SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2016, Vol. 59 092105:1–092105:16
doi: 10.1007/s11432-015-5387-6

vSpec: workload-adaptive operating system specialization for virtual machines in cloud computing

Xinkui ZHAO[1], Jianwei YIN[1]*, Zuoning CHEN[2] & Sheng HE[3]

[1] *College of Computer Science, Zhejiang University, Hangzhou* 310027, *China;*
[2] *National Parallel Computing Engineering Research Center, Beijing* 100088, *China;*
[3] *Jiangnan Institute of Computing Technology, Wuxi* 214081, *China*

Received January 14, 2015; accepted March 27, 2015; published online August 23, 2016

Abstract In general, operating systems (OSs) are designed to mediate access to device hardware by applications. They process different kinds of system calls using an indiscriminate kernel with the same configuration. Applications in cloud computing platforms are constructed from service components. Each of the service components is assigned separately to an individual virtual machine (VM), which leads to homogeneous system calls on each VM. In addition, the requirements for kernel function and configuration of system parameters from different VMs are different. Therefore, the suit-to-all design incurs an unnecessary performance overhead and restricts the OS's processing capacity in cloud computing. In this paper, we propose an adaptive model for cloud computing to resolve the conflict between generality and performance. Our model adaptively specializes the OS of a VM according to the resource-consuming characteristics of workloads on the VM. We implement a prototype of the adaptive model, vSpec. There are five classes of VM: CPU-intensive, memory-intensive, I/O-intensive, network-intensive and compound, according to the resource-consuming characteristics of the workloads running on the VMs. vSpec specializes the OS of a VM according to the VM class. We perform comprehensive experiments to evaluate the effectiveness of vSpec on benchmarks and real-world applications.

Keywords workload classification, operating system specialization, workload-aware, cloud computing, vSpec, system performance

Citation Zhao X K, Yin J W, Chen Z N, et al. vSpec: workload-aware operating system specialization for virtual machines in cloud computing. Sci China Inf Sci, 2016, 59(9): 092105, doi: 10.1007/s11432-015-5387-6

1 Introduction

Virtual machines (VMs) have replaced blade servers as the basic management unit of large-scale data centers as cloud computing matures. They are managed by virtualization technology that abstracts heterogeneous infrastructures in the data center as a resource pool. In cloud computing platforms, VMs run on the virtualization layer and the physical infrastructure is in a lower layer. The operating system (OS) of a VM mediates physical resources and system calls to fulfill the requirements of applications running on the VM. The OS acts as a hinge between the physical resource and system calls.

* Corresponding author (email: zjuyjw@zju.edu.cn)

© Science China Press and Springer-Verlag Berlin Heidelberg 2016

Software is supplied as a service (SaaS) in cloud computing, and most of the applications in cloud computing platforms are developed using a combination of loose-coupled service components. For example, a typical e-commence application may consist of load balancer service components, application server service components and database server service components. Each of the service components is assigned with an individual VM for performance and privacy. All of the VMs are equipped with an indiscriminate OS no matter what kinds of workloads are running on the VMs.

We call the OS that is commonly used in the industry environment as common OS, which may be Linux or Unix. The common OS indiscriminately allocates the capacity of the CPU, memory, I/O and network. In many applications, the dominant kernel functions and system configuration are different [1,2], so an OS designed as suit-to-all works well in some situations while it is ill-suited in others [3–6]. The conflict between generality and performance has become a stumbling block to high-performance cloud computing [7].

To improve the processing performance of cloud computing platforms, we propose an adaptive model. Our proposed model adaptively specializes the OS of a VM according to the resource-consuming characteristics of its workloads. We implement a prototype called vSpec that overcomes these challenges. Specially, vSpec selects run-time resource-consuming metrics of workloads and considers five classes of workload: CPU-intensive, memory-intensive, I/O-intensive, network-intensive and compound. In this paper, we consider that each VM is assigned to an individual service component, which is a common assumption in cloud computing [8]. Therefore, we will use the workload classification and VM classification interchangeably in the rest of this paper. vSpec provides strategies to specialize the OS for workloads in the five classes accordingly.

In this paper, we describe our experience in designing, implementing and evaluating the workload-adaptive OS specialization model. We make three contributions:

(1) We introduce a new adaptive cloud computing model based on workload-adaptive OS specialization. We adaptively specialize the OS of a VM according to the resource-consuming characteristics of the VM. The specialized OS assigns extra processing capacity and reduces unnecessary operations for the workloads within each specific category.

(2) We implement a prototype of our model, vSpec, which combines mutual information and correlated dependency to choose the most representative metrics to depict workloads. In vSpec, we propose a supervised classification algorithm named Training Set Refreshed Support Vector Machine (TSRSVM) to classify workloads into five classes. We present specializing solutions to reset the system parameters of the OS kernel. vSpec is a self-adaptive system that works automatically without human intervention.

(3) We use our prototype to quantify the potential benefits of the adaptive model for cloud computing. We conduct comprehensive experiments on benchmarks and real-world applications to demonstrate the effectiveness of vSpec.

The remainder of this paper is organized as follows. Section 2 introduces the related work on workload classification and OS specialization. Section 3 describes the structure for workload-adaptive OS specialization. Section 4 explores the detailed design of vSpec. Section 5 explains experiments conducted to evaluate vSpec. Section 6 presents the conclusion and future work.

2 Related work

There has been a series of work on workload classification and OS specialization. Past studies have come up with many strategies in both areas, and we introduce them separately.

2.1 Workload classification

Workload classification algorithms assemble workloads that have similar characteristics and separate those with discrepant characteristics. Mishra et al. [9] discovered tasks with similar resource-consuming values as a workload and gave insight into workload characteristics in a Google cluster. Lin et al. [10] identified applications using the packet size distribution and port number. Zander et al. [11] adopted an expectation

maximization algorithm to assemble workloads based on statistical flow characteristics. Karagiannis et al. [12] classified workloads according to the host behavior at the transport layer. The above solutions classify workloads with unsupervised machine learning algorithms. They discover correlated workloads and assemble them, which is clustering rather than classification. The methods cannot explicitly identify the characteristics of the workloads in each cluster.

To understand the characteristics of each class clearly, several supervised approaches have been proposed [13–17]. The authors of [13] collected running metrics and classified workloads into web server, FTP server and database server with a support vector machine (SVM). Jiang et al. [14] classified applications through flow-level statistics based directly upon standard NetFlow[1)] records, which used a naive Bayes classification method. Zhang et al. [15] used principal component analysis to decrease the dimension of the metrics collected and adopted the K-nearest neighbors algorithm to classify workloads. Zhang et al. [16] used a Bayesian network to select representative performance features systematically and classified workloads into four classes. We compared different machine learning algorithms for workload classification and proposed an improved solution to the SVM classifier in [17]. Compared to the previous work, we present an automatic feature selection algorithm and implement a prototype of the adaptive computing model in this paper. Furthermore, we conduct more comprehensive experiments to evaluate the implementation of our adaptive model.

2.2 Operating system specialization

There are several representative prototypes of a specialized OS, such as Tempo [18], ExOS [19], SPIN [20], MultiLibOS [21], R3TOS [22], SECC [23], PTask [24], Chameleon [25], Unikernels [26] and OS$^{\text{v}}$ [27]. These prototypes analyze the characteristics of workloads and reduce unnecessary kernel components for the workloads accordingly. To a large extent, these prototypes have guided the architecture design for specialized OSs.

Concurrently with the mentioned prototypes, which focus on adaptive OS structure design, there are also a few OS specialization strategies. Seltzer and Small [28] designed a self-monitoring and self-adapting solution for a common OS. Lee et al. [29] proposed a call graph approach to remove abundant functions and reduced interrelationships among applications, system libraries and the Linux kernel. Saez et al. [30] separated the cores in asymmetric multicore processors into fast and slow cores and assigned them to different kinds of workload to ensure efficiency. McNamee et al. [31] proposed a toolkit to assist the specializing, guarding and re-plugging phases in OS specialization. Soules et al. [32] introduced several ways to extend and replace active OS components. Ref. [33] provided an online OS optimization strategy to fulfill the requirements for resources and they improved applications' quality of service. Compared to the previous work, our specialization strategy tries to redistribute the processing capacity of the OS by changing system parameters, and we do not modify the kernel structure of the common OS.

Some work in the literature have proposed specialization solutions for specific types of workload. Soror et al. [34] improved database performance by tuning the configuration of VMs that host the database. Pu et al. [35] specialized the Unix file system incrementally and improved kernel call performance based on re-plugging. The authors of [36,37] optimized the protocol stack for a TCP/IP implementation. Marinos et al. [38] optimized the stack for static web content and DNS server, resulting in higher web-server and DNS throughput and lower CPU usage. Not only can the OS layers be changed; the authors of [39,40] even tested hardware reconfiguration. Compared to the above solutions, our work provides a specialization strategy for four classes of workload rather than a specific workload.

3 Workload-adaptive OS specialization

Our adaptive model for cloud computing originates from traditional MAPE (monitor, analyze, process, execute) loops in a self-adaptive system. The proposed model is an adaptive model since the OS of a VM is specialized adaptively as the workloads running on the VM change.

1) http://en.wikipedia.org/wiki/NetFlow.

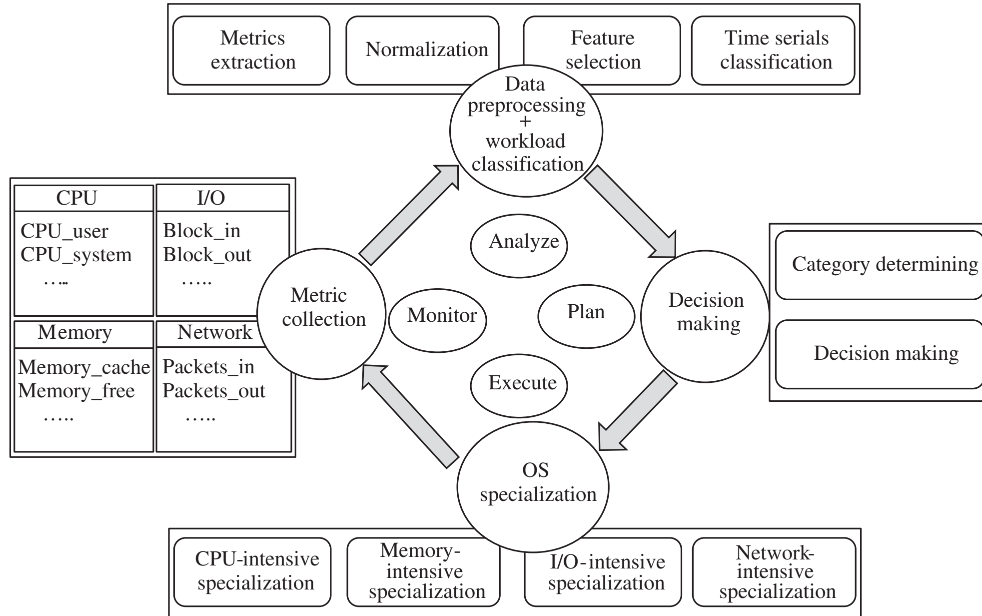

Figure 1 Model for workload-adaptive OS specialization.

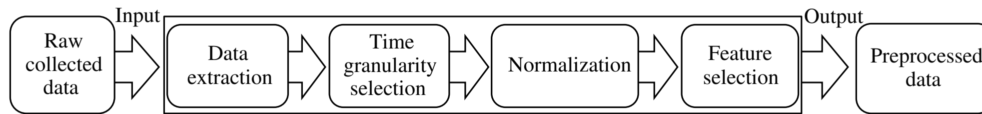

Figure 2 Data preprocessing workflow.

Figure 1 shows the structure of our model. It consists of four processes: run-time monitoring, workload classification, decision-making and OS specialization. In run-time monitoring, we collect run-time metrics to depict the resource-consuming characteristics of workloads. In the workload classification, we preprocess the collected metrics and adopt strategies to classify the preprocessed metric vectors. In decision-making, we choose the class to which the workloads should belong. In OS specialization, we propose a specializing solution for the workloads in each class separately.

4 vSpec design

In this section, we will explore the design and implementation of the prototype, vSpec.

4.1 Metric collection

To depict the resource-consuming characteristics of workloads, we develop a metric collecting tool, which borrows from the structure of Nagios[2)]. The developed metric collector gathers 65 metrics per second, which cover the system, CPU, memory, cache, disk, network, etc. The collected data are stored in a distributed database for fast storage and querying. Details of the structure of the metric collector will be introduced in Subsection 4.5.

4.2 Data preprocessing

In data preprocessing, vSpec selects useful information from the monitoring log to depict the characteristics of workloads. Figure 2 shows the workflow for data preprocessing. Collected data flow through the four processes in the dotted rectangle, which output the preprocessed data. In the first step, vSpec extracts monitoring data to form a matrix. Each row of the matrix stores the values of the 65 metrics

2) http://www.nagios.org.

that belong to a workload at a specific timestamp. A row is a metric vector. The metric vectors of a workload at different timestamps are stored row by row, and the metric vectors of different workloads are connected by row. In vSpec, we set the granularity of the monitoring data to be 1 s.

In normalization, we use the scale equation defined as follows:

$$SV_i = \frac{MV_i - MV_{\min}}{MV_{\max} - MV_{\min}}, \tag{1}$$

where $MV_{\max}$, $MV_{\min}$, MV_i and SV_i are the maximum value of the metric, the minimum value of the metric, the value of the ith metric and the scaled value, respectively.

vSpec adopts a feature selection algorithm to select the most effective metric set (or feature set). The algorithm uses mutual information to calculate the relevance of each metric to the target class. Given a feature vector X_i and a class vector Y, their mutual information is calculated as follows:

$$MI(X_i;Y) = \sum_{y\in Y}\sum_{x\in X_i} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}, \tag{2}$$

where $p(x)$ and $p(y)$ are the margin probability functions of the discrete vectors X_i and Y, and $p(x,y)$ is the joint probability distribution.

To reduce redundancy, we use the Pearson product-moment correlation coefficient (PCC) to calculate the dependency between metrics and filter out the highly correlated metrics. Given two feature metrics X and Y, their dependency weight (PCC value) is calculated as follows:

$$\rho(X;Y) = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E(X)^2}\sqrt{E(Y^2)E(Y)^2}}, \tag{3}$$

where E is the expectation of a variable.

Algorithm 1 shows the pseudocode of our approach to clustering metrics and selecting a representative metric for each cluster. We define a threshold for choosing the highly correlated metrics and adopt a hierarchical clustering algorithm to assemble them (lines 1 to 21 in Algorithm 1). We sum the PCC value of each metric with all the other metrics in the same cluster and select the metric that has the highest sum PCC value as the representative of the cluster (lines 22 to 27 in Algorithm 1).

Algorithm 2 shows the pseudocode of our approach to selecting the final feature for classification. The monitoring data collected from workloads in Table 1 (Subsection 5.1) are separated into two parts. We select two-thirds of the data as the training set and the rest as the testing set. The ratio of the size of the training set and the size of the testing set can be varied. We choose two-thirds of the data as the training set since we have tested many combinations and find that the ratio of two to one works well in our work. We choose the largest n features and filter out the highly correlated features (lines 22 to 27 in Algorithm 1) to form the basic feature set. We traverse n from 1 to 65 and select the set with the highest classification accuracy (lines 5 to 17 in Algorithm 2).

4.3 Workload classification

In workload classification, vSpec classifies workloads into five classes: CPU-intensive, memory-intensive, I/O-intensive, network-intensive and compound. The workloads are characterized with the selected features (Subsection 4.2). We propose an algorithm, TSRSVM, which is an improved form of SVM [17,41]. We choose a supervised classification algorithm to classify the workloads since only supervised classification algorithms are able to distinguish workloads and also show the exact characteristics of workloads in each class.

Figure 3 shows the TSRSVM workflow. TSRSVM first uses the original training data (Subsection 5.1) to train an SVM and it selects the support vectors of the training set to build a basic support vector set (SVs). vSpec transfers testing vectors within a predefined period to the SVM for classification. The SVM with the basic support vector set is used to classify the vectors in the testing set one by one. We compare the percentage of vectors within the selected period that fall into each class and choose the class with

Algorithm 1 Metric clustering and representative metric-selecting algorithm

Require:
 M: metric matrix
 L: target class label for each row in M
 τ: threshold to filter correlated metrics

Ensure:
 c: clustering result
 r: representative metric for each cluster

1: **for** $(i = 1; i < 65; i++)$ **do**
2: **for** $(j = i; j <= 65; j++)$ **do**
3: $P(i,j) = \text{PCC}(M(,i), M(,j))$ // introduced in Eq. 3
4: **if** $P(i,j) > \tau$ **then**
5: **switch** (i,j)
6: **case** $i \in c$ & $j \notin c$**:**
7: store j into i's cluster in c
8: **case** $i \notin c$ & $j \in c$**:**
9: store i into j's cluster in c
10: **case** $i \in c$ & $j \in c$**:**
11: **if** cluster(i) == cluster(j) **then**
12: break
13: **else**
14: combine cluster(i) and cluster(j) as a new cluster in c
15: **end if**
16: **case** $i \notin c$ & $j \notin c$**:**
17: combine them as a new cluster in c
18: **end switch**
19: **end if**
20: **end for**
21: **end for**
22: **for** $(k = 1; k <= c.\text{clusterNumber}; k++)$ **do**
23: **for** each metric in $c(k)$ **do**
24: calculate the sum of P with all other nodes in $c(k)$
25: **end for**
26: $r(k)$ = metric with the largest sum of P
27: **end for**

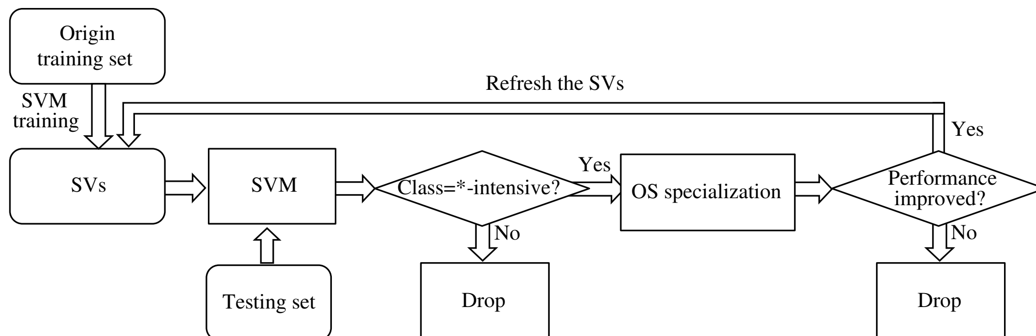

Figure 3 TSRSVM workflow. In this figure, SVs refers to the support vector set.

the largest proportion as the final classification result. We treat the classification result as positive if the workload is classified as CPU-intensive, memory-intensive, I/O-intensive or network-intensive (Subsection 4.4). Otherwise, we treat it as negative. For a positive result, we choose the specializing strategy of the

Algorithm 2 Representative metric-selecting algorithm

Require:

M_{train}, L_{train}: training set and corresponding label

M_{train}, L_{train}: testing set and corresponding label

c: clustering result of metrics

r: representative metric of each cluster

Ensure:

υ: selected feature sets

1: **for** $(i = 1; i <= M.\text{col}; i++)$ **do**
2: $\quad \eta_i = MI(M(:, i), L)$
3: **end for**
4: β= order(η, decreasing = TRUE)
5: **for** $(i = 1; i <= M.\text{col}; i++)$ **do**
6: $\quad \upsilon = \beta(1 : i)$
7: $\quad$ **for** $(j = 1; j <= i; j++)$ **do**
8: $\quad\quad$ **if** $\upsilon(j) \in c(k)$ **then**
9: $\quad\quad\quad \upsilon(j) = r(k)$
10: $\quad\quad\quad$ remove duplicate values from υ
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad$ train SVM with M_{train} and L_{train}
14: $\quad a(i)$ = accuracy to test SVM with M_{train} and L_{train}
15: $\quad$ store $a(i)$ and corresponding υ
16: **end for**
17: return the υ when $a(i)$ is the highest

class accordingly to specialize the OS of the VM. For a negative result, we ignore the classification result. After the specialization process, we use the throughput as a criterion to decide whether the performance of the OS was improved. We treat the classification result as correct if the performance of the OS was improved. We combine all the vectors in the testing set with the basic support vector set to construct a new SVM and a new support vector set. The new SVM and support vector set will be used to classify subsequent testing vectors.

TSRSVM classifies each metric vector of the workloads. However, this does not mean that a VM belongs to a class even if one of the metric vectors of the VM falls into that class. Much extra time will be consumed in changing the specializing strategy back and forth if we use only one metric vector to determine whether to specialize the OS or not. Hence, in Subsection 4.4 we develop a decision-making strategy to determine the workload category from the vector classification result.

4.4 Decision-making

In the decision-making process, vSpec determines the class which a workload should belong to based on the classification results of the metric vectors. The classification is invoked every 5 min in our work. The time granularity of the selected metrics is 1 s, so the total number of vectors to be classified in each period is 300. We set the proportion value of a class as the percentage of vectors that fall into the class within 300 vectors.

In vSpec, a workload is treated as a specific resource-intensive workload when one of the four proportion values (except for compound) is higher than 0.8. We set the threshold as 0.8 based on our project experience and several trials. A workload may be classified into more than one class if the threshold is too small, while it will sometimes not specialize the OS if the threshold is too high. We compared the classification accuracy for different thresholds and found that 0.8 is the best choice in our work.

In vSpec, we do not immediately specialize the OS after we determine the class of the workload in

CPU-Intensive	Memory-Intensive	I/O-Intensive	Network-Intensive
rqbalance	rdisk-swap	vm.pagecache	net.ipv4.tcp_sack
smp_affinity	swappiness	vm.bdflush	net.ipv4.tcp_max_syn_backlog
xps_cpus	netram-swap	vm.kswapd	net.ipv4.tcp_keepalive_time
rps_sock_flow_entries	kernel.shmmax	net.ipv4.tcp_wmem	net.core.wmem_max
	kernel.shmall	net.ipv4.tcp_rmem	net.core.rmem_max
…..	…..	vm.max-readahead	net.ipv4.tcp_timestamps
		…..	…..

Figure 4 Reference specialized parameters and kernel functions for different workload classes.

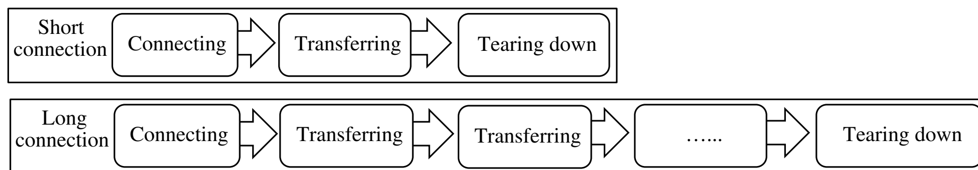

Figure 5 Life cycle of a long connection and short connection.

case the classification is wrong and specialization would increase the cost. The specializing program is invoked when two conditions are fulfilled: 1) the class of the workload in 5 min is the same as the class of the workload in the previous 5 min; and 2) the largest proportion value in this 5 min is larger than the value for the previous 5 min.

4.5 Operating system specialization

In vSpec, there are four specializing strategies: CPU-intensive, memory-intensive, I/O-intensive and network-intensive. We summarize the dominant kernel functions and system parameters, which are shown in Figure 4, according to the workloads' demand for resources and kernel support. Kernel functions and system parameters are specialized for each category to make full use of the dominant physical resource.

For CPU-intensive workloads, we bind the processes that cause interrupts to certain CPUs to reduce the adverse impact caused by interrupts. We also change the I/O scheduling strategy from interrupts to polling to reduce the time cost for the device in delivering and balancing the processor allocation for workloads according to the CPU affinity [42]. For memory-intensive workloads, we increase the page size to reduce the number of translation lookaside buffer misses. We increase the swap space and the swappiness parameter to reduce the chance of the system crashing when the physical memory runs out. We change the maximum allowable size of the shared memory segment (kernel.shmmax) to reduce the creation time of a shared memory segment. For I/O-intensive workloads, we increase the parameters for the page cache to reduce the number of disk reads, and we reset vm.dirty_background_ratio and vm.dirty_ratio to control the percentage of memory that can become dirty before a background flushing of the pages to disk starts. We increase the percentage of memory that can be occupied by dirty pages before a forced flush starts. For network-intensive workloads, we separate network connections into long connections and short connections. Figure 5 shows the life cycle of a long connection and a short connection. A long connection holds the established connection for a longer period since data will be transmitted continuously, which reduces the time cost for a three-way handshake when the connection is established and for a four-way handshake when the connection is disconnected. A short connection tears down the establishment after one data transfer to release the resources. We assign incoming requests to different connection types for more effective connection processing.

4.6 Implementation

vSpec is based on a hierarchical design, which benefits for the federations of virtual clusters in cloud computing. It distributes monitoring probes onto each monitored VM to collect resource-consuming information about the VMs. Each of the managing nodes is in charge of several monitored nodes. The

data collected on monitored nodes are transferred to a management node. A managing node organizes the received data and sends them to the root node for preprocessing and classification. We set the time granularity, for when each probe sends the monitored data to the managing node, as 1 s. The time granularity to send organized data from managing nodes to the root node is set as 5 min.

We implement four specialization solutions, which use the proc file system and sysctl command to modify the kernel configuration and system parameters for the OS of a VM. The proc file system and sysctl command ensure that the OS is specialized without turning the VM down. The source files of a specializing program are stored in the image files of each VM, and these files are executed when the decision-making finishes and gives an invoke instruction. In vSpec, we keep the OS unchanged when the workloads are classified as compound.

We design the run-time monitor, workload classification, decision-making and OS specialization in a loose-coupled manner. Each of the components has an independent interface to make the algorithms used in them flexible for different usage scenarios. We implement the prototype on an Openstack platform[3)] with KVM as the virtualization platform.

5 Experimental evaluation

5.1 Experimental setup

The experiments are conducted on 60 VMs. Each of the VMs is allocated with two CPU cores, 2 GB of memory and a 52-GB disk. These VMs are hosted on five Intel(R) Xeon(TM) Gainestown 2.40-GHz physical servers and connected by Intel 82545EM Gigabit Ethernet.

5.1.1 *Training and testing sets*

We select 33 benchmarks and collect information about their resource consumption to construct the training and testing sets. These benchmarks are from websites, such as Phoronix-Test-Suite[4)] and SPEC Benchmarks[5)], and give a predefined classification result. We take this as a reference and test the benchmarks with the OS specialization strategies to verify whether the predefined result is suitable for our work. If the selected specializing solution improves upon the benchmark result, we label the predefined classification result as correct, otherwise, we treat the benchmark as ambiguous and drop the benchmark.

Table 1 lists the 25 benchmarks for the training set and Table 2 lists the eight benchmarks for the testing set. We select well-known benchmarks rather than real-world applications for our training and testing sets since the categories of the benchmarks are determined. That the classes of workloads are determined is a preliminary for comparing the accuracy of the workload classification.

5.1.2 *Real-world applications for evaluation*

Besides the 33 benchmarks selected for the training and testing sets, we select three real-world applications for our experiments to evaluate the effectiveness of vSpec:

- A Hadoop cluster. We calculate PI with an approximation approach, sort words with a TeraSort application, and we test operations for reading from and writing to HDFS on the Hadoop cluster [43].
- A data distribution service for internet protocol television (IPTV). The IPTV application is implemented by our research group, and it processes publish-subscribe communications for TV voting [44].
- A database testing application, JtangTest. This is an application for comparing data storage and query performance in application performance management (APM). We reorganize the monitored data from a Google cluster [9] and load the data into the testing database. The application can find bottlenecks in a distributed APM before the APM is deployed in a production system.

3) https://www.openstack.org/.
4) http://www.phoronix-test-suite.com/.
5) https://www.spec.org/benchmarks.html.

Table 1 Workloads in the training set

Category	Benchmark	Description
CPU-intensive	N-Queens	Method to calculate nqueens problem
	Cray	Ray tracer for floating-point CPU calculations
	SysTester	Benchmark for CPU stress test
	Apache	Test Apache process capability for existing requests
	Hadoop_PI	PI calculation on Hadoop platform
Memory-intensive	MemTester	Test of a memory subsystem for a userspace utility
	RamSpeed	Test cache and memory
	SysBench	Test memory of distributed caching system
	CacheBench	Test read from, modify and write to memory operations
	IBS_Mem	Simulation of `malloc` memory space
I/O-intensive	FSMark	Test file-system performance
	HQParm	Get and set ATA/SATA drive parameters
	IOzone	Test reading and writing to file system
	Hadoop Texterwriter	Random text to test reading and writing on HDFS
	Hadoop TeraGen	Data generation for TeraSort on Hadoop
Network-intensive	NetTest	Measure performance of different types of network
	Ettcp	Measure network performance at the TCP/UDP level
	ApacheBench	Test HTTP server's request receive capacity
	Iperf	Test TCP and UDP bandwidth performance
	TTCP	Simulation for TCP transfer ability
Compound	Postmark	Simulation of web and mail servers
	Hadoop nnbench	MapReduce process with small workload
	IBS	IBS process to simulate CPU with low utilization
	eSpeak	Speech synthesizer to read a book
	Build-PHP	Build process for small PHP program

Table 2 Workloads in the testing set

Category	Benchmark	Description
CPU-intensive	Ffmpeg	Tool to record, convert and stream audio and video
	SysBench_CPU	Calculation of prime numbers up to a value
Memory-intensive	MemCache	Distributed memory object caching system
	Stream	Program to measure sustainable memory bandwidth
I/O-intensive	DBench	Tool to generate I/O workloads on file system
	Postmark	Test file system
Network-intensive	NetPerf	Test network performance
	Nuttcp	Latency and bandwidth test for TCP connections

5.2 Experimental validation

5.2.1 *Monitoring cost*

We select a managing node as the representative and record its consumption of resources with different numbers of monitored nodes from zero to ten, where zero means that the managing node only monitors itself.

Table 3 shows the resource usage of a management node. CPU utilization increases from 0.12% to 0.22% as the number of connected nodes increases. Memory utilization increases as the number of connected nodes increases. Nevertheless, the step size for the increase is small, and memory utilization is still under 0.4% when the number of connected nodes is ten. The number of sectors written to disk in each second is less than 2 (the size of a sector is 512 bytes). The number of packets received in each second is less than 2, and the amount of data received is less than 2 kB in each second. Table 3 demonstrates that

Table 3 Resource cost by monitoring process

Resource type	0 nodes	1 node	2 nodes	4 nodes	6 nodes	8 nodes	10 nodes
CPU utilization (%)	0.1498	0.1198	0.1999	0.1693	0.2694	0.2398	0.2200
Memory utilization (%)	0.0449	0.0579	0.0850	0.1499	0.2223	0.2972	0.3315
wrsec/s	0.1261	1.0536	2.0980	1.0989	1.1503	1.1673	1.0865
rcpackets/s	0.2899	1.0300	1.9597	1.4400	1.9099	1.8397	1.5395
rxkb/s	0.1261	1.0536	1.0980	1.0989	1.1503	1.1673	1.0865

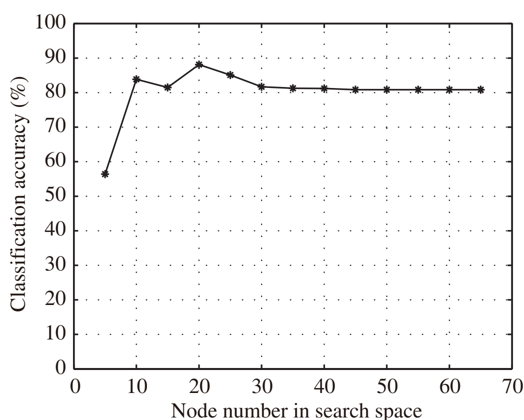

Figure 6 Classification accuracy with different features.

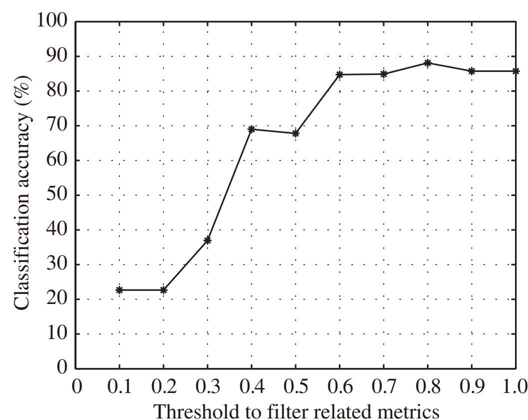

Figure 7 Accuracy with different thresholds.

vSpec monitoring has almost no extra resource cost.

5.2.2 *Feature selection*

To evaluate the algorithms used in feature selection, we separate the data matrix for the training set into two parts. The vectors in two-thirds of the data matrix are used as training vectors, and the others are used as testing vectors. The classification accuracy of TSRSVM is used as a test of the feature selection algorithm (Subsection 4.2).

In the first experiment, we vary the number of features selected by the maximum mutual information algorithm (Algorithm 2 in Subsection 4.2) from 1 to 65 (step size is 1) to compare the classification accuracy. Figure 6 plots the classification result. We can see that the highest classification accuracy is when 20 of 65 metrics with the largest mutual information are chosen.

To filter the correlated metrics, we set the initial metric set as the 20 metrics selected from the first experiment, and we change the filter threshold from 0 to 1 (step size is 0.1) to compare the classification accuracy of TSRSVM (Algorithm 1) in the second experiment. Figure 7 plots the clustering accuracy with different thresholds. We can see that the highest clustering accuracy for TSRSVM is when the threshold is 0.8. The 20 metrics selected from experiments are reduced to 18 when we set the threshold at 0.8 in the PCC filtering algorithm. Table 4 lists the selected 18 metrics.

5.2.3 *Workload classification*

To evaluate the workload classification of the algorithms, we select the workloads in Table 1 as the training set to train the classifier and use the workloads in Table 2 as the testing set to test whether the classifier works well. Four classification algorithms are compared in our experiments: K-nearest neighbors (KNN), logistic regression (LR), SVM and TSRSVM. Figure 8 plots the classification accuracy of the four algorithms. It shows that TSRSVM has a higher classification accuracy for the testing data for the CPU-intensive, memory-intensive, I/O-intensive and network-intensive classes. TSRSVM refreshes the training set after the OS of a truly classified workload is specialized. Hence, it takes more time to finish the classification. Figure 9 plots the time cost for the four classification algorithms. It shows that

Table 4 Selected features

Number	Related metrics	Description
1	%CPU_system	CPU utilization by system calls
2	%iowait	Percentage of time that the CPU is idle and waiting for a disk I/O request
3	%idle	Percentage of time that the CPU is idle and there is no outstanding disk I/O request
4	proc/s	Total number of tasks created per second
5	tps	Number of transfers that were issued to physical devices in each second
6	kbmemfree	Amount of free memory in kilobytes
7	kbmemused	Amount of used memory in kilobytes
8	kbbuffers	Amount of memory used as buffer by the kernel in kilobytes
9	kbcached	Amount of memory used as cache by the kernel in kilobytes
10	kbcommit	Amount of memory needed for current workload in kilobytes
11	kbswpfree	Amount of free swap space in kilobytes
12	kbswpused	Amount of used swap space in kilobytes
13	avgrq-sz	Average size (in sectors) of the requests that were issued to the device
14	wr_sec/s	Number of sectors written to the device (the size of a sector is 512 bytes)
15	await	Average time (in milliseconds) for I/O requests issued to the device to be served
16	rxpck/s	Total number of packets received per second
17	txpck/s	Total number of packets transferred per second
18	rxkB/s	Total number of kilobytes received per second

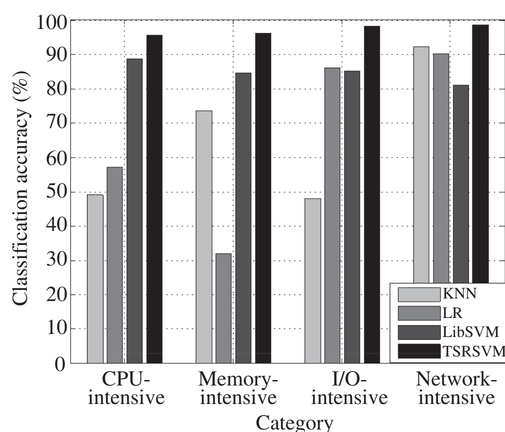

Figure 8 Classification accuracy.

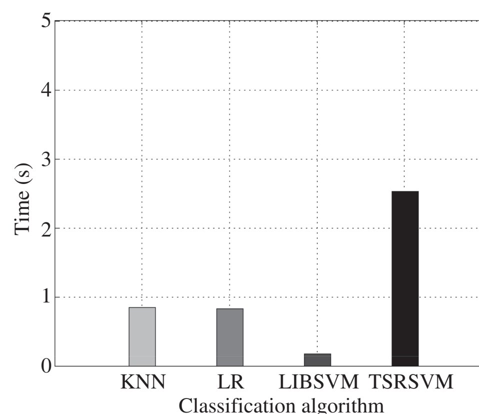

Figure 9 Time cost for classification.

TSRSVM costs the most; however, the cost is still less than 3 s, which is negligible since we run the classification once every 5 min.

Figure 10 plots the classification result for the TeraSort application on a Hadoop platform for each metric vector over 12 min (720 s). It shows that the category for the metric vectors changes over time. We label important inflections in the life cycle of TeraSort manually, such as the start and end time of the sample, map, compute and reduce phases. To illustrate why the classification changes, we compare the labeled phase with the changing trend of the classification result. We find that the trends are highly correlated. TeraSort tries to sample and map data in the first period (0 to 245 s), and the most prominent barrier is the read/write capability of HDFS during this period. Therefore, TeraSort is I/O-intensive in the first period (0 to 245 s). TeraSort sorts the data, which relies on its CPU-processing capability, during the map period (245 to 300 s). After 330 s, data are exchanged between nodes, meanwhile the job nodes have not finished the compute phase. The master node waits for data to be transferred from reduce nodes during this period (300 to 720 s), hence the demand on different resources is comparatively balanced. That is why after 330 s most of the metric vectors fall into the compound class.

We run the DFSIO application on a Hadoop platform, increasing the file size from 1 to 1000 MB

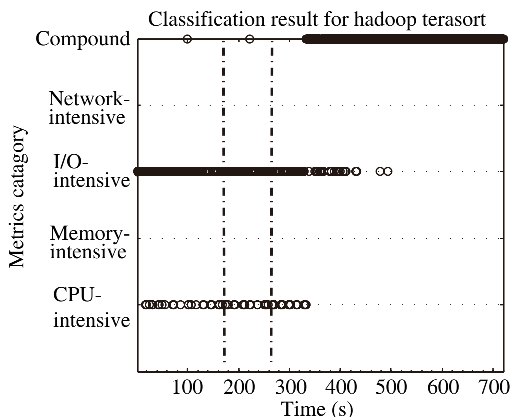

Figure 10 Classification result for TeraSort.

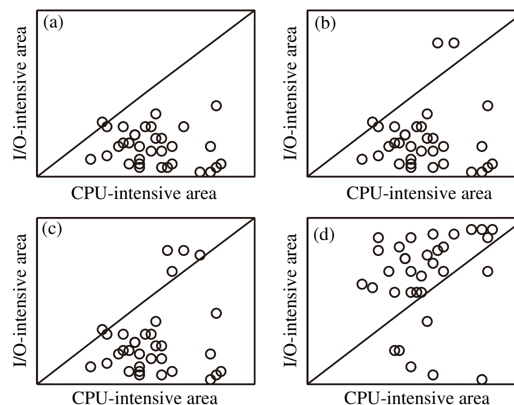

Figure 11 Classification result for DFSIO. (a) FileNumber=10000, FileSize=1 MB; (b) FileNumber=1000, FileSize=10 MB; (c) FileNumber=100, FileSize=100 MB; (d) FileNumber=10, FileSize=1000 MB.

while decreasing the number of files from 10000 to 10 to keep the total size of the files written to HDFS unchanged. Figure 11 plots the classification result for DFSIO. It shows that metric vectors mostly fall into the CPU-intensive class when the file size is 1 MB. More vectors move into the I/O-intensive category as the file size increases, and the class for DFSIO changes to I/O-intensive as the file size increases to 1000 MB. DFSIO mostly focuses on accepting and processing incoming files when the number of files is large, which puts much more pressure on the CPU. As the size of files increases, the VM has to focus more on writing data, hence the bottleneck is transferred to I/O.

Figures 10 and 11 show the classification results for TSRSVM on dynamically changing workloads. They demonstrate the effectiveness of TSRSVM for real-world applications in cloud computing.

5.2.4 *Performance improvement*

We evaluate the influence of vSpec by comparing the performance of applications in pre-specialization and post-specialization situations. We select three real-world applications: BlogBench, IPTV and JtangTest. For BlogBench, the proportion of metrics that fall into the I/O-intensive class is 87% during the first 5 min and 91% during the following 5 min. For IPTV, the proportion of metrics that fall into the network-intensive class is 93% during the first 5 min and 93.33% during the following 5 min. For JtangTest, the proportion of metrics that fall into the memory-intensive class is 82% during the first 5 min and 86% during the following 5 min. Therefore, TSRSVM classifies BlogBench, IPTV and JtangTest as I/O-intensive, network-intensive and memory-intensive, respectively.

We specialize the OS of the VMs that host these three applications with specializing programs. Figures 12, 13, 14 compare the performance of the original OS and the specialized OS. Figure 12 shows the change in the average number of transactions per second (tps) for BlogBench against time, showing that tps improves by 30%. Figure 13 shows the average time to process all the transactions in ten different tests, which is reduced by 80%. Figure 14 shows that the time to complete the insertion of jobs is increased by 10% as the number of lines of source data increases to 600.

6 Conclusion and future work

In this paper, we propose a new adaptive model for cloud computing and explore the design of a prototype named vSpec, which selects representative resource-consuming metrics to characterize workloads. It classifies the workloads as CPU-intensive, memory-intensive, I/O-intensive, network-intensive or compound. vSpec specializes the OS of the VM according to the class of the workloads running on the VM, which enables the OS to utilize the most abundant virtualized resource. The feature selection algorithm,

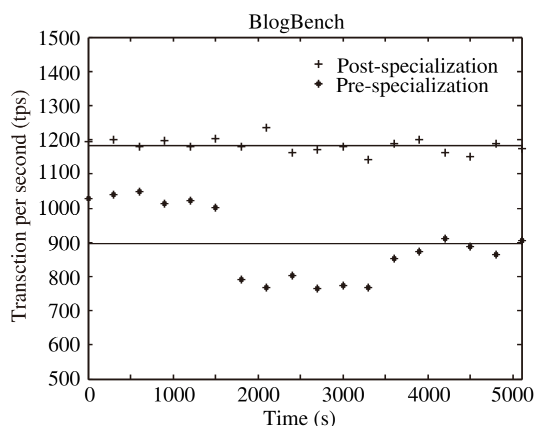

Figure 12 BlogBench.

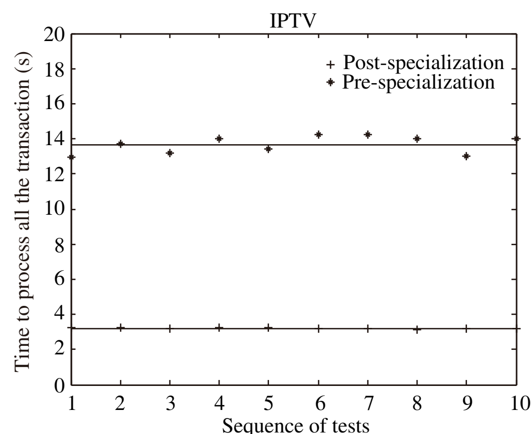

Figure 13 IPTV.

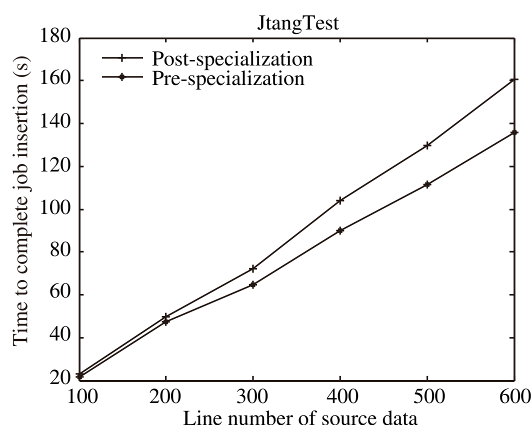

Figure 14 JtangTest.

workload classification algorithm, decision-making solution and specializing strategy are demonstrated to be effective for 33 benchmarks and three real applications.

This paper focused primarily on the workload-adaptive OS specialization for cloud computing. So far we have tested only static classification solutions. In future work, we will explore intelligent algorithms to predict the workload class in subsequent periods to assist the decision-making. We will also explore a common OS structure to support run-time specialization, such as [19, 21, 26, 27], and we will design an OS kernel customization according to the bottleneck type, rather than a static configuration.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 61272129), National High-Tech Research Program of China (Grant No. 2013AA01A213), New Century Excellent Talents Program of the Ministry of Education of China (Grant No. NCET-12-0491), Zhejiang Provincial Natural Science Foundation of China (Grant No. LR13F020002) and Science and Technology Program of Zhejiang Province (Grant No. 2012C01037-1).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Bhatia S, Consel C, Le Meur A, et al. Automatic specialization of protocol stacks in operating system kernels. In: Proceedings of 29th Annual IEEE International Conference on Local Computer Networks, Florida, 2004. 152–159

2 Gonina E, Kannan A, Shafer J, et al. Fay: extensible distributed tracing from kernels to clusters. In: Proceedings of ACM 23rd ACM Symposium on Operating Systems Principles, Cascais, 2011. 5–20

3 Makris K, Ryu K D. Dynamic and adaptive updates of non-quiescent subsystems in commodity operating system kernels. ACM SIGOPS Operat Syst Rev, 2007, 41: 327–340

4 Zhang Y, Bhargava B. Self-learning disk scheduling. IEEE Trans Knowl Data Eng, 2009, 21: 50–65

5 Anderson T E. The case for application-specific operating systems. In: Proceedings of the 3rd Workshop on Workstation Operating Systems, Key Biscayne, 1992. 92–94

6 Butrico M, Da Silva D, Krieger O, et al. Specialized execution environments. ACM SIGOPS Operat Syst Rev, 2008, 42: 106–107

7 Peter S, Li J, Zhang I, et al. Arrakis: the operating system is the control plane. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, 2014. 1–16

8 Hu L, Schwan K, Gulati A, et al. Net-cohort: detecting and managing vm ensembles in virtualized data centers. In: Proceedings of the 9th ACM International Conference on Autonomic Computing, San Jose, 2012. 3–12

9 Mishra A K, Hellerstein J L, Cirne W, et al. Towards characterizing cloud backend workloads: insights from google compute clusters. ACM SIGMETRICS Performa Eval Rev, 2010, 37: 34–41

10 Lin Y D, Lu C N, Lai Y C, et al. Application classification using packet size distribution and port association. J Netw Comput Appl, 2009, 32: 1023–1030

11 Zander S, Nguyen T, Armitage G. Automated traffic classification and application identification using machine learning. In: Proceedings of 30th IEEE Conference on Local Computer Networks, Sydney, 2005. 250–257

12 Karagiannis T, Papagiannaki K, Faloutsos M. BLINC: multilevel traffic classification in the dark. ACM SIGCOMM Comput Commun Rev, 2005, 35: 229–240

13 Rao J, Bu X, Xu C Z, et al. VCONF: a reinforcement learning approach to virtual machines auto-configuration. In: Proceedings of the 6th ACM international conference on Autonomic computing, Barcelona, 2009. 137–146

14 Jiang H, Moore A W, Ge Z, et al. Lightweight application classification for network management. In: Proceedings of the ACM SIGCOMM Workshop on Internet Network Management, Kyoto, 2007. 299–304

15 Zhang J, Figueiredo R J. Application classification through monitoring and learning of resource consumption patterns. In: Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium, Rhodes Island, 2006. 10–19

16 Zhang J, Figueiredo R J. Autonomic feature selection for application classification. In: Proceedings of IEEE International Conference on Autonomic Computing, Dublin, 2006. 43–52

17 Zhao X, Yin J, Chen Z, et al. Workload classification model for specializing virtual machine operating system. In: Proceedings of 6th IEEE International Conference on Cloud Computing (CLOUD), Santa Clara, 2013. 343–350

18 Consel C, Hornof L, Marlet R, et al. Tempo: specializing systems applications and beyond. ACM Comput Surv (CSUR), 1998, 30: 19

19 Engler D R, Kaashoek M F. Exokernel: an operating system architecture for application-level resource management. ACM SIGOPS Operat Syst Rev, 1995, 29: 251–266

20 Bershad B N, Chambers C, Eggers S, et al. SPIN—an extensible microkernel for application-specific operating system services. ACM SIGOPS Operat Syst Rev, 1995, 29: 74–77

21 Schatzberg D, Cadden J, Krieger O, et al. A way forward: enabling operating system innovation in the cloud. In: Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing, Philadelphia, 2014. 4

22 Iturbe X, Benkrid K, Erdogan A T, et al. R3TOS: a reliable reconfigurable real-time operating system. In: Proceedings of 2010 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), California, 2010. 99–104

23 Hoffmann H, Maggio M, Santambrogio M D, et al. Seec: A General and Extensible Framework for Self-Aware Computing. Technical Report MIT-CSAIL-TR-2011-046. 2011

24 Rossbach C J, Currey J, Silberstein M, et al. PTask: operating system abstractions to manage GPUs as compute devices. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, Cascais, 2011. 233–248

25 Panneerselvam S, Swift M M. Chameleon: operating system support for dynamic processors. ACM SIGPLAN Notices, 2012, 47: 99–110

26 Madhavapeddy A, Mortier R, Rotsos C, et al. Unikernels: library operating systems for the cloud. In: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, New York, 2013. 461–472

27 Kivity A, Laor D, Costa G, et al. OSv—optimizing the operating system for virtual machines. In: Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference, Berkeley, 2014. 61–72

28 Seltzer M, Small C. Self-monitoring and self-adapting operating systems. In: Proceedings of The Sixth Workshop on Hot Topics in Operating Systems, Cape Cod, 1997. 124–129

29 Lee C T, Lin J M, Hong Z W, et al. An application-oriented Linux kernel customization for embedded systems. J Inf Sci Eng, 2004, 20: 1093–1107

30 Saez J C, Prieto M, Fedorova A, et al. A comprehensive scheduler for asymmetric multicore systems. In: Proceedings of the 5th European Conference on Computer Systems, New York, 2009. 139–152

31 McNamee D, Walpole J, Pu C, et al. Specialization tools and techniques for systematic optimization of system software. ACM Trans Comput Syst, 2001, 19: 217–251

32 Soules C A N, Appavoo J, Hui K, et al. System support for online reconfiguration. In: Proceedings of USENIX Annual Technical Conference, General Track, San Antonio, 2003. 141–154

33 Oberthür S, Böke C, Griese B. Dynamic online reconfiguration for customizable and self-optimizing operating systems. In: Proceedings of the 5th ACM International Conference on Embedded Software, New Jersey, 2005. 335–338

34 Soror A A, Minhas U F, Aboulnaga A, et al. Automatic virtual machine configuration for database workloads. ACM Trans Database Syst (TODS), 2010, 35: 1–47

35 Pu C, Autrey T, Black A, et al. Optimistic incremental specialization: Streamlining a commercial operating system. ACM SIGOPS Operat Syst Rev, 1995, 29: 314–321

36 Burda R, Seger J. A tool framework for generation of application optimized communication protocols. In: Proceedings of the 3rd Annual Communication Networks and Services Research Conference, Halifax, 2005. 282–286

37 Bhatia S, Consel C, Le Meur A, et al. Automatic specialization of protocol stacks in operating system kernels. In: Proceedings of 29th Annual IEEE International Conference on Local Computer Networks, Florida, 2004. 152–159

38 Marinos I, Watson R N M, Handley M. Network stack specialization for performance. In: Proceedings of the 2014 ACM Conference on SIGCOMM, Chicago, 2014. 175–186

39 Liu L B, Jia W, Yin S Y, et al. ReSSIM: a mixed-level simulator for dynamic coarse-grained reconfigurable processor. Sci China Inf Sci, 2013, 56: 062402

40 Wang Y S, Liu L B, Yin S Y, et al. Hierarchical representation of on-chip context to reduce reconfiguration time and implementation area for coarse-grained reconfigurable architecture. Sci China Inf Sci, 2013, 56: 112401

41 Domeniconi C, Gunopulos D. Incremental support vector machine construction. In: Proceedings IEEE International Conference on Data Mining, San Jose, 2001. 589–592

42 Ciliendo E, Kunimasa T. Linux Performance and Tuning Guidelines. San Jose: IBM International Technical Support Organization, 2007. 77–135

43 Vavilapalli V K, Murthy A C, Douglas C, et al. Apache Hadoop Yarn: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing, New York, 2013. 5:1–5:16

44 Cao B, Yin J, Deng S, et al. A highly efficient cloud-based architecture for large-scale STB event processing: industry article. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, New York, 2012. 314–323