, it takes over a day each to generate the training set for the Intel and AMD machines. Therefore, with all the information acquired from the original machine, we employ a transfer learning method to see if we can shorten the profiling process whenever we need to deploy the switching mechanism to a new machine with different hardware settings. The experiment results are discussed in the following section.

Fig. 10 AMD SPEC CPU2006 INT normalized execution times with Intel i5 model and AMD model

4.2 Transfer learning

Traditional machine learning makes an assumption that training data and testing data are of the same distribution. However, this assumption does not hold for the models across different machines. As can be seen from the experiments in Sect. 4.1.5, one cannot simply apply a model trained from one machine to another one with a different hardware configuration. The solution we have discussed requires to repeat the training process on a new architecture. Although it is a significant improvement over the previous solution that is based on a manual model, the profiling time can still be prohibitive for a data center. This section explores transfer learning techniques to minimize the training time for a new machine while maintaining a high prediction accuracy for the predictive model. As shown in Table 1, the hardware configurations are different between the Intel i5 and AMD A8-3850 machines. We use the Intel i5 data as the source training samples and select a subset of the AMD data as the target training set and the rest of the AMD data is treated as the target testing set. Since the data from source and target domain are of different distribution, we need to scale the data first. We adapt two scaling schemes as listed below.

- *Scale together.* We randomly pick a subset of AMD raw data, combine them with the Intel raw data, and then apply scaling to generate the training set. We record the parameters, such as the upper bound and lower bound of the features and use them to scale the rest of AMD data.
- *Scale separately.* Pick a subset of AMD raw data, then scale them and record the scaling parameters. Scale the Intel raw data independently and combine the scaled AMD data and scaled Intel data together to form the training set. At last, use the AMD scaling parameters to scale the rest of AMD raw data as the testing set.

We conduct three groups of experiments. First, we investigate the number of samples necessary to train a new SVM model in the target domain. We randomly pick 4, 6, 8, 10 AMD data samples respectively to form a training data set then construct a SVM model. To be more specific, for each subset of different size, we randomly pick the given number of data samples, making sure that the quantity of positive samples and the negative ones are equal,

Table 12 Transfer learning model prediction performance

Train set	Accuracy %	Combined train set	Scale separately		Scale together	
		TrAdaBoost?	No	Yes	No	Yes
Intel	52					
AMD4	90 $\pm$ 10	Intel+AMD4	94 $\pm$ 4.3	97 $\pm$ 2.4	73 $\pm$ 11.1	94 $\pm$ 3.9
AMD6	90 $\pm$ 11.8	Intel+AMD6	97 $\pm$ 2.3	98 $\pm$ 2.1	73 $\pm$ 12.2	96 $\pm$ 5.6
AMD8	96 $\pm$ 2.2	Intel+AMD8	97 $\pm$ 1.6	97 $\pm$ 1.9	81 $\pm$ 1.6	97 $\pm$ 1.6
AMD10	96 $\pm$ 1.8	Intel+AMD10	97 $\pm$ 1.6	97 $\pm$ 1.6	87 $\pm$ 6.5	97 $\pm$ 2.2

and repeat this process for another nine times. Thus, ten groups of SVM models are generated and the average prediction accuracy is reported.

Second, we combine a subset of the AMD data with all the Intel data together to create a SVM model to examine the impact on the modeling performance if the training data originated from a different domain is adopted. Note, in this setting the data from the source domain is used directly with no modification other than the scaling methodology used. Lastly, transfer learning is explored via an instance-based method. TrAdaBoost is employed to verify if such a method can alleviate the negative influence by the different distribution of the source domain training data and improve the model prediction accuracy (Dai et al. 2007) (pseudocode in Algorithm 1, Sect. 2.4). When we apply TrAdaBoost towards the learning model, a toolbox named libsvm-weights-3.17 is used rather than libsvm-3.12. Instead of only taking TLB misses and PF as the inputs, an additional input of sample weights is fed into the model.

Table 12 shows the prediction accuracies of the different strategies with standard deviations. If we train a model for the AMD machine using only the Intel data, we can only achieve a 52 % prediction accuracy, suggesting that the model performance is almost as bad as a random guess. As shown in the first two columns, if we only use the AMD data points as the training samples for the AMD machine, the more data samples we have, the higher accuracy we can achieve. Although, by using only 4 AMD data points, we could generate a model with a 90 % prediction accuracy, the standard deviation is relatively high; the error and variation go down when the number of sample points increases to 8 or more.

The remaining columns in Table 12 show the results for the two scaling schemes with and without transfer learning. After incorporating with all Intel data points, we observe that, if we scale the source domain and the target domain separately, the distribution of the Intel i5 data and AMD data would be very similar to each other since we profile the same programs on both machines. On the other hand, if we combine the Intel and AMD raw data together and scale them as a whole set, the distribution of source domain and target domain data are different because the number of the TLB entries are significantly different between the two machines, which leads to different TLB miss distribution. This explains why the prediction accuracies have a significant disparity between the two scaling schemes, where independent scaling shows a higher accuracy.

With the help of the TrAdaBoost algorithm, we gain a slight improvement if the data samples are scaled separately and an appreciable improvement if the data samples are scaled together. In general, we can always obtain a model with a prediction accuracy over 95 % using the transfer learning method. Moreover, we observe that, due to the randomness of AMD training samples, some of the models demonstrate good performance others do not. Without TrAdaBoost, the standard deviation is larger when using a small number of the AMD sample points.

Table 13 AMD SPEC2006 INT execution times using scale together scaling for models with lowest and highest accuracy

Benchmark program	HAP (s)	SP (s)	model w/o		model w.	
			TrAdaBoost (s)		TrAdaBoost (s)	
			$M1_{worst}$	$M1_{best}$	$M2_{worst}$	$M2_{best}$
<i>astar</i>	832.0	798.0	812.6	804.6	804.6	798.0
<i>bzip2</i>	1,070.0	1,090.0	1,070.3	1,070.0	1,070.0	1,070.3
<i>gcc</i>	618.6	821.4	616.6	617.3	619.0	620.6
<i>gobmk</i>	651.0	650.2	651.0	650.3	650.6	650.6
<i>hmmer</i>	980.8	980.4	980.3	980.6	980.6	979.6
<i>h264ref</i>	1,142.0	1,142.0	1,142.0	1,142.0	1,142.0	1,142.0
<i>libquantum</i>	1,192.0	1,192.0	1,191.3	1,190.6	1,192.0	1,191.3
<i>mcf</i>	811.4	763.8	810.6	764.0	763.6	764.6
<i>omnetpp</i>	640.8	627.4	641.3	628.0	627.6	625.0
<i>perlbench</i>	523.0	531.6	523.3	523.0	523.3	523.3
<i>sjeng</i>	826.0	828.2	826.3	825.6	826.0	826.3
<i>xalancbmk</i>	532.0	524.0	533.3	527.3	524.6	524.0

Note that when using scaling together fashion, without TrAdaBoost, the model generated from all Intel plus 8 AMD sample data has relatively lower standard deviation than the one created from all Intel and 10 AMD data. We expect that the model should give higher accuracy and lower standard deviation when more samples are added into the training set. There are possible explanations to such phenomenon. Firstly, the expectation usually only holds on the condition that training data and testing data are of same distribution. Moreover, because of the randomness of training samples selection, we fortunately picked some points that are located at the critical region near the optimal decision plane. For example, for some groups of the *Intel + AMD10* training set, we picked *omnetpp* and *xalancbmk*, while the other groups doesn't contain such samples, but rather select data points from *mcf* or *astar*, which are extremely high in TLB misses. Thus the latter ones do not contribute much to model accuracy. Therefore some of models reaches as high as 98 % prediction accuracy but some only stay around 80 %. Thus result in turbulence in the standard deviation value.

These results illustrate how the use of knowledge from the source domain (data samples) may improve learning in the target domain. Use of TrAdaBoost generally improves the learning performance over taking the source domain data directly. These results suggest that transfer learning is an applicable approach to learn a model for a new machine with a minimal training cost for the the new machine.

In order to verify whether transfer learning help improve performance, we use the Intel and 4 AMD samples as the training set and train two models, one is with TrAdaBoost and the other is not. We embed them into XEN on the AMD machine and record the runtime of each benchmark program. Note that in the previous experiment, for each setting, we created 10 groups and train 10 SVM models respectively. Thus, we pick the models with the highest accuracy (M_{best}) and the one with the lowest accuracy (M_{worst}) for each scaling method to show the estimated runtime upper bound and lower bound.

From Table 13 and Fig. 11 we can see that, with the help of TrAdaBoost, we could achieve the performance as good as using the model derived from all AMD sample points

Fig. 11 AMD SPEC2006 INT normalized execution times using scale together scaling for models with lowest and highest accuracy

when compared with the results in Table 11. Due to its small deviation, with TrAdaBoost, even the model with the worst accuracy is comparable to the one with the best accuracy, suggesting the effectiveness of transfer learning. The model trained from Intel and 4 AMD data points without TrAdaBoost does not perform as well.

Thus, the runtime of most programs is similar to the HAP-only runtime. The only exception is *astar*, where the TLB misses in certain phases are large enough to trigger a switch-to-SP decision. Its runtime is in-between of the HAP-only and SP-only execution times. Although without TrAdaBoost, the SVM model can achieve comparable performance in the best case, the actual model performance is not stable and unpredictable due to large variation and the poor performance demonstrated in the worst case.

5 Conclusions and future work

In this paper, we propose to learn a decision model for virtual machine paging mode switching. We conduct several experiments to test the performance of an SVM-based *Adaptive Switching* mechanism. Whenever, there is a significant gap between HAP and SP, our *Adaptive Switching* mechanism can match the better one. Also, experiments suggest that using this machine learning technique is competitive compared to the hand-tuned switching mechanism. Moreover, the manual model cannot adapt to a new architecture, where a new SVM model may be trained.

The overall gains in execution time are not great for the SVM *Adaptive Switching* mechanism, because there are few benchmark applications where the virtual memory management mode exhibits great differences to be capitalized. However, this work demonstrates the overall feasibility of the use of machine learning methods in data center resource management decisions. We plan to explore the use of machine learning methods to more complicated data center tasks, that may involve additional software and hardware features to be collected.

For a large scale data center, it might be still impractical to train different models for different hardware configurations in a timely-manner even though the training process is

automated. We believe transfer learning will be a viable solution to alleviate the cost and help transfer a model from one architecture to another architecture.

Acknowledgments We thank the anonymous reviewers and the editors for their constructive comments. We also thank Yingwei Luo, Xiaolin Wang, Lingmei Weng, and Jiarui Zang for their comments and help on this work. This work is supported in part by NSF Career CCF0643664 and the National Science Foundation of China Grant No. 61232008, 61272158 and 61328201.

References

- Adams, K., & Agesen, O. (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, (pp. 2–13), New York, ACM.
- Bae, C.S., Lange, J.R., & Dinda, P.A. (2011). Enhancing virtualized application performance through dynamic adaptive paging mode selection. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, (pp. 255–264), New York, ACM.
- Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al. (2003). Xen and the art of virtualization. *SIGOPS Operating Systems Review*, 37(5), 164–177.
- Bhargava, R., Serebrin, B., Spadini, F., & Manne, S. (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, (pp. 26–35), New York, ACM.
- Blitzer, J., Dredze, M., & Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the Association for Computational Linguistics*, ACL07. ACL.
- Bonilla, E.V., Chai, K.M.A., & Williams, C.K. (2008). Multi-task gaussian process prediction. In *Proceedings of the Conference on Neural Information Processing Systems*, NIPS'08, (pp. 153–160).
- Boser, B.E., Guyon, I.M., & Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on COLT*, (pp. 144–152) 1992.
- Brown, L. E., Tsamardinos, I., & Hardin, D. P. (2012). To feature space and back: Identifying top-weighted features in polynomial support vector machine models. *Intelligent Data Analysis*, 16(4), 551–579.
- Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 27:1–27:27.
- Collobert, R., & Bengio, S. (2001). Svmtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1, 143–160.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.
- Dai, W., Chen, Y., Xue, G.-R., Yang, Q., & Yu, Y. (2008). Translated learning: Transfer learning across different feature spaces. In *Proceedings of the Conference on Neural Information Processing Systems*, (pp. 353–360), 2008.
- Dai, W., Xue, G.-R., Yang, Q., & Yu, Y. (2007). Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'07, (pp. 210–219), SIGKDD: ACM.
- Dai, W., Xue, G.-R., Yang, Q., & Yu, Y. (2007). Transferring naive bayes classifiers for text classification. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, AAAI'07, (pp. 540–545), AAAI: AAAI Press.
- Dai, W., Yang, Q., Xue, G.-R., & Yu, Y. (2007). Boosting for transfer learning. In *Proceedings of 24th International Conference on Machine Learning*, ICML07. ICML, 2007.
- Devin, S., Bugnion, E., & Rosenblum, M. (1998). Virtualization system including a virtual machine monitor for a computer with a segmented architecture, Oct. 1998. US Patent, 6397242.
- Duan, L., Tsang, I.W., Xu, D., & Maybank, S.J. (2009). Domain transfer svm for video concept detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'09, (pp. 1375–1381), IEEE, 2009.
- Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119–139.
- Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1–3), 389–422.
- Jiang, J., & Zhai, C. (2007). Instance weighting for domain adaptation in nlp. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, ACL07, (pp. 264–271), ACL, 2007.

- Jiang, W., Zavesky, E., Chang, S.-F., & Loui, A. (2008). Cross-domain learning methods for high-level visual concept classification. In *Proceedings of the 15th IEEE International Conference on Image Processing, ICIP'08*, (pp. 161–164), IEEE, 2008.
- Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning, EMCL'98* (pp. 137–142). London, UK: Springer-Verlag.
- Joachims, T. (2002). *Learning to classify text using support vector machines: Methods, theory and algorithms*. Norwell: Kluwer Academic Publishers.
- Liao, S.-W., Hung, T.-H., Nguyen, D., Chou, C., Tu, C., & Zhou, H. (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*, (pp. 56:1–56:10), New York, 2009. ACM.
- Gillespie, M. (2009). Best practice for paravirtualization enhancements from intel virtualization technology: Ept and vt-d. Technical report, 2009.
- Pan, S. J., Kwok, J. T., Yang, Q., & Pan, J. J. (2007). Adaptive localization in a dynamic wifi environment through multi-view learning. In *Proceedings of the 22nd National Conference on Artificial Intelligence, AAAI'07*, (pp. 1108–1113). AAAI: AAAI Press.
- Pan, S. J., Shen, D., Yang, Q., & Kwok, J. T. (2008). (2008) Transferring localization models across space. In *Proceedings of 23rd National Conference on Artificial Intelligence, AAAI'08*, (pp. 1383–1388). AAAI: AAAI Press.
- Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345–1359.
- Shen, X., Zhong, Y., & Ding, C. (2004). Locality phase prediction. In *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, ASPLOS XI*, (pp. 165–176), New York, 2004. ACM.
- Sherwood, T., Perelman, E., & Calder, B. (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*, (pp. 3–14), Barcelona, 2001.
- Sherwood, T., Sair, S., & Calder, B. (2003). Phase tracking and prediction. In *Proceedings of the 30th Annual International Symposium on Computer architecture, ISCA '03*, (pp. 336–349), New York, 2003. ACM.
- SPEC CPU2006. <http://www.spec.org/cpu2006>
- VMware. (2008). Large page performance: Esx server 3.5 and esx server 3i v3.5. Technical Report.
- VMware. (2009). Performance evaluation of intel ept hardware assist. Technical Report.
- Waldspurger, C. A. (2002). Memory resource management in vmware esx server. *SIGOPS Operating Systems Review*, 36(SI), 181–194.
- Wang, X., Zang, J., Wang, Z., Luo, Y., & Li, X. (2011). Selective hardware/software memory virtualization. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, VEE '11*, (pp. 217–226), New York, 2011. ACM.
- Yao, Y., & Doretto, G. (2010). Boosting for transfer learning with multiple sources. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume i of CVPR'10, (pp. 1855–1862). CVPR, 2010.
- Zhao, W., Jin, X., Wang, Z., Wang, X., Luo, Y., & Li, X. (2011). Low cost working set size tracking. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'11*, (p 17), Berkeley, 2011. USENIX Association.
- Zheng, V. W., Xiang, E. W., Yang, Q., & Shen, D. (2008). (2008). Transferring localization models over time. In *Proceedings of the 23rd National Conference on Artificial Intelligence, AAAI'08*, (pp. 1421–1426). AAAI: AAAI Press.