Performance implications from sizing a VM on multi-core systems: A Data analytic application's view

Seung-Hwan Lim*, James Horey*, Yanjun Yao†, Edmon Begoli* and Qing Cao†
*Oak Ridge National Laboratory
Oak Ridge, TN 37831
Email: {lims1,horeyjl,begolie}@ornl.gov
†University of Tennessee, Knoxville
Knoxville, TN 37916
Email: {yyao9, cao}@utk.edu

***Abstract*—In this paper, we present a quantitative performance analysis of data analytics applications running on multi-core virtual machines. Such environments form the core of cloud computing. In addition, data analytics applications, such as Cassandra and Hadoop, are becoming increasingly popular on cloud computing platforms. This convergence necessitates a better understanding of the performance and cost implications of such hybrid systems. For example, the very first step in hosting applications in virtualized environments, requires the user to configure the number of virtual processors and the size of memory. To understand performance implications of this step, we benchmarked three Yahoo Cloud Serving Benchmark (YCSB) workloads in a virtualized multi-core environment. Our measurements indicate that the performance of Cassandra for YCSB workloads does not heavily depend on the processing capacity of a system, while the size of the data set is critical to performance relative to allocated memory. We also identified a strong relationship between the running time of workloads and various hardware events (last level cache loads, misses, and CPU migrations). From this analysis, we provide several suggestions to improve the performance of data analytics applications running on cloud computing environments.**

I. Introduction

Data-intensive discoveries with Big Data [1] refers to analyzing the massive amount of stored data set to translate data into knowledge across a variety of social, scientific, and business applications [2]–[4], in which we use data analytics applications upon a variety of rapidly growing data. These data analytics applications are often distributed, operating in large cluster environments. Ultimately, the success of these applications is determined by whether massive amount of data can be processed in a timely manner [3]. To simplify storage and programming, tools like Hadoop [5], BigTable [6], and Cassandra [7] have emerged that expose simple programming interfaces without sacrificing capability. Often, these data analytics platforms are deployed in cloud or virtualized environments [8] in order to reduce the cost associated with the maintenance and operation of the infrastructure [9]. Such a cost reduction is achieved by allowing users to construct their computing infrastructure by themselves without dedicated infrastructure experts. Thus, in the Big Data era, non-expert users are likely to be exposed to the problems of obtaining reasonable performance of massively parallel applications in virtualized environments, presumably with minimal help from experts.

A. Backgrounds and related work

The first task when instantiating a Big Data platform in a cloud environment is to decide the type of instance [9], [10]. This includes, but is not limited to, selecting the number of virtualized CPUs (VCPUs) and the size of memory (often referred to as *sizing a VM* [11]). Sizing a VM has significant impact on the behavior of the application due to pressure on the system resources and the inherent sensitivity of the application on such pressure [12], [13]. Since data analytics applications are memory and I/O intensive, they are particularly sensitive to memory pressure.

Indeed, understanding the behavior of data analytics applications in virtualized environments requires expert knowledge across multiple layers. These layers include the hardware (often multi-core), the operating system, hypervisors, middleware (i.e., Java VM), and the application. Due to this complexity, choosing the optimal VM configuration can be a daunting challenge for non-expert users [10]. This is especially true in public cloud settings where poor choices may immediately lead to unnecessary monetary cost. To begin dealing with this problem, we attempt to understand the performance implication of determining the size of a VM for a data analytics application, specifically, Cassandra [7] (an open-source implementation of Google's BigTable [6]).

There exists a related body of work, complementary to our own, that attempts to compensate the misconfiguration of virtual machines. Virtualized resource management schemes [14] dynamically adjusts allocated VM resources within the known boundaries. Performance isolation schemes [15] minimize the interference among multiple VMs provided that the resource requirements are known ahead of time. Finally, virtual machine assignment schemes [16], [17] arrange VMs onto a set of physical hosts to balance load. A practical limitation in resizing VMs

(the number of VCPUs and size of memory), is that it often requires the reboot of VMs [18], [19]. In addition, the resources configured during VM sizing may ultimately offset the potential of resource management schemes. This makes it critically important to select good resource values since this affects the efficacy of performance isolation and VM assignment schemes. Hence, this study presents possible performance implications with various combinations of VM sizing for data analytics workloads.

B. Contributions

In this study, we collected experimental data from a minimal virtualized setting running Cassandra [7]. Our system consisted of a single VM, based upon KVM [19], running on a quad-core physical machine. VM parameters were adjusted methodically to determine the effect on overall performance. This minimal setting is intended to exclude the performance variability from network systems. We employ three pre-defined YCSB [20] benchmark workloads for Cassandra, a popular open-source key-value storage system.

While varying the number of virtual processors, we considered two different scenarios: when the size of data set is larger than the size of reserved memory; and when the size of data set is equal to the size of reserved memory. As we might expect, the experimental data suggests that, for data analytics applications, the number of VCPUs does not significantly affect performance (either positively or negatively). It is primarily because most of pressure on system resources is either on the memory system and secondary storage systems rather than processing units. For example, 99% of update operations complete in less than $1ms$ indicating update operations mostly hit the memory system.

In order to identify significant performance factors, we performed linear regression on the running time of workloads and hardware events (LLC loads and misses) instead of resource utilization for workload analysis as with [11], [21]. We identified that variation of running time is related to the number of LLC loads and LLC misses for two different size of data sets, given the size of virtual machine. All considered cases in this study showed R^2 values above 0.93. To understand the driving factor for the variance of last level cache accesses, we considered process scheduling. Process scheduling can create undesirable performance impact on memory-intensive workloads when the process scheduler assigns a process to different physical cores from prior time slices in order to balance CPU load [22], [23]. In our regression analysis, we found a linear relationship between the running time and CPU migrations with $R^2 = 0.94$ for 8GB data set and 0.86 for 2GB of data set. This statistical finding implies that current scheduling mechanism of virtual processors for KVM may contribute to extending the execution time of memory intensive workloads, which might be shared with other virtualization technologies like Xen [24] and VMware [25].

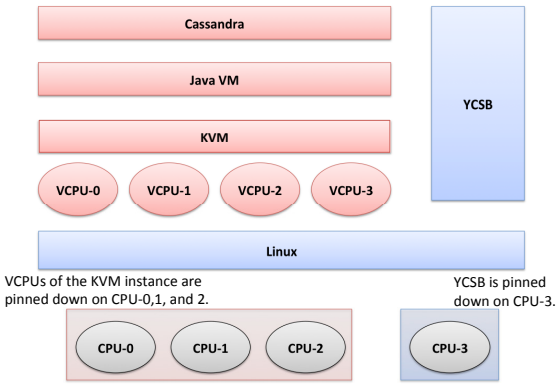


Figure 1: An overview of the experimental platform

The remainder of this paper is as follows. Methodology is presented in §II, along with a brief introduction of the systems used. Experimental results are presented in §III, along with discussion. Finally, we offer a short conclusion in §IV.

II. Platform and Methodology

Platform: An overview of the experimental platform is illustrated in Figure 1. We deployed a single KVM instance on the host system. Cassandra [7], a distributed key-value storage, ran on this instance. Details on the system software/hardware configurations are summarized in Table I. We employed three pre-defined YCSB workloads for workloads (Table II). The YCSB client ran on the host OS and was pinned down on a single core (CPU-3) . The YCSB client and KVM instance were dedicated to different sets of CPUs in order to minimize the interference on processing cores and private cache. Since the YCSB client applies

Table I: System configuration

Parameter	value	comments
Virtualization	KVM	
Host OS	Linux 3.0.0-20	Ubuntu SMP
Guest OS	Linux 2.6.35-32	Ubuntu SMP
Processor	Intel Xeon W3520	2.67GHz
# of cores	4	single thread per core
L1 cache	256 KB	write through
L2 cache	1024 KB	write back
Last Level cache	8192 KB	shared
System Memory size	6GB	4GB for host OS
VM memory size	2GB	
Memory Max bandwidth	25.6GB/s	
Hard Disk	Hitachi HDS721025CLA682	250GB, 7200RPM
VM disk type	file	
VM disk driver	qemu	type: raw
Java VM (JVM)	Open JDK 1.6.0_20	
JVM heap size	1GB	Cassandra default
Record size in YCSB	1 KB	default
# of clients in YCSB	1	default
request scheduler	no scheduler	Cassandra default
# of concurrent read threads	32	Cassandra default
# of concurrent write threads	32	Cassandra default

Table II: Summary of YCSB workloads

workload	operations	record selection	application example
A - Heavy update	50% read, 50% update	Zipfian	Session store recording recent actions in a user session
B - Read heavy	95% read, 5% update	Zipfian	Photo tagging
C - Read only	100% read	Zipfian	User profile cache, where profiles are constructed elsewhere like Hadoop

constant pressure on system resources, the authors believe that the conclusions drawn from the experimental results will not significantly differ from the situation where the KVM instance and YCSB client are hosted separately. These assumptions are shared with prior work [22], [23]. We now briefly describe KVM and Cassandra.

Kernel-based Virtual Machine (KVM): Kernel-based Virtual Machine (KVM) [26] is a Linux hypervisor (a.k.a, virtual machine monitor) implemented as a kernel module with a capability to leverage the rest of the Linux kernel as a hypervisor. By doing this, KVM reuses the majority of kernel modules and subsystems, and leverages any optimizations available through the kernel. Each virtual machine managed by KVM runs as a separate user process, and has private virtualized hardware(a network card, disk, graphics adapter, etc.). Although guest operating systems appear as any other user process, the hypervisor identifies each guest operating system as being in the "guest" mode independent of the kernel and user modes.

Cassandra: Apache Cassandra is an open-source distributed, key-value store inspired by Google's BigTable [6] and Amazon's Dynamo [27]. Cassandra exposes a multi-level map interface to programmers (Row $\rightarrow$ Column Family $\rightarrow$ Column $\rightarrow$ Value), and values are partitioned by the row value. Like Dynamo, Cassandra maps the row value to a machine via consistent hashing. Also like BigTable and Dynamo, Cassandra employs eventual write consistency across a set of replicas. Users, can however, specify the number of replica reads to enforce stronger read consistency.

The write path for Cassandra takes the following steps (see Figure2). First, the write is partitioned by row key to identify the target nodes. The write is then transmitted to each target, where the write is stored simultaneously in the in-memory table (memtable) and disk-based commit log. Once the memtable reaches a particular size threshold, the memtable is flushed to file systems as an *sstable*. Similarly, reads identify the target nodes and attempts to find the values in the memtables. If the values are not found, Cassandra will scan the sstables (Cassandra also employs bloom filters [28]) to reduce disk activity).

Methodology: We collected the overall running time of pre-defined YCSB workloads from the output of the YCSB. `perf` is utilized in order to obtain hardware event statistics such as CPU migrations and Last Level Cache (LLC) misses. For setting processor affinity of the YCSB client and the KVM instance, we used `taskset`. As for pinning the VCPU of KVM instance, we used `virsh vcpupin` in addition to `taskset`.

III. The performance impact from different VM sizes

As shown in Figure 3, we measured the average running time of 800,000 operations for three pre-defined YCSB

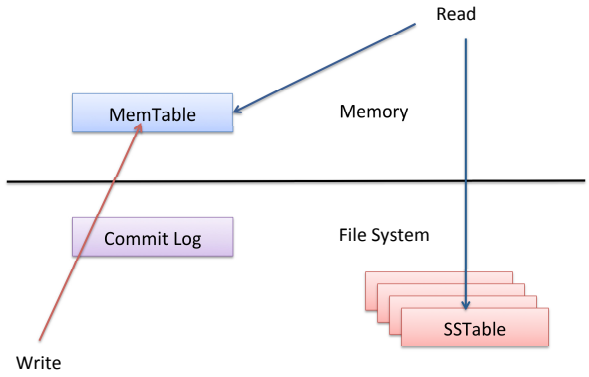


Figure 2: The overview of READ/WRITE operations in Cassandra.

workloads [20] on Cassandra [7] in virtualized environments. While fixing the size of reserved memory to the virtual machines, we varied the number of virtual processors and the data set size, due to the limitation of available memory size in the testing environments. Since three out of four physical cores were dedicated to the VM instance that hosted Cassandra, varying the number of VCPUs from one to four can represent three situations: the number of VCPUs are smaller than, equal to, and larger than the number of physical cores. We employed both a 2 GB data set and an 8 GB data set while fixing the size of allocated memory to 2 GB. This tests two situations: one where the VM has enough memory to buffer most of the data set in memory; and where the VM does not have enough memory.

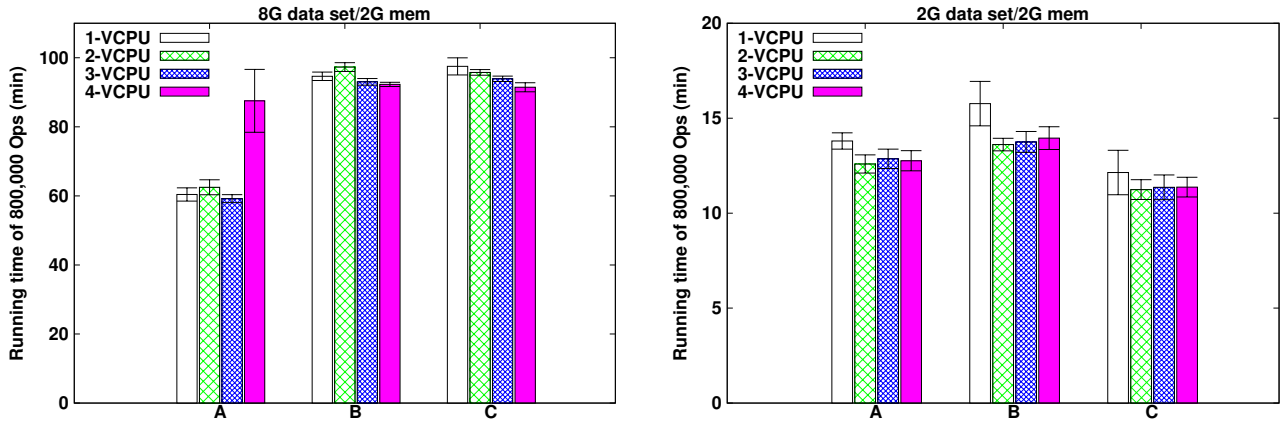
Figure 3 finds that the execution times of Cassandra workloads do not clearly depend on the number of virtual processors, as long as the associated physical resources remains the same. However, the ratio of data set size and reserved memory size is a more decisive factor when we compare the scale of running time between Figure 3a and Figure 3b. This result matches with the intuition that most data analytics workloads like Cassandra will depend on memory capacity instead of processor capacity since they are mostly memory (or IO) intensive.

Suppose, however, that for a user the system condition was initially similar to Figure 3a. Then, the user may want to find answers to the following questions:

- Performance implications as the data set size increases
- Performance benefits as the data set size decreases
- Performance optimization opportunities without changing the data set or virtual machine configuration

In order to address those issues, the rest of this section provides statistical analysis of the behavior of Cassandra workloads and the most influential factor to the execution time of workloads.

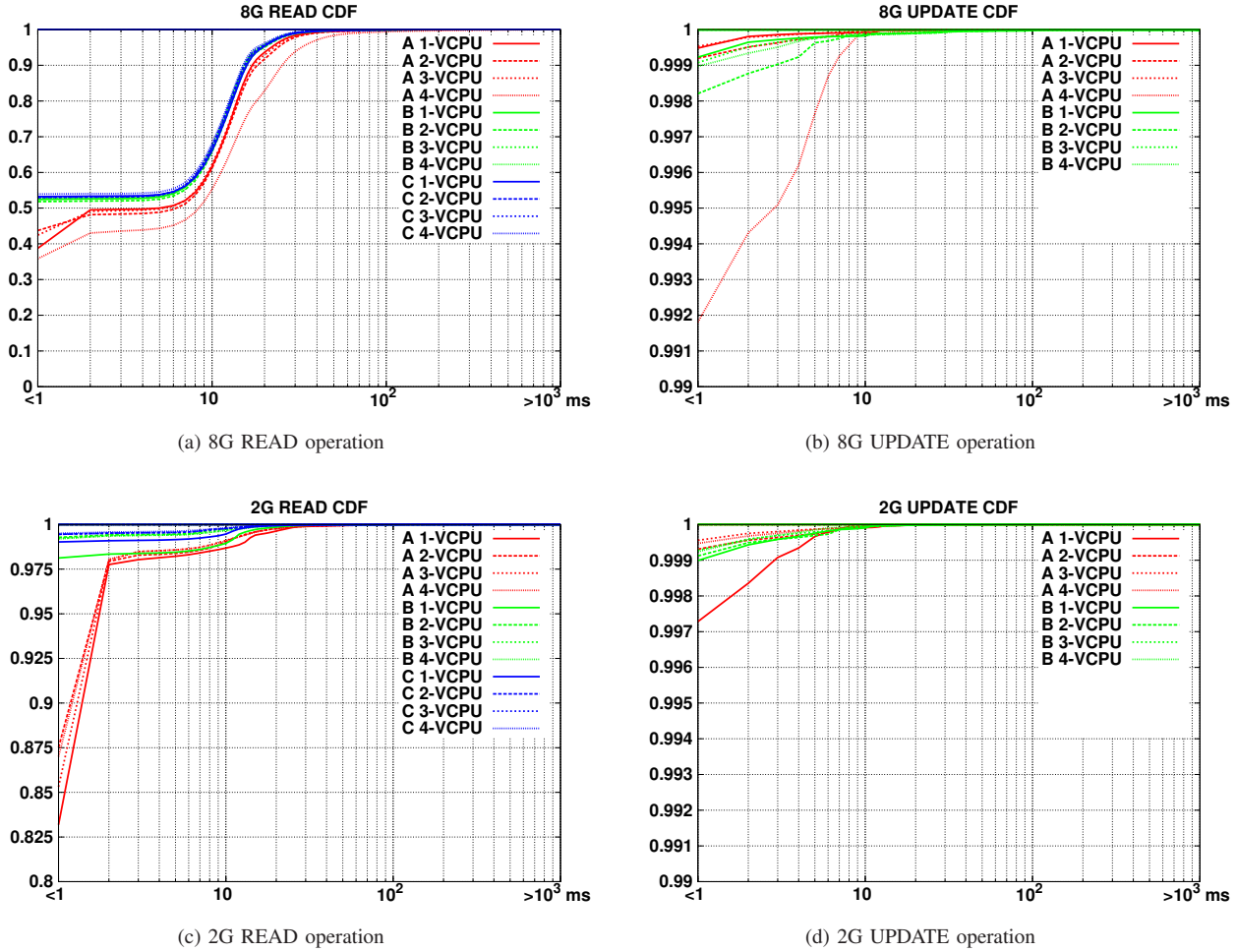
We now illustrate the statistical relationships between the size of memory for each operation against set workloads (consisting of both *READ* and *UPDATE*).



(a) 8G data set

(b) 2G data set

Figure 3: Average running time of 800,000 operations for three pre-defined YCSB workloads in virtualized environments. Error bars represent 95% confidence intervals.



(a) 8G READ operation

(b) 8G UPDATE operation

(c) 2G READ operation

(d) 2G UPDATE operation

Figure 4: CDFs of each type of operations: Workload-A (50% read, 50% update), Workload-B (95% read, 5% update), and Workload-C (100% read, 0% update)

A. Understanding related operations in Cassandra

Figure 4 shows the cumulative distribution function of each operation for three pre-defined workloads while varying the number of virtual processors and the size of the data set. From these analysis, we find that 90% of UPDATE operations take less than 1 ms whether we employ different sizes of data sets. This indicates that most UPDATE operations in Cassandra only involve memory accesses. However, for READ operations, as data size grows from 2G to 8G, disk accesses increase. 50% of READ operations show less than 1 ms of latency, but the majority of the rest take between 10 ms and 30 ms.

Finding *1:* A UPDATE operation in Cassandra is principally related to memory access. However, depending on the ratio between data set size and allocated memory size, a READ operation may be linked to disk access as well as memory access.

B. Identifying the influential factor by regression analysis

In order to identify the influential factor for Cassandra performance, we examined the relationship between the running time of workloads and various hardware/software events such as the number of Last Level Cache (LLC) loads; LLC load misses; and CPU migrations. These events are relatively easy to obtain, compared with the detailed application-specific information. We performed regression analysis from 120 runs for all three pre-defined workloads with four different virtual processor configurations, as shown in Figure 5. In this regression analysis, we confirm that all considered events were linearly associated with the running time of workloads with R^2 values of greater than 0.86. Hence, we argue that the variance of running time of workloads stem from the variance of those hardware events in each run instead of higher level informaion like resource utilization like [11].

Intuitively, LLC loads and LLC load misses are important for data analytics workloads since these workloads are memory-intensive. Thus, for memory-intensive workloads, prior work [22] have used LLC load misses as an indicator to memory pressure from a workload. Our regression analysis on LLC loads and LLC load misses supports these prior studies with quantitative experimental data. Additionally, we found that CPU migrations pose greater impact than cache accesses. Since CPU migration is an attempt to balance the load across processors, one may expect that overall performance is not negatively affected by CPU migrations. However, we will demonstrate how CPU migration can negatively impact the performance of memory intensive workloads.

Running time and last level cache (LLC) loads: LLC loads occur when private cache misses happen, which can be described by:

$$N(miss(private\ cache)) = N(LLC\ loads)\,.$$

Thus, the regression analysis on LLC loads and running time will mainly show the performance impact from private cache misses. Figure 5a indicates that one LLC load attributes to 56.2 ns on average for the 2GB data set and 20.6 us on average for the 8GB data set (inferred from the average slope of the linear regression equations). Although a single miss results in a small penalty, the number of LLC loads may be quite numerous. If we can reduce a small percentage of the number of LLC loads, we can expect a noticable performance improvement. At an extreme case, if we have no LLC loads (all operations only access L1/L2 cache), the running time of all workloads would operate in the ns scale, as suggested by the Y-intersect of the regression equations.

Additionally, the steeper slope for the 8GB data set suggests that the size of the data set changes the overall sensitivity of the workload to memory access patterns. Since the operations are the same, the difference in sensitivity to memory pressure stems from the difference in access patterns of next level memory subsystem or secondary storage systems. To summarize, LLC loads include LLC load misses, that is,

$$N(LLC\ loads) = N(LLC\ hits) + N(LLC\ misses).$$

Thus, we hypothesize that LLC load misses produce greater performance impact on the workloads with 8GB data sets.

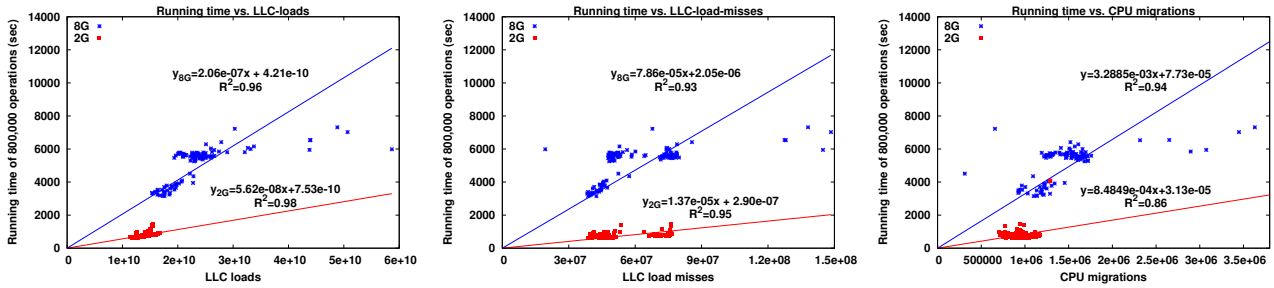
Finding *2:* While individual LLC loads pose small performance impact, their numbers are numerous and noticably affects overall performance. This implies that we have room to enhance performance of memory-intensive data analytics workloads by carefully treating LLC loads. Additionally, for $Size(memory) > Size(data\ set)$, LLC loads may have greater performance impact.

Running time and LLC load misses: We confirm the linear relationship between running time of workloads and LLC load misses in Figure 5b. We observe that one LLC load miss is more responsible for the running time of workloads for the 8GB data set than the 2GB data set case as the slope for the 8GB set is several times steeper. This occurs principally because of more disk access for the 8G read operation as we have shown in CDF plots (Refer to Figure 4a.) In short,

$$N(LLC\ miss) = N(Mem\ access) + N(Disk\ access),$$

where memory accesses can happen in two layers in our experimental environment: the page/buffer cache at guest OS and host OS. In addition, we find that one LLC load miss incurs two or three orders higher magnitudes of performance impact on the Cassandra workloads than LLC loads, though the number of LLC load misses is less than 1% of the number of LLC loads. The lesson from this analysis can be summarized by:

Finding *3:* For memory-intensive workloads like Cassandra, hardware events related to memory behavior can be a good indicator to understand the overall behavior



(a) LLC-loads (b) LLC-load-misses (c) CPU migrations

Figure 5: We observe a linear relation between the running time of workloads and the number of LLC loads;LLC-load misses; and CPU migrations.

of workloads. For example, we can observe the different sensitivity of each application to memory pressure, which can reveal their access patterns through space-resources like on-chip cache; off-chip memory; and secondary storage.

Until now, we have identified that performance implications from underlying memory or storage hierarchy can be understood by a linear relationship between the running time of workloads and hardware performance events like LLC loads or LLC load misses. Additionally, we have analyzed factors that cause variance in the number of LLC loads and LLC load misses. We expect to use these factors to develop mechanisms to control LLC loads and LLC load misses. For this, we analyzed the explicit relationship between the running time and CPU migrations.

Running time and CPU migrations: CPU migrations are the migration of processes across processing cores in a machine as a result of processor scheduling. The version of Linux kernel employed in this study uses the Completely Fair Scheduler (CFS) [29]. This Linux process scheduler pays special care to cache behavior by considering migration cost. Migration cost in the Linux scheduler denotes the expected amount of time after the last execution that a task is considered to be "cache hot" in migration decisions. The default Linux kernel scheduler considered in this study, CFS, uses a fixed migration cost for all processes in the system ($0.5ms$ by default). Thus, if a task is on the run queue less than $0.5ms$, it will not be migrated. Otherwise, it might be migrated depending on the load of the previously allocated processor. Although used to balance load and improve performance, paradoxically, CPU migration can hurt the performance of memory-intensive workloads. For example, when a CPU migration happens, the system must re-populate the private cache after the process is migrated from one core to another core. This private cache re-population will create additional LLC loads, which could possibly lead to LLC load misses and, thereby, increase memory and I/O accesses.

We illustrate (Figure 5c) that the processor-bound Linux scheduler is associated with the slow-down of memory-intensive workloads. The slopes of regression equations can be interpreted as the impact on running time per migration, that is, the slope will be related to the sensitivity of workloads to CPU migrations. As expected, workloads with 8GB of data sets are more sensitive to CPU migrations since, for the larger data set, CPU migration leads to either off-chip memory accesses or disk accesses with higher probability.

However, the sensitivity of workloads to CPU migrations should enclose the duration of one operation, that is,

$$T(operation) = T(migration) + T(processing)\,.$$

And the slope accounts for $T(migration)$ part. For instance, the average latency of one operation is around $1ms$ for 2GB data set, and $5.6ms$ for 8 GB data set, Thus, we observe that, on average, one CPU migration shares the latency of one operation by $8.4849 \times 10^{-4}/1 \times 10^{-3} = 84.8\%$ and $3.2885 \times 10^{-3}/5.6 \times 10^{-3} = 58.7\%$ for 2GB and 8GB data sets, respectively. Hence, by optimizing the scheduler behavior for each workload, we can maximally expect the same portion of reduction in the running time of workload, or equivalently the average running time of each operation.

Intuitively, the 8GB data set case consists of significant portions of I/O accesses in its processing time, $T(processing)$. On the other hand, the 2GB data set mostly hits memory only, which makes the $T(processing)$ portion much smaller than 8GB data set cases. Therefore, for optimizing CPU migration, 2GB cases may potentially reduce larger share of running times than 8GB cases. However, note that the aforementioned optimization only accounts for the enhancement of memory access parts in memory-intensive workloads. A trade-off from minimizing CPU migrations may include negative performance impact on processing portion of the workloads, $T(processing)$, because of unbalanced loads on processors. We summarize our finding from regression analysis on the running time and CPU migration as follows:

Finding *4:* The performance impact from CPU migration exceeds the performance impact from cache access patterns.

It is primarily because one CPU migration can produce additional cache accesses from private cache to LLC, which can affect the main memory accesses as well as disk accesses.

C. Discussion

Since CPU migration is the result of process scheduling, we can manage CPU migrations by adjusting tunable parameters of existing process schedulers or inventing new process schedulers. For instance, a recent study [23] expressed the same argument and proposed a process scheduler that balances memory load from processes in virtualized environments. Similarly, *Zhuravlev et. al.* [22] and *Blagodurov et. al.* [30] proposed novel process schedulers that aware memory contention in native Linux environments. While those prior work engineered actual process schedulers in multi-core environments, our analysis provides a quantitative analysis on slow-down of workloads,which can be attributed to CPU migrations. Thereby, this study can help to develop a method to estimate the potential performance enhancement from optimal process scheduling in multi-core environments.

Considering the analysis in § III-B, we propose pursueing several development directions for multi-core schedulers:

1) An efficient mechanism to quantify the overhead from CPU migration for workloads in a specific system. A runtime mechanisms that can adapt to dynamic workloads could further reduce the average execution time of a variety of workloads.
2) A variety of scheduling methods might be able to utilize a CPU migration cost analysis model. This includes both conventional process schedulers [29] as well as affinity-based schedulers [23]. By better utilizing multi-cores in a system, the performance of co-hosted heterogeneous workloads [13] will improve.
3) In addition, we propose a novel scheduling mechanism capable of considering multiple resources. This mechanism does not need to exhaustively monitor resources, since we showed that we can infer the various resource usage like cache, memory, and secondary storage from analyzing the running time and a few specific hardware/software events.

IV. Conclusions

The cost and flexibility afforded by cloud computing makes it an ideal platform to host large-scale data analytic applications. Although it is simple to get started from an organizational viewpoint, we have shown that optimal performance of these applications may depend on particular virtual machine configurations. Specifically we investigated the role of the number of virtual processors and the size of memory on the overall performance of Cassandra.

We found the running time of data analytics workloads is not strongly dependent on processing capability, and that the memory and I/O subsystems play a larger role in overall performance. We have shown that update operations in Cassandra consists of primarily memory accesses even with large data set sizes relative to VM and host memory. Our analysis has shown that read operations can be I/O bound, however, depending on the relation between the data set size and the size of VM memory.

We also presented regression analysis that identified the factors that influence the performance of Cassandra for a set of workloads. We performed regression analysis against the running time of workloads over three performance counter events: last level cache loads, last level cache load misses, and CPU migrations. These results showed a statistically strong ($R^2 > 0.8$) linear relationship with the running time of workloads. The regression analysis from this work suggests that investigating the behavior of last level cache accesses can produce insights on the dependencies between cache, memory, and secondary storage.

We believe that results from this study can be used to devise a mechanism to quantify the overhead from CPU migrations for heterogeneous workloads in a system. Using such information, it would be possible to construct process schedulers that simultaneously takes care of both memory and processor usage by quantifying the CPU migration cost. Such a scheduler could improve overall performance of heterogeneous workloads that characterize cloud computing scenarios. In addition, our work indicates need to further optimize Big Data tools to take advantage of multi-core architectures. Since multi-core architectures will only become more commonplace in the near future, we believe it is paramount to address this need.

Acknowledgment

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] T. Hey, S. Tansley, and K. Tolle, *The Fourth Paradigm: Data-Intensive Scientific Discovery*, 2009.

[2] T. Kalil, "Big data is a big deal," http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal.

[3] S. Lohr, "The age of big data," *New York Times*, 2012.

[4] A. McAfee and E. Brynjolfsson, "Big data: the management revolution." *Harvard Business Review*, vol. 90, no. 10, pp. 60–6, 68, 128, 2012.

[5] Apache Hadoop, http://hadoop.apache.org.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06, 2006.

[7] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Operating System Review*, vol. 44, no. 2, Apr. 2010.

[8] S. Chaudhuri, U. Dayal, and V. Narasayya, "An overview of business intelligence technology," *Communication of the ACM*, 2011.

[9] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "Understanding the cost of cloud: Cost analysis of in-house vs. cloud-based hosting options," *The European Business Review*, Sep 2011.

[10] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics," ser. SoCC '11, 2011.

[11] R. Ganesan, S. Sarkar, and A. Narayan, "Analysis of saas business platform workloads for sizing and collocation," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, june 2012.

[12] P. J. Denning, "The working set model for program behavior," *Commun. ACM*, vol. 11, no. 5, May 1968.

[13] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[14] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009.

[15] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Eurosys*, 2010.

[16] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.

[17] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2009.

[18] Citrix Systems, "Citrix xen server 6.0 administrator guide," http://docs.vmd.citrix.com/XenServer/6.0.0/1.0/en_gb/reference.html.

[19] KVM, http://www.linux-kvm.org/page/Main_Page.

[20] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," ser. SoCC '10, 2010.

[21] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*, july 2011.

[22] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ser. ASPLOS '10, 2010.

[23] M. Lee and K. Schwan, "Region scheduling: efficiently using the cache architectures via page-level affinity," in *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '12, 2012.

[24] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," ser. SOSP '03, 2003.

[25] VMware, "Vmware vsphere: The cpu scheduler in vmware esx 4.1," 2010, http://www.vmware.com/resources/techresources/10131.

[26] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[27] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07, 2007.

[28] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, Jul. 1970.

[29] C. S. Pabla, August 2009, http://www.linuxjournal.com/magazine/completely-fair-scheduler.

[30] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A case for numa-aware contention management on multicore systems," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, ser. USENIXATC'11, 2011.