An in-depth analysis of system-level techniques for Simultaneous Multi-threaded Processors in Clouds

Yaohua Wang, Rongze Li, Zhentao Huang, Xu Zhou

An In-depth Analysis of System-level Techniques for Simultaneous Multi-threaded Processors in Clouds

Yaohua Wang
National University of Defense Technology
Changsha, China
yhwang@nudt.edu.cn

Rongze Li
University of Nottingham
Ningbo, China
scyrl1@nottingham.edu.cn

Zhentao Huang
University of Nottingham
Ningbo, China
scyzh3@nottingham.edu.cn

Xu Zhou *
National University of Defense Technology
Changsha, China
zhouxu@nudt.edu.cn

ABSTRACT
To improve the overall system utilization, Simultaneous Multi-Threading (SMT) has become a norm in clouds. Usually, Hardware threads are viewed and deployed directly as physical cores for attempts to improve resource utilization and system throughput. However, context switches in virtualized systems might incur severe resource waste, which further led to significant performance degradation. Worse, virtualized systems suffer from performance variations since the rescheduled vCPU may affect other hardware threads on the same physical core. In this paper, we perform an in-depth experimental study about how existing system software techniques improves the utilization of SMT Processors in Clouds. Considering the default Linux hypervisor *vanilla KVM* as the baseline, we evaluated two update-to-date kernel patches *IdlePoll* and *HaltPoll* through the combination of 14 real-world workloads. Our results show that mitigating they could significantly mitigate the number of context switches, which further improves the overall system throughput and decreases its latency. Based on our findings, we summarize key lessons from the previous wisdom and then discuss promising directions to be explored in the future.

CCS Concepts
• **Software and its engineering~Software organization and properties~Software system structures~Distributed systems organizing principles~Cloud computing**

Keywords
Simultaneous Multi-threading; Operating Systems; Hypervisor.

1. INTRODUCTION
In the era of cloud computing, Simultaneous Multi-Threading (SMT) has been widely enabled to improve resource utilization and system throughput [2, 8, 16, 23]. Out of its nature, enabling SMT could allow multiple hardware threads to share one physical core simultaneously, and the number of sharing threads are dependent to the levels of resource partitions.

For better levels of resource utilization and energy consumption in Clouds, multiple Virtual Machines (VMs) are often consolidated on a single physical host, and multiple Virtual CPUs (vCPUs) often time-share hardware threads. Hence, when one vCPU is idling/busy-waiting or its time slice uses up, Virtual Machine Management (VMM) deschedules the vCPU and re-schedule another vCPU on this hardware thread to utilize system resources.

However, the frequent context switches and its accompanying high overheads in Clouds have caused a huge performance gap, which could be elaborated in two the following two aspects. Firstly, the frequent context switches are caused by the intention of better system resource utilization, due to the nature of VMM. Secondly, in virtualized systems, the cost of context switches could be at least 5.6X than those in physical machines. And this has wasted system resources and thus incurs significant performance degradation.

Towards the above performance challenges, several preliminary system-level schemes have been proposed to mitigate such issues. Taking Linux hypervisor *vanilla KVM* as an example, there are two outstanding approaches to address and tackle these issues. One is called *IdlePoll*, which prolongs the staying periods of idling threads in its place, without being descheduled once idling [12]. The other is called *HaltPoll*, which only keeps idling/busy-waiting vCPUs for a shorter period than *IdlePoll* [11].

To better understand and obtain in-depth insights about the pros and cons of those techniques, an in-depth and comprehensive experimental study is needed in general. Hence, we performed such a study through a rigorous and comprehensive study step-by-step. We first implement the above two techniques through kernel-based patches, by using *vanilla KVM* as the baseline. Then we select 14 real-world workloads to examine their benefits and issues, through a comprehensive evaluations and analysis.

Our results present quantitative envidence that both *IdlePoll* and *HaltPoll* could improve the throughput up and reduce the energy consumption significantly. We also present several breakdown analysis to explore other characteristics while deploying the above approaches, in order to explore and vision the future of SMT in the era of Cloud Computing

This paper are organized as follow. We present relevant background information in Section 2. Then we illustrate details regarding two key patches *IdlePoll* and *HaltPoll* and detailed setup

The corresponding author is Xu Zhou

Conference'10, Month 1–2, 2010, City, State, Country.
Copyright 2010 ACM 1-58113-000-0/00/0010 …$15.00.
DOI: http://dx.doi.org/10.1145/12345.67890

of our experimental study in Section 3. Next, we present the results and analysis in Section 5. Finally, we discuss potential future work and obtain key conclusions in Section 6 and 7 respectively.

2. BACKGOUND & MOTIVATION

In this section, we explore background and motivation of our study. First, we briefly explore Simultaneous Multi-threading (SMT) in Section 2.1. Then, we describes key characteristics of Synchronization in Virtualized Systems in Section 2.2. Finally, we end up with a lightweight experiment to support our hypothesis on context switches in Section 2.3, which has substantially motivated our study.

2.1 Simultaneous Multi-threading

Nowadays modern processors utilize Simultaneous Multi-threading (SMT) to improve the level of resource utilization and increase the overall system throughput [13, 14, 18, 22]. SMT achieves such benefits by allowing physical cores to share multiple hardware threads within the same set of pipeline, function units, cache and so on. For better levels of resource occupancy and efficiency, each hardware thread maintains their own hardware contexts to fully utilize the shared resources properly. Though IBM provides 8-way SMT support, hereby we focus on Intel processors with 2-way SMT, which are usually referred as Hyper-threading.

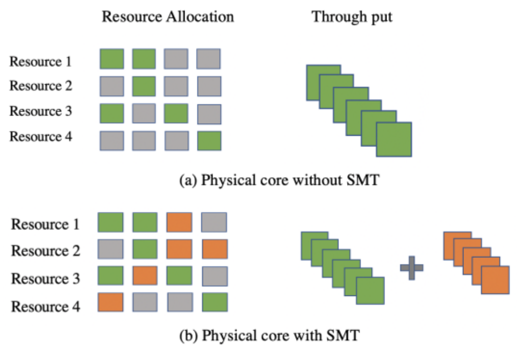

Figure 1. A Conceptual Comparison between SMT-enabled and SMT-disabled Cores.

Figure 1 gives out a conceputal comparison between cores with SMT-enabled and those without, which was the initial design motivation of such a promising technique. Figure 1 (a) shows that when there is only one hardware thread running on the physical core, resources such as pipeline, function units and cache are underutilized (white boxes). Figure 1 (b) shows two hardware threads sharing one SMT enabled physical core. The resource utilization and system throughput of SMT enabled physical core is higher since SMT can improve the resource utilization.

2.2 Synchronization within Virtual Machines

In virtualized systems, inter-vCPU synchronization incurs high overheads and significant variations in the context of applications' performance. More specifically, there are two basic primitives for inter-vCPU synchronization: 1) *Idling*. It refers to that one vCPU has been blocked blocking since its required resources are unavailable and yields its resources for executions into another ready vCPU; 2) *Busy-waiting*. It refers to that one vCPU has been spinning, in order to check whether its required resources are available so that it could continue to make progress. Particularly for busy-waiting, modern hardware mechanisms, like Intel Pause-Loop Exiting and AMD Pause Filter, could interrupt the *busy-waiting* vCPU to avoid system resources' wastes.

Whenever one vCPU has been either *idling* or *busy-waiting*, such vCPU would yield its resources for execution to another vCPU by performing a context switch. More accurately, such a context switch starts from Virtual Machine Exit and ends up at vCPU Resume. When one vCPU begins *idling* or *busy-waiting*, it traps into Virtual Machine Mangement layer and the scheduler within this layer would schedule another non-blocking vCPU to enters Virtual Machine by resuming its context to continue progressing.

2.3 High Costs of Context Switches

In order to get a scratch about the high costs of context switches, which were caused by its high volume and significant overheads, we illustrate its through microbench as our motivation for indepth-study among real-world workloads further. Hereby, we patched the system with *IdlePoll* and select *Parsec dedup (detnoted as p.dedup)*, *Splash2X volrend (denoted as s.volrend)* and *MatMuls (Matrix Multiplication)* as the workloads. We configure that 4 VMs are consolidated in our host OS (i.e. Ubuntu 14.04) with 48 logical cores, and each VM has 24 vCPUs and two vCPUs share one logical core in average. Particularly, there are two separate case studies to demonstrate the evidence of our concerns, which would be elaborated in more details later.

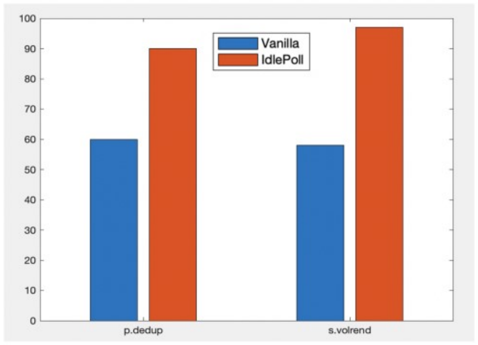

Figure 2. 1st part of results from Lightweight Experimental Study, where we address the low resource utilization from the high volume of avoidable context witches.

We first illustrate performance degradation by low resource utilization from the high volume of avoidable context switches. In this case, we deploy each VM to run the same program, which has been considered as *sychronization-intensive*. As Figure 2 shown, the *IdlePoll* patch has improved the throughput of *Parsec dedup* by around 53.6% and the throughput of *Splash2X volrend* by 80.8%, compared with the default setting in the *Vanilla KVM* hypervisor. Also, there are considerable performance degradation when *Vanilla KVM* is deployed, compared with no virtualization support. This is caused by extremely large resource wastes from unnecessary context switches, as illustrated in Section 2.2. And *IdlePoll* could mitigate such issues by prolonging the stay periods of vCPUs, no matter the workloads has been executed.

Then we describe performance variation issues, which has been caused also from the high volume of avoidable context switches.

As for this case, we distribute both *sychronization-intensive* and *computation-intensive* workloads into four different virtual machines. More specifically, VM1 and VM2 are responsible for *Pasrsec dedup* and *Splash volrend* respectively, and VM3 and VM4 both run *matmuls*. As Figure 3 shown, the significant performance variation has been caused and *IdlePoll* could mitigate such issue. Particularly, *IdlePoll* has maximazied the throughput since all avoidable context switches have been eliminated, which substantially remove the performance interference between two vCPUs on different logical cores from the same physical core.

However, the performance variation among four workloads are still considered large, where the throughput of *MatMuls* workloads have been lowered down. We believe this is because that both *computation-intensive* vCPU and *synchronization-intensive* vCPU are not always paired together on two different logical cores of the same physical core, since *IdlePoll* configuration will occupy the system resources with *MatMuls-related* vCPUs, even though it might be idling.

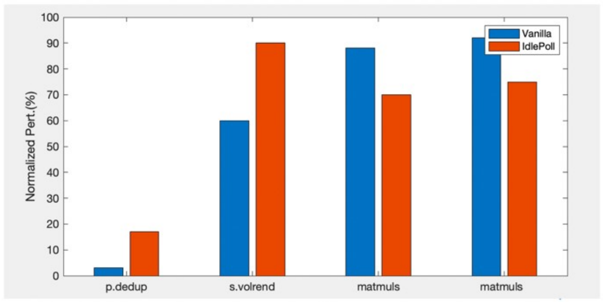

Figure 3. 2nd part of results from Lightweight Experimental Study, where we address the performance variations from the high costs of avoidable context witches.

For our experiments, 4 Virtual Machines are launched on the physical server. For each Virtual Machine, 24 vCPUs and 16 GB memory are allocated. The Virtual Machine Management has been decided as KVM, and both the host Operating System and guest Operating Systems are Ubuntu 14.04. The kernel versions of host OS and guest OS are both 3.16.39.

3. IN-DEPTH STUDY DESIGN

In this section, we present detailed strategies and designs for our in-depth experimental study. First, we took a revisit to all system-level designs principles and relevant mitigation techniques. Then, we introduce details about how we setup the experimental studies, in the context of both host and guest machines. Finally, we describe out experimental study methodology and details.

3.1 System-level Techniques Revisited

We have investigated all relevant strategies and we found there are three major techniques, which are *Default Vanilla KVM* setting, *IdlePoll* setting and *HaltPoll* setting. And we would elaborate their characteristics separately in the following.

Vanilla KVM is the internal virtualization module of Linux Operating System support, which has been abbreviated from *Kernel-based Virtual Machine*. *KVM* allows the kernel to function as the hypervisor. Both *IdlePoll* and *HaltPoll* are relevant kernel patches, as previous tryouts, to utilize SMT resources in a more efficient fashion. The main difference is that, *IdlePoll* keeps threads polling until required resources become available, but *HaltPoll* would keep them for a short period before necessary context switches.

3.2 Experimental Setup

In this section, we report the configurations about the machines, which have been used for our experimental study. All our evaluations are done on a DELL TM PowerEdge TM R430 server with 64GB of DRAM, and two 2.60GHz Intel Xeon E5-2690 processors. Each processor has 12 physical cores, and each physical core has two hardware threads.

3.3 Real-world Workloads

In order to examine the real-world effects, we have selected 14 real-world workloads and paired them properly. These 14 real-world workloads are: *Hadoop*, a popular big data processing framework; *XGBoost*, a widely used artificial intelligence library; *MySQL*, MySQL OLTP benchmark with SysBench; *Spark*, an open-source cluster-computing framework; *SSDB*, a widely used key value storage system; *PgSQL*, a widely used databased management system; *DBT1*, TPCW; *HDFS*, a distributed file system for big data; *ClamAV*, an anti-virus system; *Apache*, a widely used web server; *MediaTomb*, a popular media server for encoding and decoding videos; *FileServer*, a FileServer with FileBench storage benchmark; *PageRank*, a web searching algorithm; *WaterMark*, a watermark application based on lighttpd ; *MongoDB*, a document-oriented key-value store system.

Table 1 gives an outline of these applications. In general, one *synchronzaion-intensive* and one *computation-intensive* workloads are placed in two Virtual Machines respectively, and the other two follow the same configuration.

Application	Workload
Apache	Simultaneous requests on a web server w/ ApacheBen
ClamAV	Antivirus files with clamscan 【5】
DBT1	TPC-W web-based transactions performance test
FileServer	FileBench 【21】 with FileServer benchmark
Hadoop	TeraSort with Hadoop MapReduce 【10】 framework
HDFS	Sequential read 16GB with HDFS TestDFSIO 【9】
Lighttpd	Simultaneous requests using PhP to watermark images
MediaT	Simultaneous requests on transcoding a 1.1GB video
MongoDB	Sequential scan records with YCSB 【25】
MySQL	MySQL OLTP transactional with SysBench 【17】
PgSql	PgBench 【19】 with TPC-B benchmark
Spark	PageRank and Kmeans algorithms in Spark 【20】
SSDB	50% SET, 50% GET requests arriving in batches
XGBoost	4 AI algorithms in XGBoost 【24】 library

Table 1. Detailed Breakdown of 14 Real-world Applications

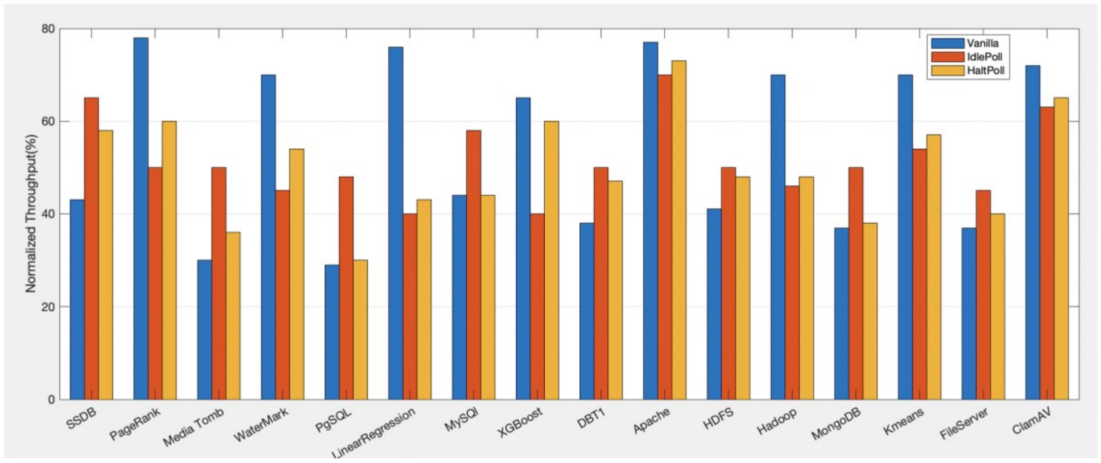

Figure 4. Throughput Plots of 14 Real-world Applications through 4 Virtual Machines; each VM has 24 vCPUs. VM1 runs synchronization-intensive application; VM2 runs computation-intensive application; VM3 and VM4 run the same application as VM1 and VM2 respectively.

4. RESULTS

In this section, we report results and implications from our experimental study. We first examine the throughput, then we break down the whole procedure with context switch statistics. Finally we discuss the energy consumption of these approaches.

4.1 Context Switches Breakdown

Table 2 shows that all *synchronization-intensive* applications, and details are as follow. *IdlePoll* incurs no context switches during the whole procedure, and *HaltPoll* could reduce the context switches significantly, compared with *Vanilla KVM*.

Application	*Vanilla*	*IdlePoll*	*HaltPoll*
DBT1	169329682	0	91744198
FileServer	91478963	0	20359352
HDFS	5135702	0	3269870
MediaTomb	394215	0	184126
MongoDB	12741002	0	9241032
MySQL	14412627	0	7254103
PgSql	1904564	0	1162871
SSDB	41116782	0	30576993

Table 2. Detailed Breakdown about Context Switches for Selected Real-world Applications.

4.2 Throughput Overview

As Figure 4 shown, both *IdlePoll* and *HaltPoll's* performance and performance variation on 14 applications are clearly presented. To showcase the effects properly, we have normalized all results bare-metal (i.e. without virtualization).

IdlePoll avoids unnecessary context switch by keeping vCPUs when they become idling. However, *IdlePoll* affects the performance of co-running *computation-intensive* vCPUs on the same physical cores. For instance, *MediaTomb*, *DBT1* and *MongoDB* could be highlighted since these applications usually combine I/O and compute operations in each task. Besides, *IdlePoll* does not pair *synchronization-intensive* vCPUs and *computation intensive* vCPUs on the same physical cores to improve performance.

As a comparison, Figure 4 also shows *HaltPoll*'s throughput on computation intensive applications is much higher than *IdlePoll* on average. *HaltPoll* only keeps vCPUs active for a short period to see whether the contended resources are available. 1), *HaltPoll* still incurs much unnecessary context switch. 2), *HaltPoll* does not schedule *synchronization-intensive* and *computation-intensive* vCPUs on the same physical cores to improve performance.

4.3 Energy Savings

Figure 5 presents the energy savings through *IdlePoll* and *HaltPoll* through *kembench*, *Exim* and *Metis* benchmarks to evaluate. As the results shown, *IdlePoll* and *HaltPoll* could significantly reduce the energy consumption. However, *HaltPoll* couldn't save more, which indicates that it's not necessary to use *IdlePoll* (i.e. keeping all vCPUs without context switches).

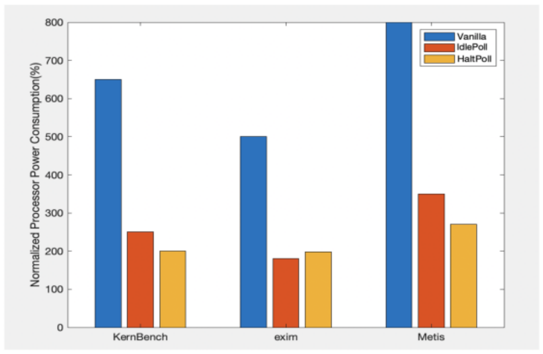

Figure 5. Energy Consumption Plots of Selected

5. DISCUSSIONS

Through our in-depth experimental study, we have examined the effectiveness of existing system-level approaches to mitigate performance issues. Hence, we observe that the need of resource-aware scheduling policies and detailed workload characterizations.

For resource-aware scheduling, we could enable schemes like Symbiotic Jobscheduling to co-locate *sychronization-intensive* threads and *computation-intensive* threads on the same core [4].

As for detailed workload characterizations, we could enable in-time schedule changes, since workloads usually have different features in different stages with their progress [7].

6. CONCLUSIONS

In this paper, we performed an in-depth experimental study of system-level techniques for Simultaneous Multi-threaded Processors in Clouds. After investigations, we have chosen two most recent techniques *IdlePoll* and *HaltPoll*, and then perform an experimental study among 14 real-world workloads. The results show that these two techniques could significantly improve the throughput and reduce energy consumption, by benefiting from large reduction of context switches.

7. ACKNOWLEDGMENTS

We thank for valuable suggestions and feedbacks from HP3C'20. This research is supported by ported by The Science and Technology Planning Project of Hunan Province (2019RS2027).

8. REFERENCES

[1] Adding watermarks to images. http://php.net/manual/en/image.examples-watermark.php.

[2] Amazon EC2 T-series instances. https://aws.amazon.com/ec2/instance-types/t2/.

[3] Apache Server. http://www.apache.org.

[4] J. R. Bulpin and I. Pratt. Hyper-threading aware process scheduling heuristics. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, General Track, pages 399 – 402, 2005.

[5] ClamAV. http://www.clamav.net/.

[6] DBT1. http://osdldbt.sourceforge.net/.

[7] A. Fedorova, M. Seltzer, C. Small, and D. Nussbaum. Performance of multithreaded chip multiprocessors and implications for operating system design. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 26 – 26. USENIX Association, 2005.

[8] Google Cloud F1 and G1 Instances. https://cloud.google.com/compute/docs/machine-types.

[9] Hadoop Core System. http://hadoop.apache.org/core/

[10] Hadoop Distributed File System. http://hadoop.apache.org/hdfs/.

[11] HaltPoll Kernel Patch. https://lkml.org/lkml/2017/6/22/296.

[12] IdlePoll Kernel Patch. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=aca6ff29c4063a8d467cdee241e6b3bf7dc4a171

[13] IBM POWER9 Processor. https://en.wikipedia.org/wiki/POWER9.

[14] Intel R64 and ia-32 architectures developer's manual. https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html.

[15] MediaTomb - Free MediaServer. http://mediatomb.cc/.

[16] Microsoft Azure B-series instances. https://azure.microsoft.com/en-us/blog/introducing-b-series-our-new-burstable-vm-size/.

[17] MySQL Database. http://www.mysql.com/.

[18] Oracle SPARC M8 Processor. http://www.oracle.com/us/products/servers-storage/sparc-m8-processor-ds-3864282.pdf..

[19] PostgreSQL Database. https://www.postgresql.org.

[20] Spark PageRank benchmark. https://github.com/apache/spark/blob/master/examples/src/main/java/org/apache/spark/examples/JavaPageRank.java.

[21] V. Tarasov, E. Zadok, and S. Shepler. Filebench: A flexible framework for file system benchmarking. USENIX; login, 41, 2016.

[22] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings. Of IEEE 22nd Annual International Symposium on Computer Architecture*, pages 392–403. 1995.

[23] VMware. Vmware horizon view architecture planning 6.0. In VMware Technical White Paper, 2014.

[24] XGBoost. http://dmlc.cs.washington.edu/xgboost.html.

[25] Yahoo! Cloud Serving Benchmark. https://github.com/brianfrankcooper/YCSB.