vSMT-IO: Improving I/O Performance and Efficiency on SMT Processors in Virtualized Clouds

Weiwei Jia, Jianchen Shan, Tsz On Li, Xiaowei Shang, Heming Cui, Xiaoning Ding

New Jersey Institute of Technology, Hofstra University, Hong Kong University

SMT is widely enabled in clouds

- Most types of virtual machines (VMs) in public clouds run on processors with SMT (Simultaneous Multi-Threading) enabled.
  - A hardware thread may be dedicatedly used by a virtual CPU (vCPU).
  - It may also be time-shared by multiple vCPUs.

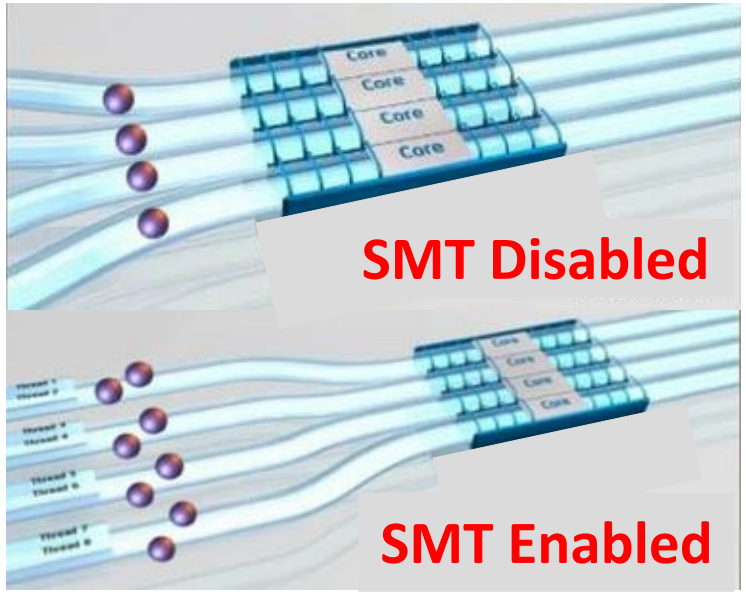


Figure from internet

- Enabling SMT can improve system throughput.
  - Multiple hardware threads (HTs) share the hardware resources on each core.
  - Hardware resource utilization is increased.

CPU scheduler is crucial for SMT processors

- To achieve high throughput, CPU scheduler must be optimized to maximize CPU utilization and minimize overhead.

- Extensively studied: symbiotic scheduling focuses on maximizing utilization for computation intensive workloads (SOS[ASPLOS'00], cycle accounting[ASPLOS'09, HPCA'16], ...).
  - Co-schedule on the same core the threads with high symbiosis levels.
    - Symbiosis level: how well threads can fully utilize hardware resources with minimal conflicts.

- Under-studied: Scheduling I/O workloads with low overhead on SMT processors.
  - I/O workloads incur high scheduling overhead due to frequent I/O operations.
  - The overhead reduces throughput when there are computation workloads on the same SMT core.

Outline

✓Problem: efficiently schedule I/O workloads on SMT CPUs in virtualized clouds

- vSMT-IO
  - Basic Idea: make I/O workloads "dormant" on hardware threads
  - Key issues and solutions
- Evaluation
  - KVM-based prototype implementation is tested with real world applications
  - Increases the throughput of both I/O workload (up to 88.3%) and computation workload (up to 123.1%)

I/O workloads are mixed with computation workloads in clouds

- I/O applications and computation applications are usually consolidated on the same server to improve system utilization.
- Even in the same application (e.g., a database server), some threads are computation intensive, and some other threads are I/O intensive.

- The scheduling of I/O workloads affects both I/O and computation workloads.
  - High I/O throughput is not the only requirement.
  - High I/O efficiency (low overhead) is equally important to avoid degrading throughput of computation workloads.

Existing I/O-Improving techniques are inefficient on SMT processors

- To improve I/O performance, CPU scheduler increases the responsiveness of I/O workloads to I/O events.
  - Common pattern in I/O workloads: waiting for I/O events, responding and processing them, and generating new I/O requests.
  - Respond to I/O events quickly to keep I/O device busy.

- Existing techniques in CPU scheduler for increasing I/O responsiveness
  - *Polling* (Jisoo Yang et. al. [FAST'2012]): I/O workloads enter busy loops while waiting for I/O events.
  - *Priority boosting* (xBalloon [SoCC'2017]): Prioritize I/O workloads to preempt running workloads.
  - Incur busy-looping and context switches and reduce resources available to other hardware threads.

Polling and *priority boosting* incur higher overhead in virtualized clouds

- Polling on one hardware thread slows down the computation on the other hardware thread by about 30%.
  - Execute repeatedly instructions controlling busy-loops.
  - Incur costly VM_EXITs because polling is implemented at the host level.

- Switching vCPUs incurred by priority boosting on one hardware thread may slow down the computation on the other hardware thread by about 70%.
  - Save and restore contexts
  - Execute of scheduling algorithm
  - Flush L1 data cache for security reasons
  - Handle rescheduling inter-processor interrupts (IPIs)

Outline

- Problem: efficiently schedule I/O workload on SMT CPUs in virtualized clouds

✓vSMT-IO

  - Basic Idea: make I/O workloads "dormant" on hardware threads
  - Key issues and solutions

- Evaluation
  - KVM-based prototype implementation is tested with real world applications
  - Increases the throughput of both I/O workload (up to 88.3%) and computation workload (up to 123.1%)

Basic idea: make I/O workloads "dormant" on hardware threads

- Motivated by the hardware design in SMT processors for efficient blocking synchronization (D.M. Tullsen et. al. [HPCA'1999]).
- Key technique: **Context Retention**, an efficient blocking mechanism for vCPUs.
  - A vCPU can "block" on a hardware thread and release all its resources while waiting for an I/O event (no busy-looping).
    - High efficiency: other hardware threads can get extra resources.
  - The vCPU can be quickly "unblocked" without context switches upon the I/O event.
    - High I/O performance: I/O workload can quickly resume execution.
    - High efficiency: no context switches involved.
  - Implemented with MONITOR/MWAIT support on Intel CPUs.

Issue #1: uncontrolled context retention can diminish the benefits from SMT

- Context retention reduces the number of active hardware threads on a core.
  - On x86 CPUs, only one hardware thread remains active, when the other retains context.
  - Delay the execution of computation workloads or other I/O workloads on the core.
- Uncontrolled context retention may be long time periods.
  - Some I/O operations have very long latencies (e.g., HDD seeks, queuing/scheduling delays).

- ***Solution***: enforce an adjustable timeout on context retentions.
  - Timeout interrupts context retentions before they become overlong.
  - Timeout value being too low or too high reduces both I/O performance and computation performance.
    - Value too low: context retention is ineffective (low I/O performance); high overhead from context switches (low computation performance).
    - The timeout value is adjusted dynamically (algorithm shown on next page).

Start from a relatively low value

Gradually adjust timeout value

.
.
.

If new value can improve both I/O and computation performance

Algorithm 1 Context Retention Timeout Adjustment

1: T_d: desired timeout value; T_e: effective timeout value; T_{init}: initial timeout value; P: time period between two adjustments
2: $T_d \leftarrow T_{init}$
3: **while** true **do**
4: $T_e \leftarrow T_d$, collect performance data for a time period of P
5: **if** TESTTIMEOUT(T_d * 1.1) **then**
6: $T_d \leftarrow T_d * 1.1$; continue
7: **else**
8: $T_e \leftarrow T_d$, collect performance data for a time period of P
9: **end if**
10: **if** TESTTIMEOUT(T_d * 0.9) **then**
11: $T_d \leftarrow T_d * 0.9$; continue
12: **end if**
13: **end while**
14: **function** TESTTIMEOUT(T)
15: $T_e \leftarrow T$, collect performance data for a time period of P
16: $S_{cpu} \leftarrow$ average speed-up of CPU-bound vCPUs
17: $S_{io} \leftarrow$ average speed-up of I/O bound vCPUs
18: **if** $S_{cpu} > 1$ and $S_{io} > 1$ **then return** true; **end if**
19: **return** false
20: **end function**

Issue #2: existing symbiotic scheduling techniques cannot handle mixed workloads

- To maximize throughput, scheduler must co-schedule workloads with complementary resource demand.
- The resource demand of I/O workloads change dramatically due to context retention and burstiness of I/O operations.
- Existing symbiotic scheduling techniques target steady computation workloads and precisely characterize resource demand.

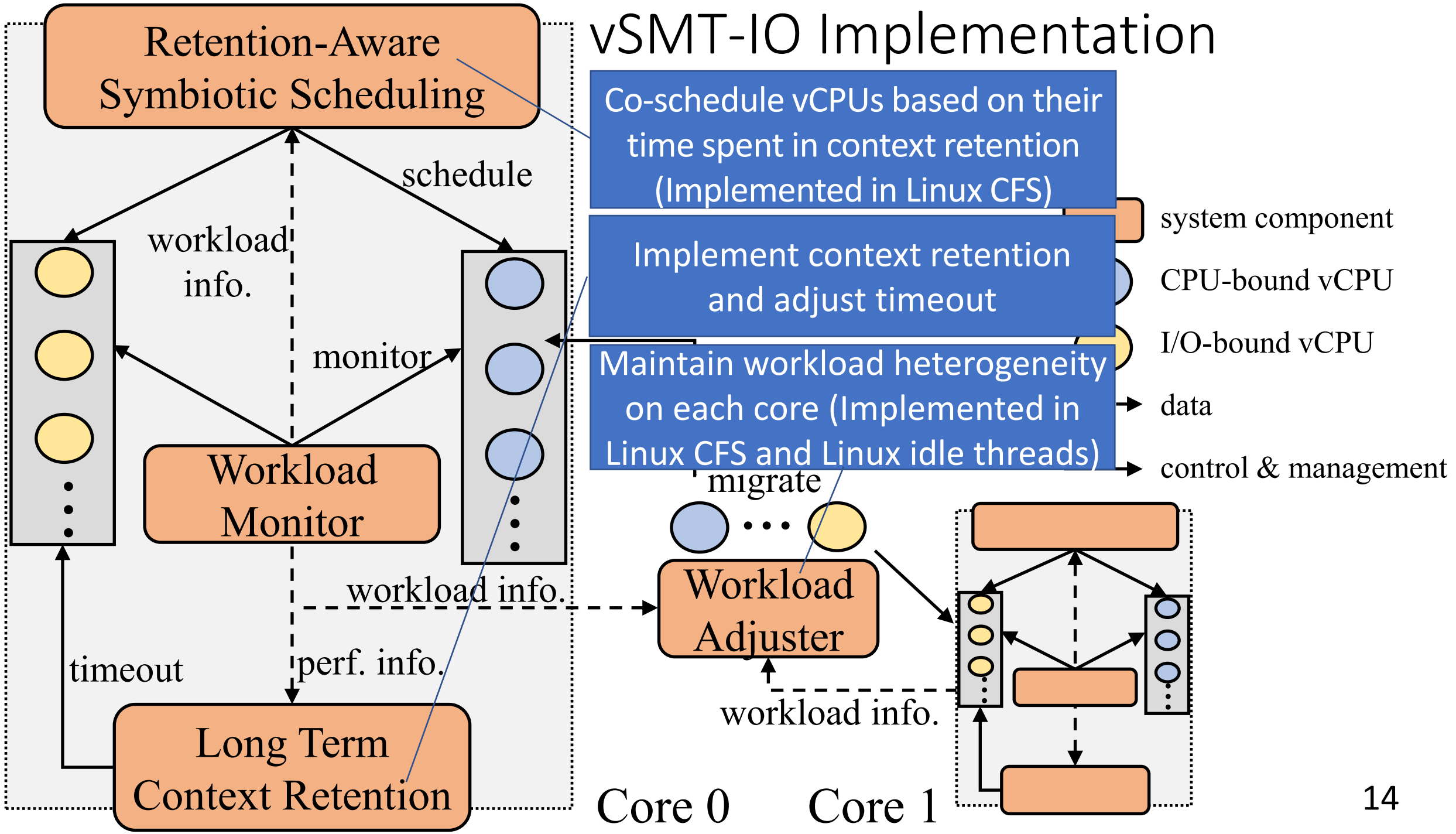
- ***Solution***: target dynamic and mixed workloads and coarsely characterize resource demand based on the time spent in context retention.
  - Rank and then categorize vCPUs based on the amount of time they spend on context retention.
    - Category #1: ***Low retention*** --- vCPUs with less context retention time are resource-hungry.
    - Category #2: ***High retention*** --- vCPUs with more context retention time consume little resource.
  - vCPUs from different categories have complementary resource demand and are co-scheduled on different hardware threads.
  - A conventional symbiotic scheduling technique is used only when all the ``runnable" vCPUs are in low retention category.

Other issues

- Issue #3: context retention may reduce the throughput of I/O workloads since it reduces the timeslice available for their computation.
- ***Solution:***
  - Timeouts (explained earlier) help reduce the timeslice consumed by long context retentions.
  - Compensate I/O workloads by increasing their weights/priorities.

- Issue #4: the effectiveness of vSTM-IO reduces when the workloads become homogeneous on each core.
- ***Solution***:
  - Migrate workloads across different cores to increase the workload heterogeneity on each core.
    - Workloads on different cores may still be heterogeneous.
      - E.g., computation workloads on one core, and I/O workloads on another core.

vSMT-IO Implementation



Outline

- Problem: efficiently schedule I/O workload on SMT CPUs in virtualized clouds
- vSMT-IO
  - Basic Idea: make I/O workloads "dormant" on hardware threads
  - Key issues and solutions

✓Evaluation

  - KVM-based prototype implementation is tested with real world applications
  - Increase the throughput of both I/O workload (up to 88.3%) and computation workload (up to 123.1%)

Experimental Setup

- Dell PowerEdge R430 with 24 cores (48 HTs), a 1TB HDD, and 64GB DRAM.
- Both VMs and VMM (Linux QEMU/KVM) use Ubuntu Linux 18.04 with kernel version updated to 5.3.1.
- Each VM has 24 vCPUs and 16GB DRAM.
- Compared with three competing solutions
  - Priority boosting (implemented in KVM with HALT-Polling disabled)
  - Polling (implemented by booting guest OS with parameter ``idle=poll'' configured)
  - Polling with a dynamically adjusted timeout (implemented by enhancing KVM HALT-Polling)
- Test under two settings
  - Each vCPU has a dedicated hardware thread.
  - Each hardware thread is time-shared by multiple vCPUs.

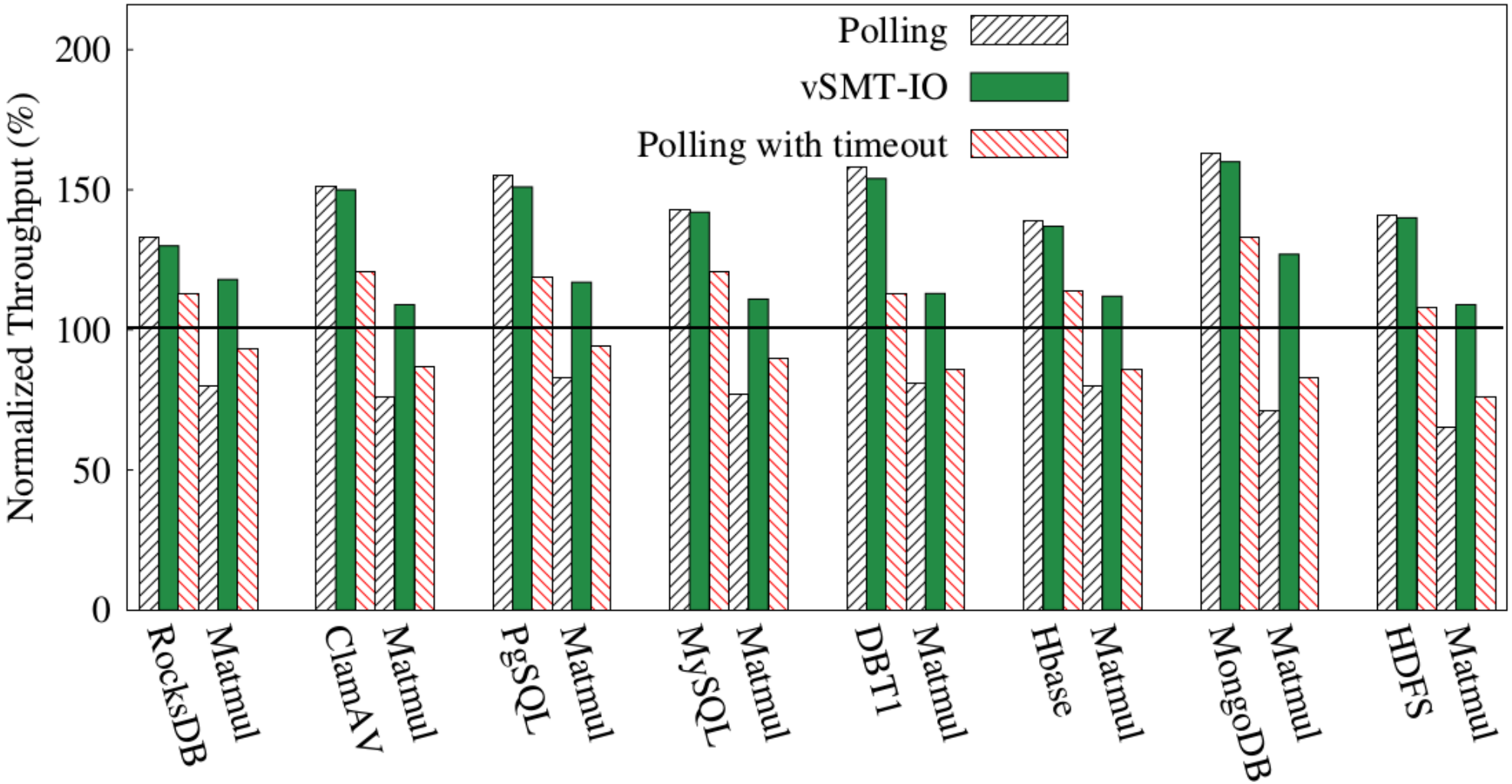
Evaluation applications and workloads

Application	Workload
Redis	Serve requests (randomly chosen keys, 50% SET, 50% GET)
HDFS	Read 10GB data sequentially with HDFS TestDFSIO
Hadoop	TeraSort with Hadoop
HBase	Read and update records sequentially with YCSB
MySQL	OLTP workload generated by SysBench for MySQL
Nginx	Serve web requests generated by ApacheBench
ClamAV	Virus scan a large file set with clamscan
RocksDB	Serve requests (randomly chosen keys, 50% SET, 50% GET)
PgSQL	TPC-B like workload generated by PgBench
Spark	PageRank and Kmeans algorithms in Spark
DBT1	TPC-W like workload
XGBoost	Four AI algorithms included in XGBoost system
Matmul	Multiply two 8000x8000 matrices of integers
Sockperf	TCP ping-pong test with Sockperf

Evaluation objectives

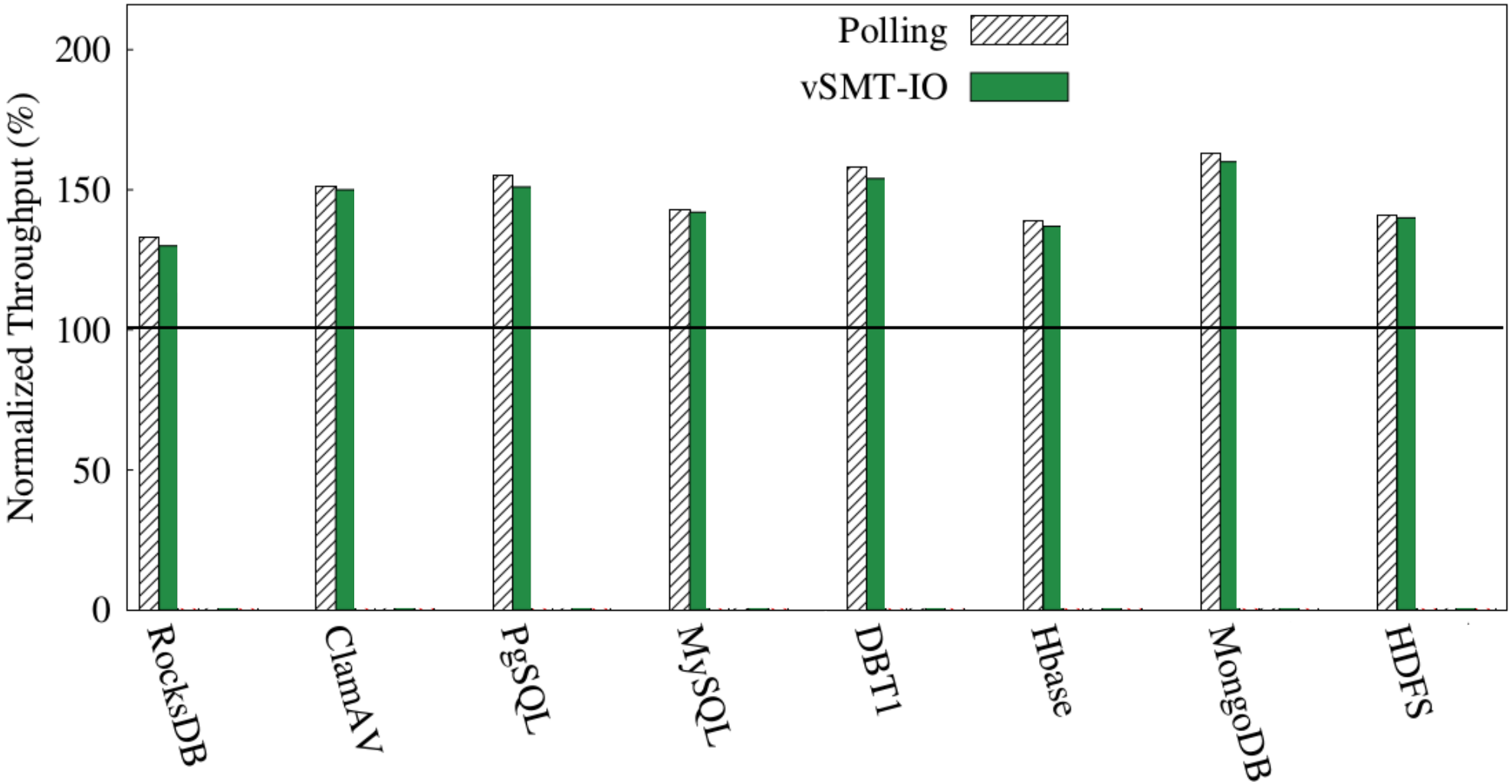
- To show vSMT-IO can improve I/O performance with high efficiency and benefit both I/O workloads and computation workloads.
- To verify the effectiveness of the major techniques used in vSMT-IO.
- To understand the performance advantages of vSMT-IO across diverse workload mixtures and different scenarios.
- To evaluate the overhead of vSMT-IO

Throughputs of eight benchmark pairs (one vCPU on each hardware thread)



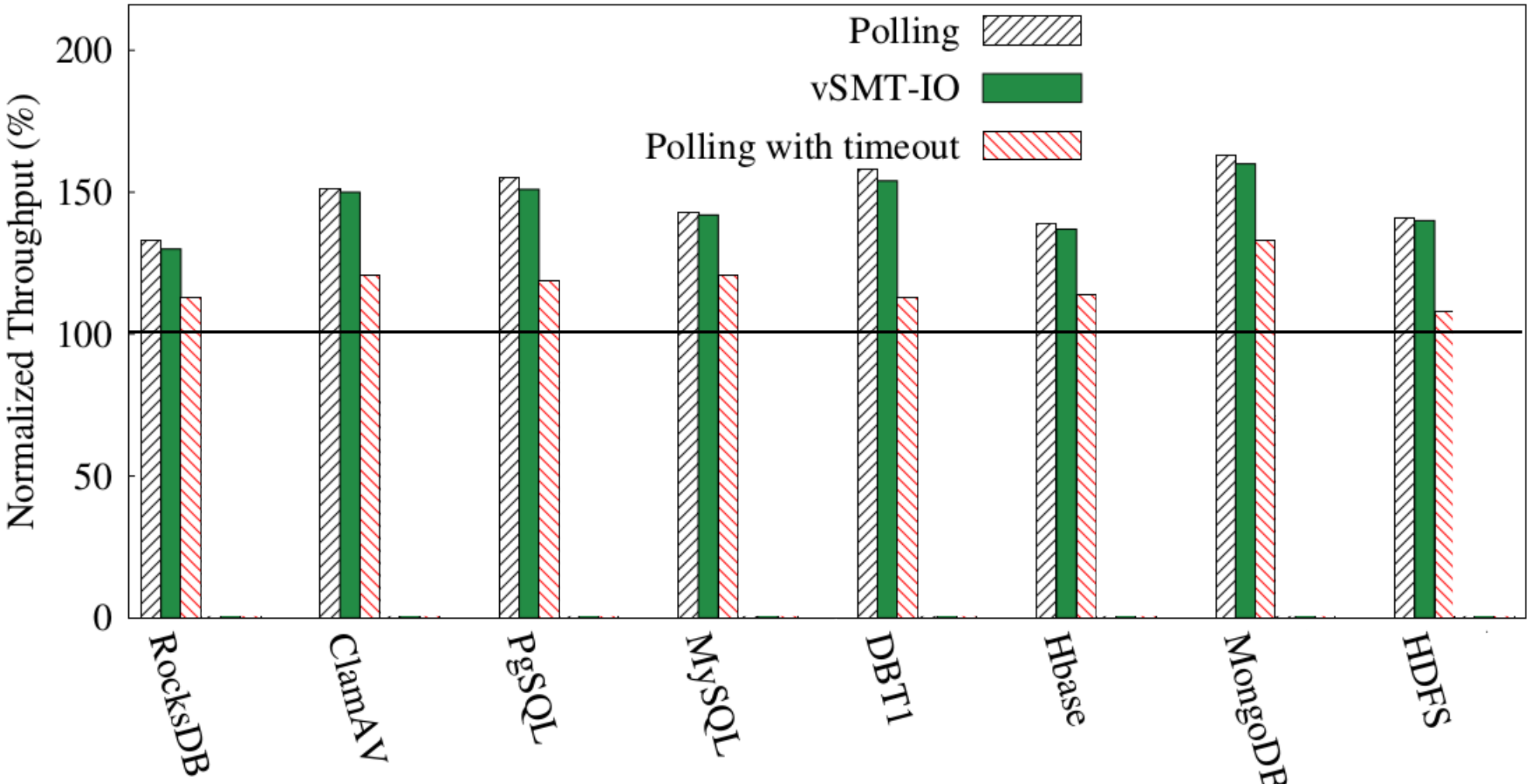
*Throughputs are relative to *priority boosting* (shown with horizontal line at 100%)

Throughputs of eight benchmark pairs (one vCPU on each hardware thread)



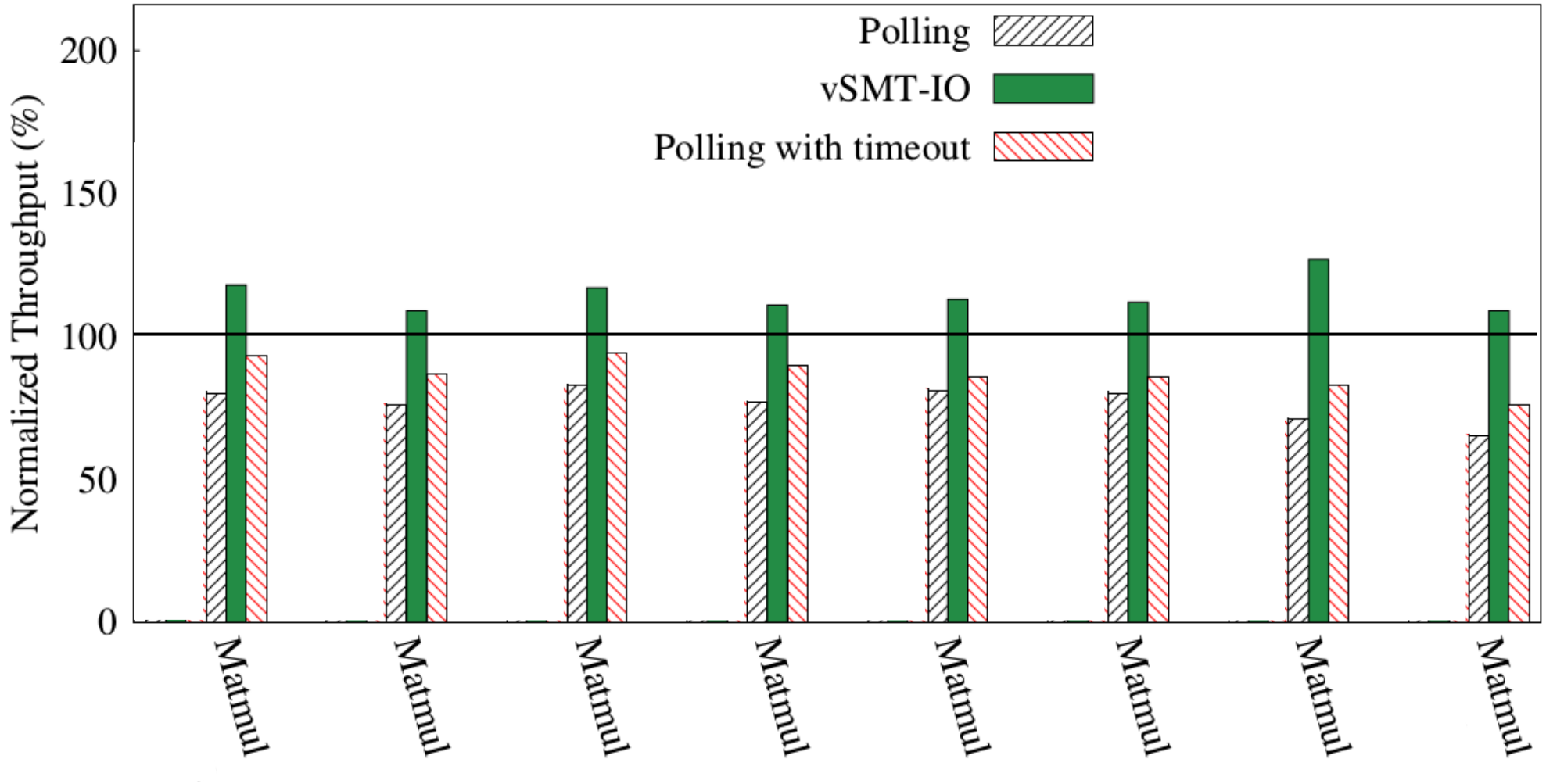
- vSMT-IO and *polling* achieve similar I/O throughput.

Throughputs of eight benchmark pairs (one vCPU on each hardware thread)



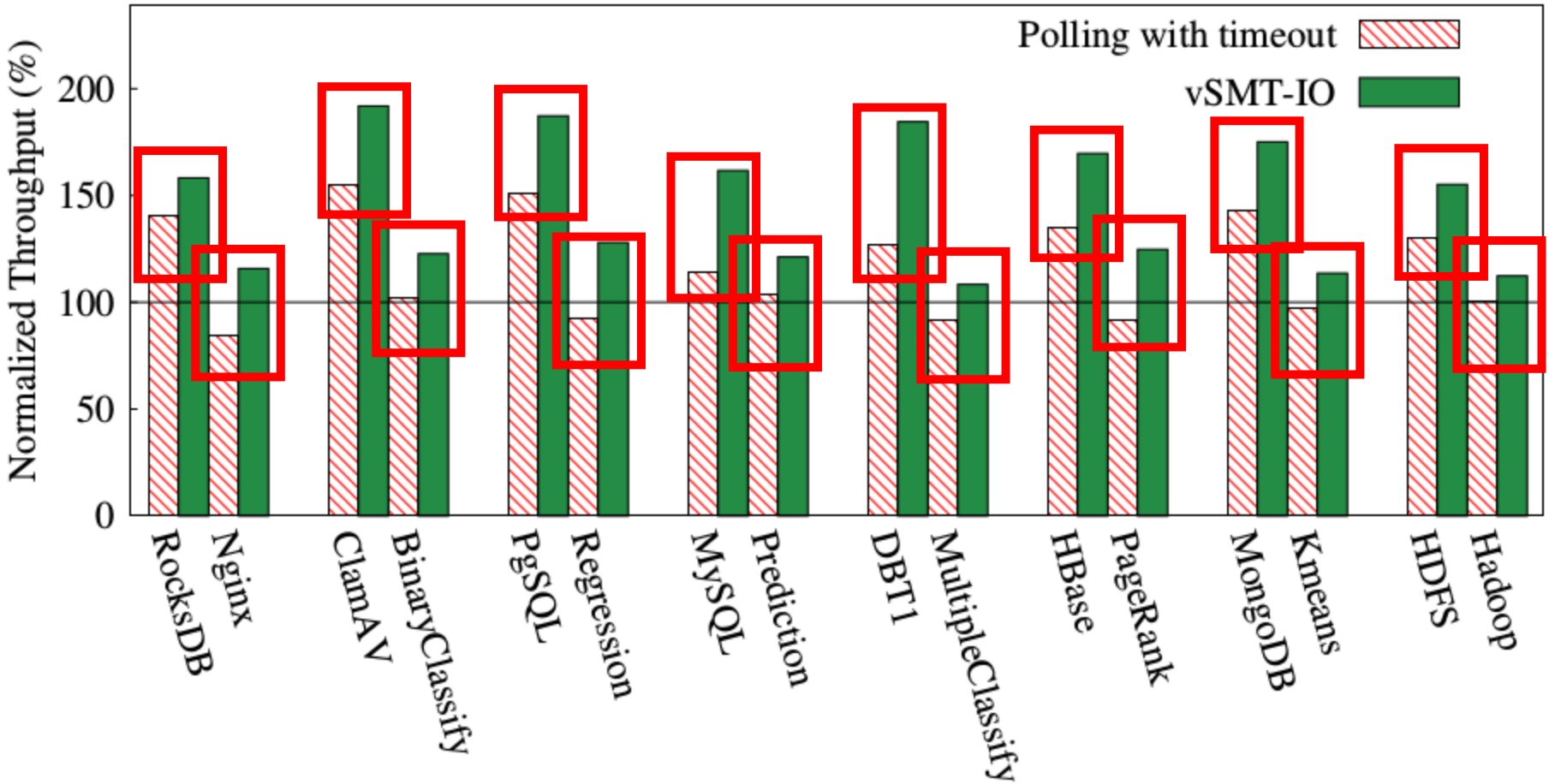
- I/O workload throughput increased by 46% relative to *priority boosting* and by 28% relative to *polling with timeout*.

Throughputs of eight benchmark pairs (one vCPU on each hardware thread)



- Throughput of computation workload (Matmul) increased by 38%, 15% and 28%, respectively, relative to *Polling*, *priority boosting* and *polling with timeout*.

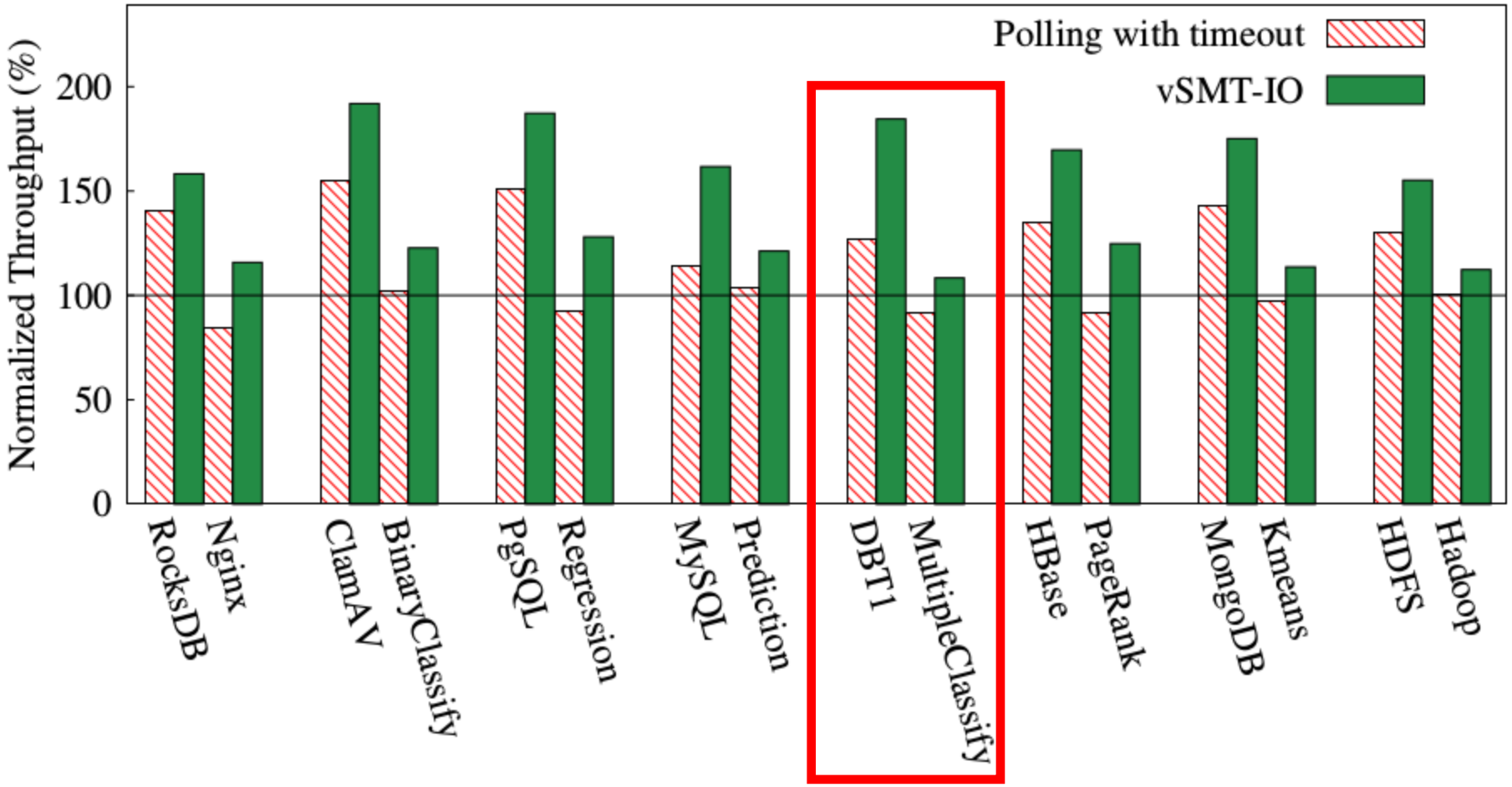
Throughputs of eight benchmark pairs (two vCPUs time-share a hardware thread)



* Throughputs are relative to priority boosting (shown with horizontal line at 100%).

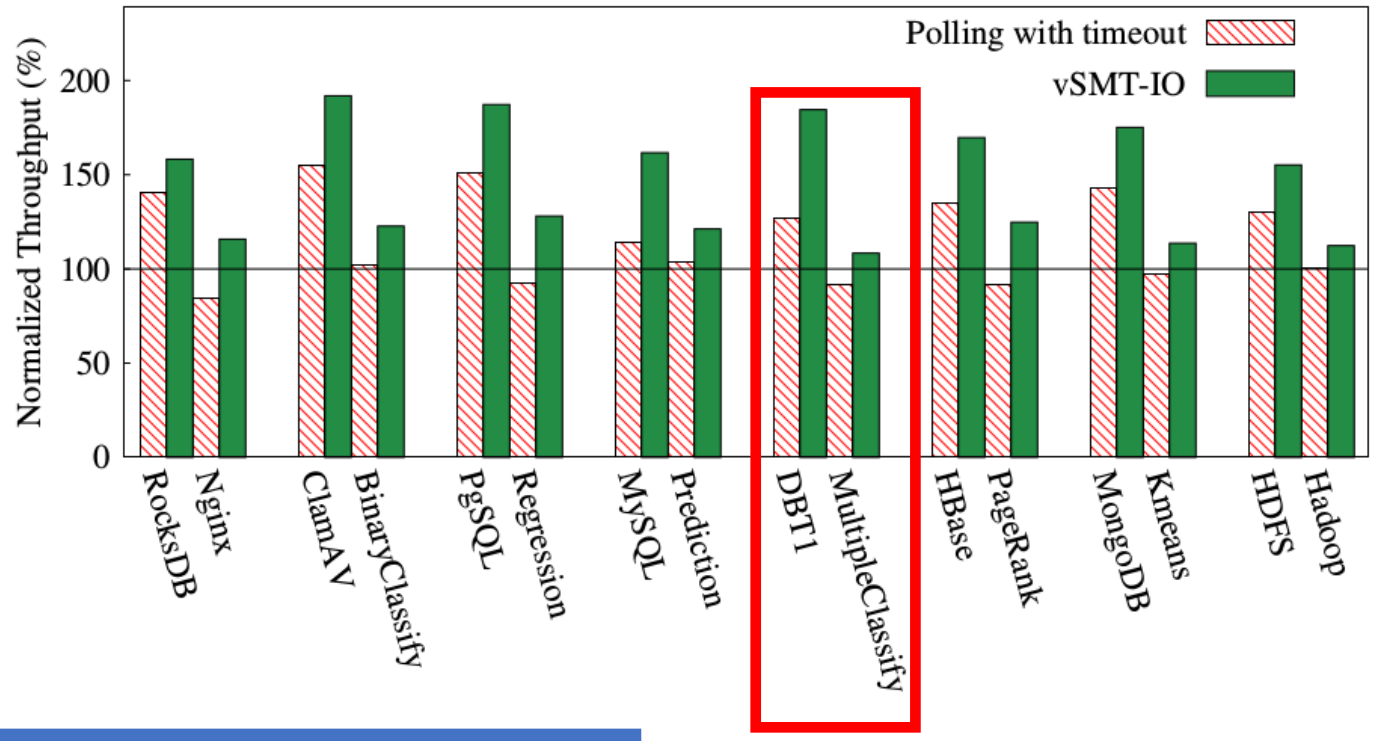
- I/O workload throughput increased by 30% relative to *polling with timeout*
- Computation workload throughput increased by 18% relative to *priority boosting*.

Analyzing performance Improvement with DBT1 and MultipleClassify



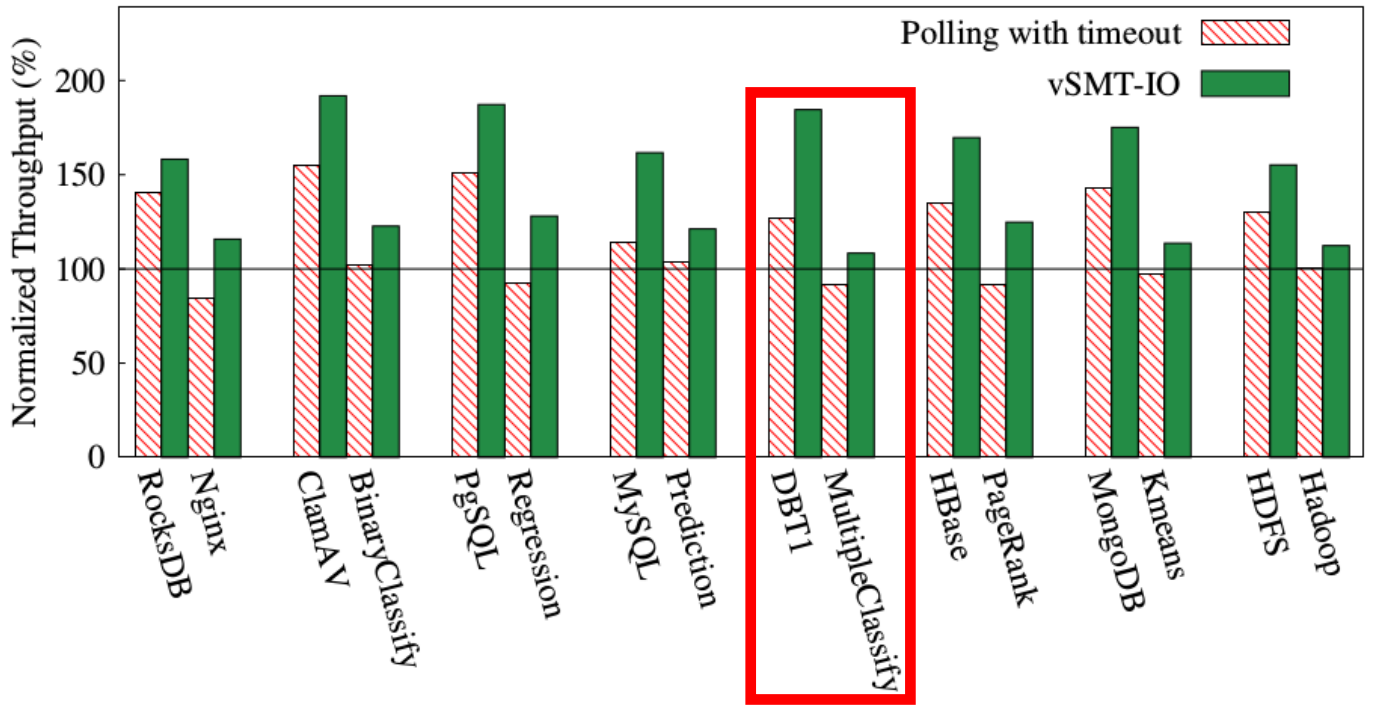
* Throughputs are relative to priority boosting (shown with horizontal line at 100%).

Analyzing performance Improvement with DBT1 and MultipleClassify



Number of vCPU switches per second		
Priority boosting	*Polling with timeout*	vSMT-IO
61.3k	29.5k	3.9k

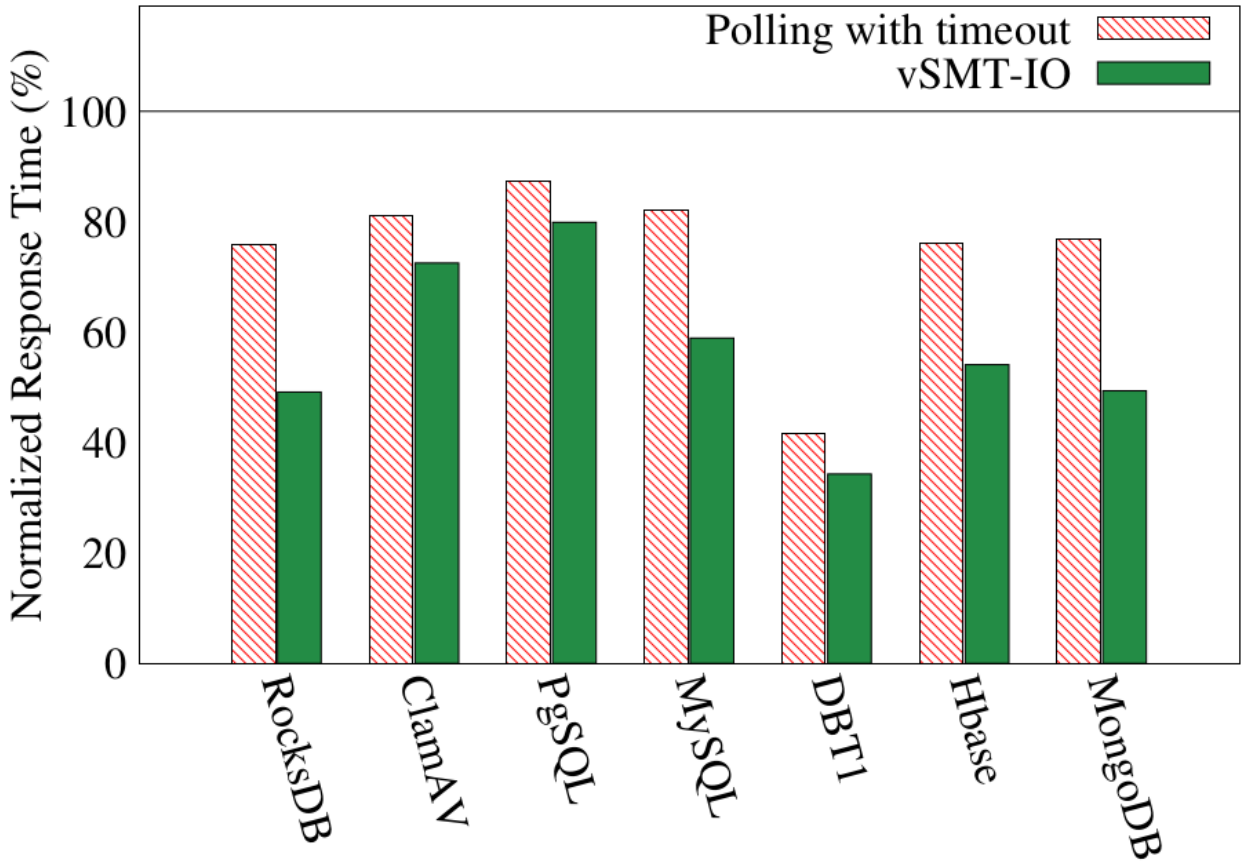
Analyzing performance Improvement with DBT1 and MultipleClassify



Number of vCPU switches per second		
Priority boosting	*Polling with timeout*	vSMT-IO
61.3k	29.5k	3.9k

Time (%) spent on hyper-threads for I/O bound vCPUs		
Context Retention	I/O Workload	Computation workload
32.7%	54.4%	12.9%

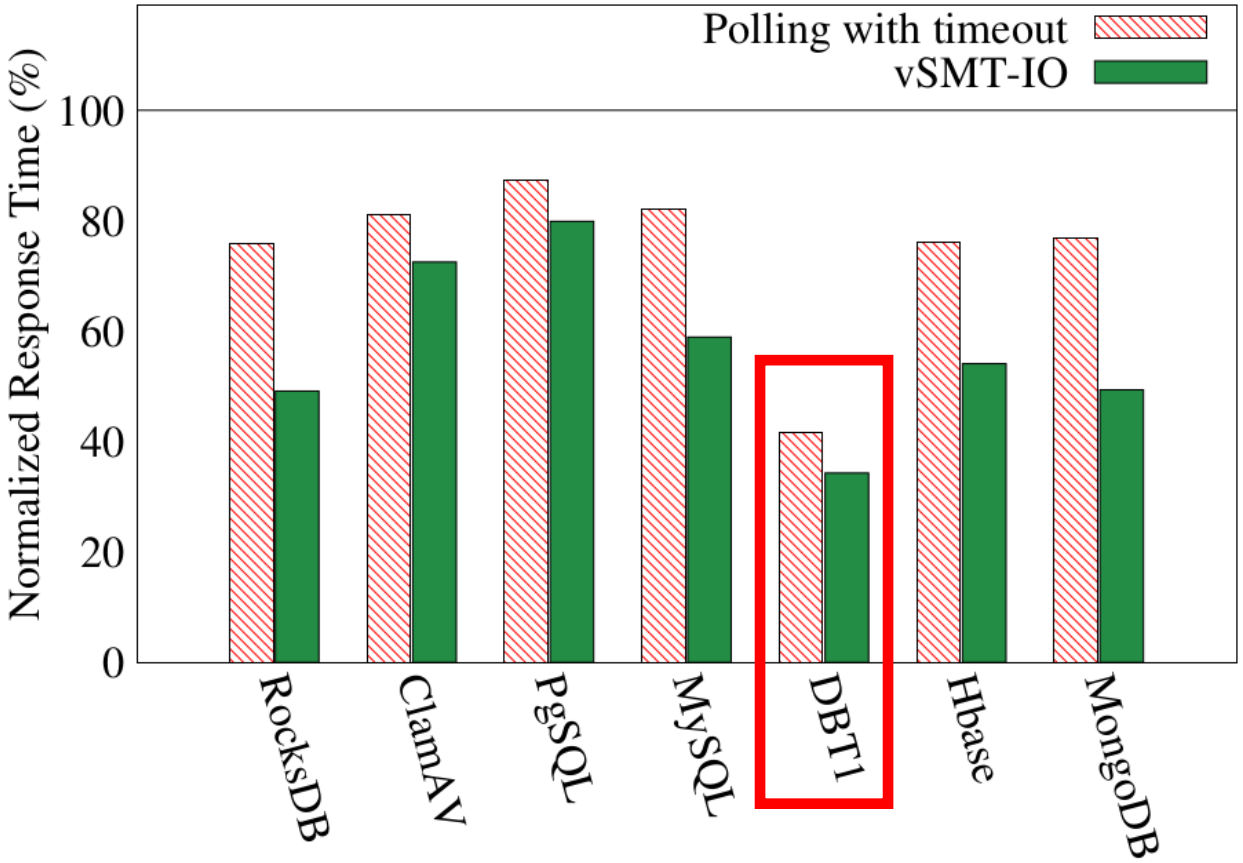
Response times of seven benchmarks (two vCPUs time-share a hardware thread)



* Response times are relative *to priority boosting* (shown with horizontal line at 100%).

- Response times are reduced by 51% relative to *priority boosting* and by 29% relative to *polling with timeout*.

vSMT-IO reduces response time by reducing scheduling delay of vCPUs

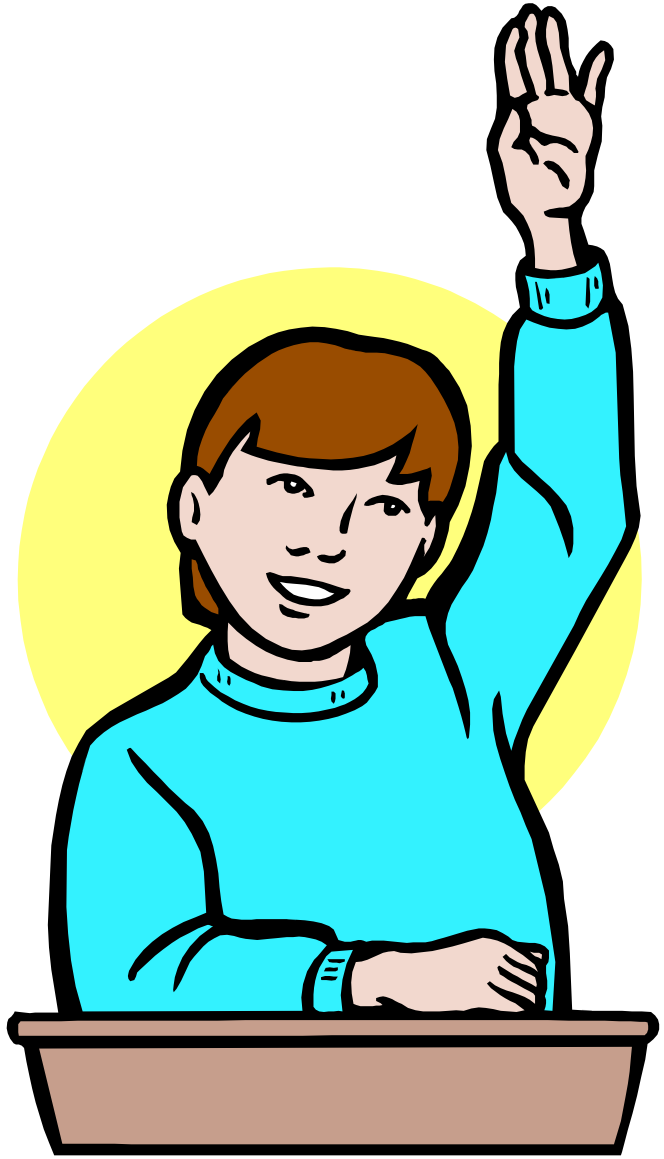


DBT1	*Priority boosting*	*Polling w/ timeout*	vSMT-IO
Time spent (ms) by vCPUs in ready state	1831.4	842.9	643.2
Time spent (ms) by vCPUs in waiting state	1390.2	1035.8	641.6

* Response times are relative *to priority boosting* (shown with horizontal line at 100%).

Conclusions

- How to improve I/O performance and efficiency on SMT processors is under-studied.
  - Existing techniques used by CPU schedulers are inefficient.
  - Such inefficiency makes it hard to achieve high CPU and I/O throughputs.
- vSMT-IO is an efficient solution for scheduling I/O workloads on x86 virtualized clouds.
  - Context retention uses a hardware thread to hold the context of an I/O workload waiting for I/O events.
  - Two key issues: 1) uncontrolled context retention can diminish the benefits from SMT; 2) existing symbiotic scheduling techniques cannot handle mixed workloads.
- Evaluation shows vSMT-IO can substantially increase both CPU throughput and I/O throughput.



Thank you!

Questions?