2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)

The Power of ARM64 in Public Clouds

Qingye Jiang
School of Computer Science
The University of Sydney
Sydney NSW 2006, Australia
Email: qjiang@ieee.org

Young Choon Lee
Department of Computing
Macquarie University
Sydney NSW 2109, Australia
Email: young.lee@mq.edu.au

Albert Y. Zomaya
School of Computer Science
The University of Sydney
Sydney NSW 2006, Australia
Email: albert.zomaya@sydney.edu.au

***Abstract*—ARM processors, with their low power consumption and heat dissipation, have been highly successful in embedded systems. In the recent past, there have been attempts to adopt these energy-efficient processors for servers in data centers. However, a fundamental question remains open with ARM-based systems on server side is whether they are capable of handling compute-intensive workloads at scale. This paper gives our answer to this question with an empirical approach. We study the performance characteristics of the Amazon Graviton Processor – an ARM64 processor with the Cortex-A72 micro-architecture – using the A1 (Graviton) product family on AWS EC2, with comparisons to the I3 and M5 product families based on Intel Xeon processors. We use a combination of micro benchmark and performance counters to identify the lack of L3 cache and the slower memory access speed limit Graviton's capability in achieving higher performance. We confirm Graviton's capability in handling various large-scale horizontally scalable compute-intensive workloads, including multi-tier web service, video transcoding and terabyte scale sorting. In our large-scale evaluations, the test worker fleet has up to 1600 vCPU cores, which is by far the largest ARM64 cluster that has been reported. We observe that the A1 product family achieves the same price-performance in multi-tier web service, up to 37% cost saving in video transcoding, and up to 65% cost saving in terabyte scale sorting, as compared with the I3 and M5 product families.**

I. Introduction

In recent years, there is an increasing attention on energy consumption in the IT industry, data centers in particular [1]–[3]. From the processor architectural point of view, there have been two distinctive approaches to achieve higher level of energy efficiency in data centers. One approach is using low-power processors with a complex instruction set computer (CISC) architecture [4], for example, the Intel Atom product family. The other approach is using processors with a reduced instruction set computer (RISC) architecture, for example, the ARM [5] product family. As compared to processors with a CISC architecture, processors with a RISC architecture usually require fewer transistors, leading to lower power consumption and heat dissipation. Traditionally, ARM processors have been highly successful in embedded systems including tablet computers and smart phones. However, the lack of successful deployments with reasonable scale on the server side hinders the adoption of ARM systems in data centers, which in turn makes large-scale studies difficult. To date, a fundamental question remains open: are ARM systems capable of handling compute-intensive workloads at scale?

TABLE I: EC2 Instance Types.

Instance Type[2]	vCPU Cores	MEM (GB)	Price ($/hr)	EBS Bandwidth (Mbps)[3]	Network Bandwidth (Mbps)
a1.large	2	4	0.051	3500	10000*
a1.xlarge	4	8	0.102	3500	10000*
a1.2xlarge	8	16	0.204	3500	10000*
a1.4xlarge	16	32	0.408	3500	10000*
i3.large	2	15.25	0.156	425	10000*
i3.xlarge	4	30.5	0.312	850	10000*
i3.2xlarge	8	61	0.624	1700	10000*
i3.4xlarge	16	122	1.248	3500	10000*
i3.8xlarge	32	244	2.496	7000	10000
i3.16xlarge	64	488	4.992	14000	25000
m5.large	2	8	0.096	3500*	10000*
m5.xlarge	4	16	0.192	3500*	10000*
m5.2xlarge	8	32	0.384	3500*	10000*
m5.4xlarge	16	64	0.768	3500	10000*
m5.8xlarge	32	128	2.304	5000	10000
m5.12xlarge	48	96	2.304	7000	10000
m5.16xlarge	96	192	4.608	10000	20000
m5.24xlarge	96	192	4.608	14000	25000

The rise of public clouds has significantly changed the horizon in the computing resources market [6]. As public cloud service providers grow bigger, they are also constantly looking for ways to reduce the energy consumption in their data centers. In 2018, AWS announced the Amazon Graviton Processor, a 64-bit ARM processor based on the Cortex-A72 micro-architecture. The Amazon Graviton Processor is made available on EC2 as the A1 product family [1]. This allows the general public to access ARM64 systems on-demand with the pay-as-you-go (PAYG) pricing model, and opens the door for research in ARM64 systems at affordable costs.

In this paper, we evaluate the capacities of the Amazon Graviton Processor using a combination of micro benchmark and case studies, with comparisons to Intel Xeon E5 Processor and Intel Xeon Platium Processor. The micro benchmarks are performed to understand the performance characteristics of the

[1] https://aws.amazon.com/ec2/instance-types/a1/

[2] The A1 product family uses Amazon Graviton Processor @ 2.30 GHz, with 16 cores on the physical chip. The I3 product family uses Intel Xeon E5 2686 v4 Processor @ 2.30 GHz, with 18 cores on the physical chip. The M5 product family uses Intel Xeon Platinum 8175M Processor @ 2.50 GHz, with 24 cores on the physical chip.

[3] A star symbol (*) in the value indicates this particular instance type can support maximum performance for 30 minutes at least once every 24 hours. The same applies to network bandwidth.

978-1-7281-6095-5/20/$31.00 ©2020 IEEE
DOI 10.1109/CCGrid49817.2020.00-47

TABLE II: Individual Tests in UnixBench.

Test Name	Test Description
DH	Dhrystone 2 using register variables
WH	Double-Precision Whetstone
ET	Execl Throughput
FC1	File Copy 256 bufsize 500 maxblocks
FC2	File Copy 1024 bufsize 2000 maxblocks
FC3	File Copy 4096 bufsize 8000 maxblocks
PT	Pipe Throughput
CS	Pipe-based Context Switching
PC	Process Creation
SS1	Shell Scripts (1 concurrent)
SS8	Shell Scripts (8 concurrent)
SC	System Call Overhead

Amazon Graviton Processor. The case studies are performed to evaluate the feasibility of using the Amazon Graviton Processor for various horizontally scalable compute intensive workloads, including multi-tier web service, video transcoding and terabyte scale sorting. These experiments are carried out on AWS EC2 using the A1, I3 and M5 product families, with their detailed configuration presented in Table I. The specific contributions of this paper include:

- We use a combination of micro benchmark and Linux performance counters to analyze the performance of the target systems. We reveal that although the Amazon Graviton Processor demonstrates reasonable performance in various tests, the lack of L3 cache and the slower memory access speed prevent the A1 instances from achieving better performance for compute-intensive workload with intensive demand on memory.
- We verify that the Amazon Graviton Processor is capable of handling various horizontally scalable compute-intensive workloads (multi-tier web service, video transcoding and terabyte scale sorting) at scale in a cost effective way. The test worker fleet has up to 1600 vCPU cores, which is by far the largest ARM64 clusters that has been reported. In our evaluations, the Amazon Graviton Processor achieves the same price-performance in multi-tier web service, up to 37% cost saving in video transcoding, and up to 65% cost saving in terabyte scale sorting. This opens the door for further optimization for workloads with both cost and deadline constraints.

The rest of this paper is organized as follows. Section II describes the method and tools used in the micro benchmark framework. In Section III, we analyze and discuss the micro benchmark results of a1 instances in comparison with those of i3 instances. Section IV presents large-scale evaluations on Graviton's capability in handling horizontally scale compute-intensive workloads, with comparison to Intel Xeon processors. Section V reviews related work, followed by our conclusions in Section VI.

II. Micro Benchmark Design

We use UnixBench 5.1.3 [4] for the micro benchmark study. UnixBench includes a number of individual tests, with their details listed in Table II. Each individual test was designed to evaluate the performance of certain components on a computer system. When there are more than one CPU cores on the system, UnixBench executes the same test twice in sequence, one with a single thread and the other with multiple threads, where the number of threads equals to the number of CPU cores. These tests are executed with a fixed time, and the test result is reported as index score. The index score reflects how much work is done during the fixed time, as compared to the amount of work done during the same fixed time on a SPARCstation 20-61 [5]. The benchmark suite produces a single-thread index score from the single-thread execution, and a multi-thread index score from the multi-thread execution. When we need to execute a particular test or a set of tests with a single thread only, we modify the test script **Run** and replace the value of **$numCpus** with 1.

The benchmark tests are executed in conjunction with *perf* - the performance analysis tool for Linux[6]. We use the "perf stat" command to collect performance counters, including context switches, CPU cycles, instructions, cache misses, etc. These are hardware level performance counters, which can be enabled via certain instructions. Once enabled, they can be read by accessing specific processor registers. Due to the multi-tenant nature in public clouds, on AWS EC2 instances access to these processor registers is intentionally disabled for security reasons. It is an undocumented feature that these processor registers are made available on very few instance types running on dedicated hosts [7]. Our extensive testing reveals that this feature is enabled on i3.16xlarge and a1.4xlarge instances running on dedicated hosts. As such, we choose i3.16xlarge and a1.4xlarge instances running on dedicated hosts as the test bed for our benchmark study.

Since a1.4xlarge and i3.16xlarge instances have different number of vCPU cores and different amount of memory, we need to find a way to make a fair comparison using the index scores reported by UnixBench. In UnixBench, each point in the index score represents a fixed amount of work. Therefore, we can calculate the cost of doing the same amount of work in terms of performance counters, as below:

$$COST = \frac{Performance\ Counter}{Index\ Score} \quad (1)$$

Further, we use relative score (**RS**) to represent how one system performs as compared to another reference system, as below:

$$RS = \frac{Index\ Score}{Index\ Score\ on\ Ref.\ System} \quad (2)$$

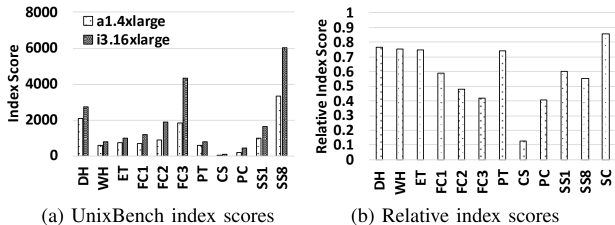
Memory and cache has a significant impact on the performance of software applications. We use *lstopo* and the proc filesystem */proc* to understand the memory topology on the target system – including the size of L1, L2, L3 caches and the

[4]https://github.com/kdlucas/byte-unixbench

[5]A SPARC workstation manufactured by Sun Microsystems in 1994.

[6]https://perf.wiki.kernel.org/index.php/Main_Page

[7]http://www.brendangregg.com/blog/2017-05-04/the-pmcs-of-ec2.html

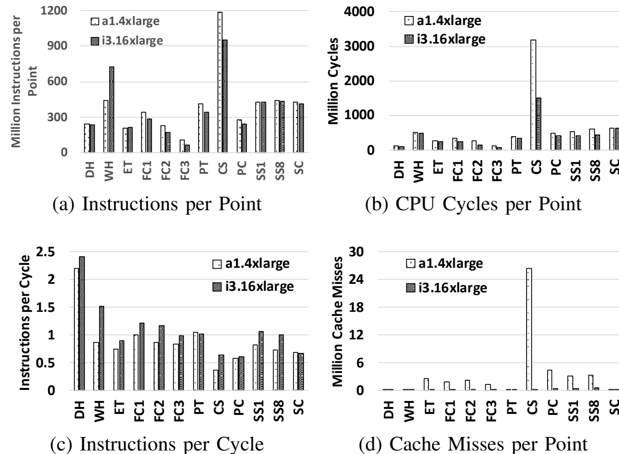


(a) UnixBench index scores (b) Relative index scores

Fig. 1: Single-thread UnixBench index scores on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.

TABLE III: Memory topologies on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.

	L1 Data	L1 Instruction	L2 Unified	L3 Unified
a1.4xlarge	32KB	48 KB	2048 KB	N/A
i3.16xlarge	32 KB	32 KB	256 KB	2 x 45 MB



(a) Instructions per Point (b) CPU Cycles per Point

(c) Instructions per Cycle (d) Cache Misses per Point

Fig. 2: Cost analysis of the single-thread UnixBench test results on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.

amount of DRAM. We use the memory bandwidth benchmark *mbw* [8] to evaluate the speed of memory access.

The experiments are performed on AWS EC2 in the us-east-1 (N. Virginia) region. The operating system used in the evaluation is Amazon Linux 2. Each test is executed three times, with each test run being executed on a newly launched EC2 instance. The data reported in this paper is the average of the index scores obtained from three independent test runs. Since the EC2 instances are launched on dedicated hosts, and only one EC2 instance is being launched and tested at a time, this can be treated as a single-tenant environment, and is not exposed to the impact of the noisy-neighbour effect [7], [8] commonly observed in a multi-tenant environment.

The hosts for the A1 product family has 1 physical CPU with 16 physical cores, providing 16 vCPU cores for the EC2 instances. The hosts for the I3 product family has 2 physical CPU's with a total number of 36 physical cores. With hyper-threading enabled, a physical host provides 64 vCPU cores for the EC2 instances. This utilizes 32 physical cores on the host, leaving 4 physical cores for the underlying virtualization management software.

III. Micro Benchmark Results

A. Single-Thread Benchmark

Figure 1a presents the single-thread UnixBench test results on a1.4xlarge and i3.16xlarge instances running on dedicated hosts. Despite the fact that the CPU cores on both systems have the same clock rate (2.30 GHz), the index score on a1.4xlarge is smaller than that on i3.16xlarge in all individual tests. Figure 1b presents the relative scores achieved on a1.4xlarge, with i3.16xlarge as the reference system. In the best case, a1.4xlarge achieves a relative score of 86% in system call overhead (SC). In the worse case, a1.4xlarge achieves a relative score of 13% in context switch (CS).

[8]https://github.com/raas/mbw

Figure 2 presents the cost analysis of the single-thread UnixBench test results on a1.4xlarge and i3.16xlarge instances running on dedicated hosts. On the instruction level, a1.4xlarge needs more instructions to achieve a point in all tests except for Whetstone and Execl Throughput (Figure 2a). On the one hand, RISC processors have a smaller set of instructions with few addressing nodes than CISC processors. On the other hand, in CISC processors, several low-level operations can be executed with a single complex instruction (for example, loading data from memory, performing an arithmetic operation, then storing the result to memory). As such, it is reasonable that the same amount of work (a point) requires more instructions on a1.4xlarge than on i3.16xlarge. Since the CPU's on a1.4xlarge and i3.16xlarge have the same clock rate, one might expect when more instructions are needed to achieve a point then more CPU cycles are needed to execute those instructions. Comparing Figure 2b and 2a, this is true in general, but with the exception of Whetstone and Execl Throughput. In these two tests, i3.16xlarge executes more instructions with less CPU cycles. This indicates that for different tests, on average different number of instructions are executed in one CPU cycle. Figure 2c presents the number of instructions per cycle (IPC) observed in the single-thread UnixBench tests. The i3.16xlarge instance achieves higher IPC in all tests except for Piped Throughput and system call overhead (SC).

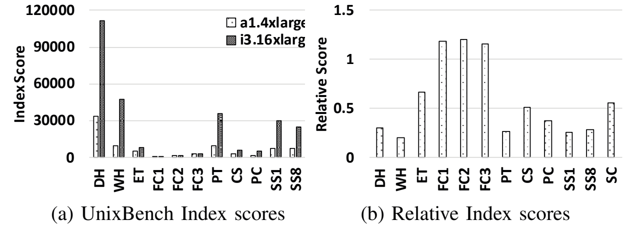
The biggest influence on IPC comes from memory hierarchy. Although memory access is fast, it is still relatively slow as compared to the processor. This is commonly known as the *memory wall* [9]. Most modern computers use a hierarchy of caches between the processor and the DRAM. The first level cache (L1) is the fastest and smallest, and is usually separate for data (L1d) and instructions (L1i). The second level cache (L2) is slower than L1, but larger, and is usually unified for data and instructions. Both L1 and L2 are specific to a

particular processor core, and are not shared among different processor cores on the same chip. Some processors have a third level cache (L3), which is slower but larger than L2, and is usually shared among different processor cores on the same chip. When the processor requests a particular data item, it checks the cache (which is faster) before accessing DRAM (which is slower). If the data is present in the cache, this is called a *cache hit*, then computation can be speed up by not accessing the slower DRAM. If the data is not present at all levels in the cache hierarchy, this is called a *cache miss*, then computation is slowed down because the data needs to be fetched from the slower DRAM. As shown in Table III, a1.4xlarge has a larger L1i cache and a larger L2 cache, but i3.16xlarge has an extra L3 cache. For i3.16xlarge, there are two CPU chips on the physical host, each chip has a 45 MB L3 cache, which is shared by all CPU cores on the same chip. Further more, memory benchmark *mbw* indicates i3.16xlarge has much higher memory access speed. In particular, i3.16xlarge achieves 5440 MB/s in memory copy, a1.4xlarge achieves 3960 MB/s in the same test, which is only 73% as compared to i3.16xlarge. Such difference in memory access speed correlates well with the 77%, 75%, 75% and 74% relative scores observed in the Dhrystone, Whetstone, Execl Throughput, and Pipe Throughput. The impact of memory hierarchy is reflected in the significant amount of cache misses observed on a1.4xlarge when the relative index score is below 0.6 (Figures 1b and 2d). This leads to the conclusion that cache misses due to the lack of L3 cache, as well as the slower memory access speed, are the primary reasons for the under-performance observed on a1.4xlarge in these tests.

A good example on the impact from memory hierarchy is the test on context switch (CS). As shown in Figure 2d, in this particular test the amount of cache misses on a1.4xlarge is 25 times more than the amount of cache misses on i3.16xlarge. The frequent need to access the slower DRAM prevents the instructions from being executed efficiently, which is reflected in the smaller number of instructions per cycle in Figure 2c. On a1.4xlarge, the test only requires 25% more instructions to achieve a point (Figure 2a), but demands more than twice the amount of CPU cycles (Figure 2b). As a result, the test achieves the lowest relative score (13%) among all tests. These observations are in accordance with the observations made by Li et. al. [10] and their conclusion that the overhead of cache misses has a substantial impact on the cost of context switch.

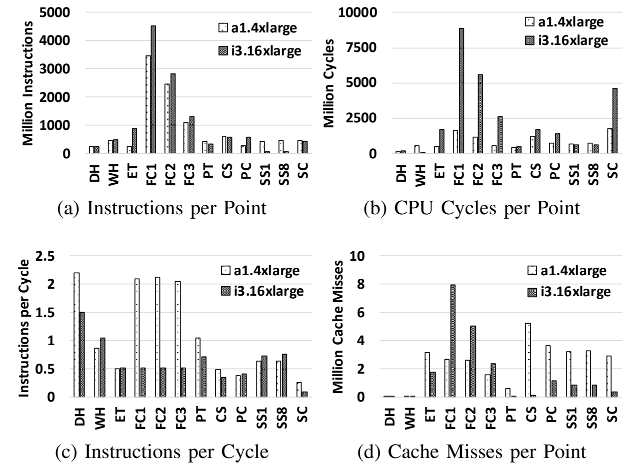
B. Mult-Thread Benchmark

Figure 3a presents the multi-thread UnixBench test results on a1.4xlarge and i3.16xlarge instances running on dedicated hosts. Figure 3b presents the relative scores achieved on a1.4xlarge, with i3.16xlarge as the reference system. In most tests, the index score on a1.4xlarge is significantly lower than i3.16xlarge. This is reasonable because i3.16xlarge has 64 vCPU cores while a1.4xlarge has only 16 vCPU cores. However, in all three file copy tests, the index score on a1.4xlarge is approximately 20% higher than i3.16xlarge. In our tests a1.4xlarge and i3.16xlarge have EBS volumes of



(a) UnixBench Index scores (b) Relative Index scores

Fig. 3: Multi-thread UnixBench index scores on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.



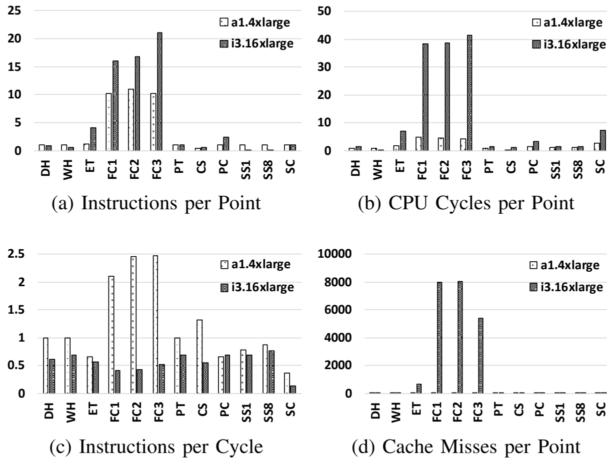
(a) Instructions per Point (b) CPU Cycles per Point

(c) Instructions per Cycle (d) Cache Misses per Point

Fig. 4: Cost analysis of the multi-thread UnixBench test results on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.

the same type, size, and disk I/O capacity. This suggests that file copy performance is controlled by disk I/O capacity. In the single-thread test, the disk I/O capacity is not fully utilized. In this case, i3.16xlarge achieves better performance with less cache misses (Figure 2d). In the multi-thread test, the disk I/O capacity becomes fully utilized by the combined workload. Since i3.16xlarge has more worker threads, there exists a higher level of competition for disk I/O among the worker threads, leading to the increased level of performance degradation.

Figure 4 presents the cost analysis of the multi-thread UnixBench test results on a1.4xlarge and i3.16xlarge instances running on dedicated hosts. Figure 5 presents the ratio between the costs of achieving a point in the multi-thread UnixBench test and the single-thread UnixBench test. As shown in Figure 5d, in the file copy tests, significantly more cache misses are observed on i3.16xlarge. On i3.16xlarge, each CPU core has its own L1 and L2 caches, but the L3 cache is shared among all 18 CPU cores on the same chip. In the single-thread test, the test thread on i3.16xlarge has access to 256 KB L2 cache and 45 MB L3 cache on one physical processor. The compensation from L3 cache allows i3.16xlarge to achieve better performance with less cache misses. In the multi-thread test, the 90 MB L3 cache (45 MB on each physical processor)



(a) Instructions per Point (b) CPU Cycles per Point

(c) Instructions per Cycle (d) Cache Misses per Point

Fig. 5: The ratio between the cost of achieving a point in the multi-thread UnixBench test and the cost of achieving a point in the single-thread UnixBench test on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.

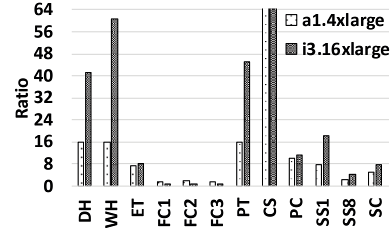
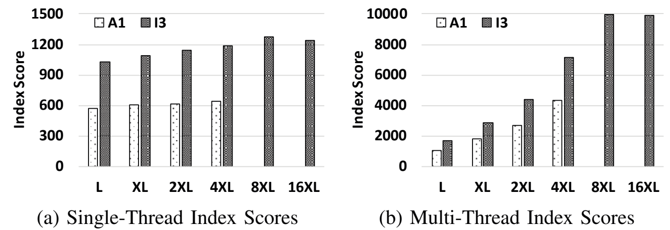


Fig. 6: The ratio between multi-thread and single-thread index scores on a1.4xlarge and i3.16xlarge instances running on dedicated hosts.



(a) Single-Thread Index Scores (b) Multi-Thread Index Scores

Fig. 7: Vertical scaling and UnixBench index scores.

is saturated by the combined workload from 64 threads, leading to the significant cache misses on the L3 level shown in Figure 5d. At the same time, due to the saturation of in disk I/O, additional idle instructions (*iowait*) are now needed to perform the same amount of work in the multi-thread tests. The competition for disk I/O occurs on both a1.4xlarge and i3.16xlarge, but the situation is worse on i3.16xlarge because there are 64 threads instead of 16. As a result, i3.16xlarge now needs more instructions to achieve a point in these tests (Figure 5a).

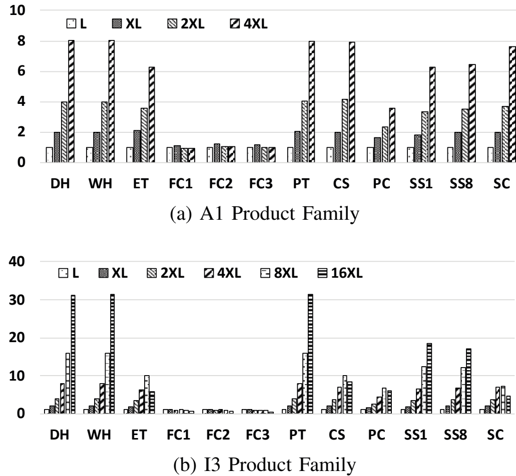
Another notable observation is, in terms of instructions per cycle (IPC), i3.16xlarge achieves less IPC in all multi-thread tests than in the single-thread tests (the ratio is smaller than 1 in Figure 5c). This indicates that in the multi-thread tests on i3.16xlarge each thread achieves less performance than what is achieved in the single-thread test. Again, this is largely due to the increased level of cache misses when the 64 threads compete for the 90 MB L3 cache. For a1.4xlarge, the ratio for IPC in the multi-thread and single-thread tests is 1 for the integer processing (DH), floating point processing (WH) and pipe throughput (PT) tests. Because of the lack of a shared L3 cache, in these multi-thread tests each thread does not have any impact on other threads running in parallel. When the number of threads increases, linear performance gain is observed on the system level. This is reflected in Figure 6 – for a1.4xlarge, with 16 threads, the multi-thread index score is 16 times as big as the single-thread index score. Also, for a1.4xlarge, the ratio for IPC in the multi-thread and single-thread tests is greater than 1 for the file copy tests. As discussed in the previous paragraph, the increased instructions are the result of *iowait* and do not lead to performance gain in each thread.

Another notable observation in Figure 6 is context switch (CS) benefits the most from multi-thread processing - performance grows faster than the additional of worker threads. On a1.4xlarge, there is approximately 25 million cache misses per point in the single-thread test, which is reduced to 5 million in the multi-thread test. Similar level of reduction in cache misses is also observed on i3.16xlarge. Since cache misses have a substantial impact on context swith, the reduction in cache misses results in better performance.

C. Vertical Scaling

Figure 7 presents the system benchmark index scores for all instance types in the A1 and I3 product families running on dedicated hosts. In the single-thread tests (Figure 7a), there exists only some insignificant increase in the index scores achieved on bigger instance types. This is reasonable because the test thread only utilizes one of the multiple vCPU cores on the system. The insignificant increase in index score is largely due to the fact that system processes are now off loaded to other vCPU cores, while having an abundance of memory does not significantly improve the benchmark score. In the multi-thread tests (Figure 7b), when the number of vCPU cores doubles, approximately 40% to 60% increase in index score is observed. The only exception is i3.16xlarge – a slight performance lost is observed as compared to i3.8xlarge. On i3.8xlarge there are only 32 threads running on two physical processors with 36 physical cores and 90 MB L3 cache. The CPU and the L3 cache experiences less pressure, as compared to the tests on i3.16xlarge where 64 threads are competing for the same amount of computing resources. As such, i3.8xlarge achieves slightly better benchmark score than i3.16xlarge. However, the hardware performance counters are disabled on i3.8xlarge and we are unable to verify this hypothesis.

Figure 8 presents the relative multi-thread index scores for the individual tests, with the reference system being a1.large



(a) A1 Product Family

(b) I3 Product Family

Fig. 8: Relative multi-thread index scores on A1 and I3 product families.

and i3.large respectively. In the A1 product family (Figure 8a), in general the index score grows linearly with the growth in the number of vCPU cores. However, in the file copy tests, a1.xlarge achieves only very slight performance gain as compared to a1.large, while both a1.2xlarge and a1.4xlarge exhibit some slight performance lost. In the I3 product family (Figure 8b), linear performance are observed in Dhrystone, Whetstone, and pipe throughput. For the file copy tests, i3.xlarge instance achieves only very slight performance gain as compared to i3.large, while performance lost occurs on i3.2xlarge and bigger instances. On i3.16xlarge, performance lost is also observed in execl throughput, context switch, process creation, and system call overhead.

IV. Case Studies

In this section, we use multi-tier web service, video transcoding, and tera-byte scale sorting as cases studies to evaluate the potential of using the A1 product family to handle horizontally scalable compute-intensive workloads at scale, with comparison to the I3 and M5 product families. According to AWS, both A1 and M5 are "general-purpose" product families, providing a balance of compute, memory and networking resources, and can be used for a variety of diverse workloads. I3 is a "storage optimized" product family, which is designed for workloads that require high sequential read and write access to very large data sets on local storage. However, the local storage is not being utilized in our case studies.

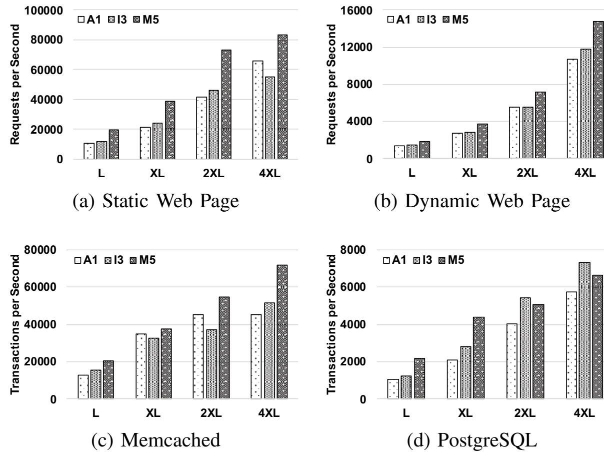
In the case studies, we only evaluate the large, xlarge, 2xlarge and 4xlarge instance sizes, because other instance sizes are not commonly available for the A1, I3 and M5 product families. All of the EC2 instances are launched with shared tenancy – i.e, not on dedicated hosts. As such, the observations are exposed to the impact of the noisy-neighbor effect in a multi-tenant environment.

Application performance depends on many factors including the operating system itself, different configurations on the operating system level, as well as different configurations for the application being tested. In the case studies, we simulate the behavior of the ordinary public cloud users in that we use the operating system and application as they are, with only a minimum amount of custom configurations. Performance gain might be achieved by applying "beter" runtime configurations, while performance lost might be observed when "improper" runtime configurations are used. In a public cloud environment, application performance is also subjected to the performance variation commonly observed in a multi-tenant environment.

A. Multi-Tier Web Service

Multi-tier web service is a typical use case in public clouds. A multi-tier web service usually includes the following components: (a) a web server as the front end, (b) a database server to persist user data, and (c) a cache layer for session sharing and caching database query results. In this case study, we first evaluate the performance of these individual components, then evaluate the performance of a sample dynamic web page that integrates these components. More specifically, our tests include the following:

- Static web page - Fetch a static web page 6,400,000 times from the web server using 128 threads and report the number of requests per second. The static web page being tested is the default index.html that comes with the Apache2 web server.
- Dynamic web page - Fetch a dynamic web page 6,400,000 times from the web server using 128 threads and report the number of requests per second. The dynamic web page is in PHP, which calls the phpinfo() function to display the configurations in the PHP run-time environment.
- In-memory caching - Perform 6,400,000 transactions against the Memcached server using 128 threads and report the number of transactions per second. Each transaction is a combination of one SET and one GET operation. The SET command writes to the cache with a 10-byte key and a 256-byte value, while the GET command reads the value back from the cache using the same key.
- Relational database - Perform 6,400,000 transactions against the PostgreSQL server using 128 threads and report the number of transactions per second. Each transaction includes three SELECT commands, one INSERT command and one UPDATE command. The test table has 10,000,000 records, with the average record size being 16 KB. In this particular test, we use 6 x 5000 GB EBS volumes to form a RAID0 disk array to provide the maximum level of disk I/O capacity (80,000 IOPS on each EC2 instance).
- Integration test - Fetch a dynamic web page 6,400,000 times from the web server using 128 threads and report the number of requests per second. The dynamic web page creates a new session and writes the session information into Memcached running on localhost, reads the



(a) Static Web Page (b) Dynamic Web Page

(c) Memcached (d) PostgreSQL

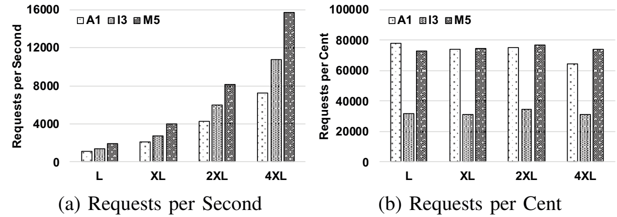
Fig. 9: The performance of individual components in a multi-tier web application.

latest 10 records from PostgreSQL running on localhost, then generates the HTML response based on the result set.

In this test, we use Ubuntu 18.04.3 LTS as the operating system, with Apache2 version 2.4.29, PHP version 7.3, Memcached version 1.5.6, and PostgreSQL version 10.10. The client machine is m5.24xlarge with 6000 GB EBS volume, which provides up to 16,000 IPOS disk I/O capacity. The test client uses persistent connections to interact with the servers, otherwise the client machine will gradually run out of ephemeral ports and fail to establish new connections.

Figure 9 presents the performance of individual components in a multi-tier web application. In terms of serving static web pages (Figure 9a), M5 performs best, achieving 30% to 50% more requests per second than A1 and I3. I3 performs slightly better than A1, except for the 4xlarge instance type. In terms of serving dynamic web pages in PHP (Figure 9b), M5 performs best, achieving 15% to 25% more requests per second than A1 and I3. I3 performs slightly better than A1, but the advantage is insignificant. In terms of in-memory caching with Memcached (Figure 9c), M5 performs best, achieving 15% to 40% more transactions per second than A1 and I3. I3 performs better than A1 on large and 4xlarge instance types, while A1 performs better than I3 on xlarge and 2xlarge instance types. In terms of relational database with PostgreSQL (Figure 9d), M5 performs best on large and xlarge instance types, while I3 performs best on 2xlarge and 4xlarge instance types.

Since M5 uses a CPU with higher CPU clock rate, it is not surprising that M5 performs best in almost all component-level tests. The only exception being the relational database evaluation, which is at the same time CPU-intensive, memory-intensive and disk I/O intensive. On large and xlarge instance types, M5 achieves better performance because CPU is the bottleneck. On 2xlarge and 4xlarge instance types where CPU is no longer the bottleneck, I3 achieves better performance because it has more memory and disk I/O bandwidth.



(a) Requests per Second (b) Requests per Cent

Fig. 10: The performance of the demo multi-tier web application.

I3 and A3 have the same CPU clock rate, but I3 has more memory with faster memory access speed. Therefore, it is not surprising that I3 performs slights better than A3 in almost all component-level tests, except for (a) the static web page test for the 4xlarge instance type, and (b) the in-memory caching test for the xlarge and 2xlarge instance type.

Figure 10a presents the performance of the demo multi-tier web application. M5 achieves the best performance in the integration test, which is expected considering its outstanding performance in all component-level tests. This is followed by I3, while A1 achieves the worse performance.

In a real-life use case one needs to consider both the performance and the cost to achieve the desired performance. Figure 10b presents the number of requests that can be achieve with 1 US cent in the integration test. Surprisingly, A1 achieves higher requests per US cent than M5 on the large instance type (by 7%) , the same level of requests per US cent as M5 on xlarge and 2xlarge instance types, and slightly less requests per US cent than M5 on the 4xlarge instance type (by 15%). I3 achieves much smaller requests per US cent than A1 and M5 on all instance types (approximately 60% less). This is because I3 is a storage-optimized product family. The pricing of I3 includes the instance-store volumes that is attached to the underlying physical host, but the instance-store volumes are not being utilized in this particular test.

Most modern web services running in public clouds are designed to be horizontally scalable. With techniques such as auto scaling, public cloud users can build horizontally scalable websites using the A1 product family, which can achieve the same price-performance as compared to using the M5 product family.

B. *Video Transcoding*

Video transcoding is a common use case for websites and mobile applications providing video sharing service. Figure 11 presents a typical use case of video transcoding for a video sharing website. When the end user uploads a video to the video sharing service, the video is stored in a bucket on AWS S3, a record is written to the database, and a transcoding job is published into a message queue. The uploaded videos are then transcoded in the background with a batch processing system. In this case study, we use the large-scale video transcoding system described in [11] for benchmarking. With a producer/consumer design, the transcoder nodes are stateless in that

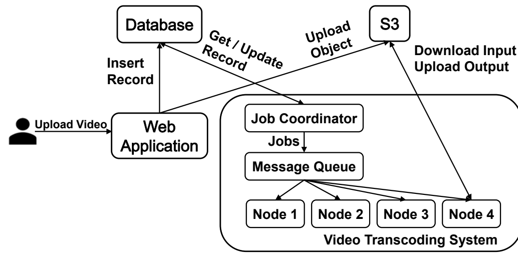
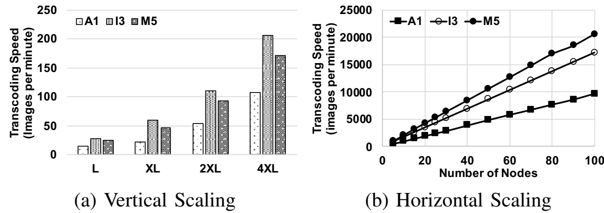


Fig. 11: Architecture of the video transcoding application.



(a) Vertical Scaling (b) Horizontal Scaling

Fig. 12: Scalability of the video transcoding application.

they (a) do not persist any job state on themselves, and (b) do not communicate with each other. On each transcoder node, multiple threads are launched to do the actual work, where the number of threads equals to the number of vCPU cores available on the node. Each thread performs the following tasks in sequence: (a) obtains one job from the message queue, (b) downloads the video clip from S3 to local disk, (c) uses FFmpeg [9] to convert the video clip from MP4 format to WMV format, (e) uploads the output file from local disk back to S3, (f) deletes both the MP4 and WMV files from local disk, and (g) acknowledge to the message queue that the job is completed. The performance of the video transcoding application is measured by transcoding speed, i.e., the number of video clips processed in a minute.

We use 1,000,000 video clips as the test input. The lengths of the video clips vary from 6 to 117 seconds. The video clips are pre-staged to S3 and are evenly distributed in the message queue. A UUID is used as the object key for each video clip, which “hints” S3 to distribute the video clips in multiple partitions for improved performance.

Figure 12 presents the video transcoding test results on the A1, I3 and M5 product families. As shown in Figure 12a, the transcoding speed increases when the EC2 instance becomes bigger. There is roughly a linear relationship between transcoding speed and the number of vCPU cores. For the same instance size, the A1 product family has the slowest transcoding speed, while the I3 product family has the fastest transcoding speed. This suggests that video transcoding is more memory-intensive than CPU-intensive, because I3 has slower CPU clock rate but more memory than M5. M5 has a faster CPU clock rate than A1, therefore achieves better transcoding performance than A1.

[9] https://www.ffmpeg.org

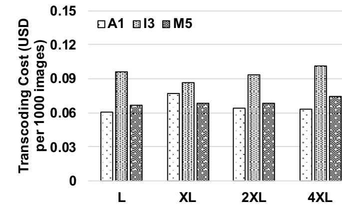


Fig. 13: The cost of transcoding 1000 video clips.

We use the 4xlarge instance types in the horizontal scaling evaluation. For each instance type, we gradually increase the number of nodes from 5 to 100, with up to 1600 vCPU cores in the worker fleet. As shown in Figure 12b, for all three instance types, transcoding speed increases linearly with the addition of worker nodes. Although a1.4xlarge has the slowest transcoding speed per node, such linear scaling behavior allows it to achieve the desired level of performance by adding more nodes into the worker fleet.

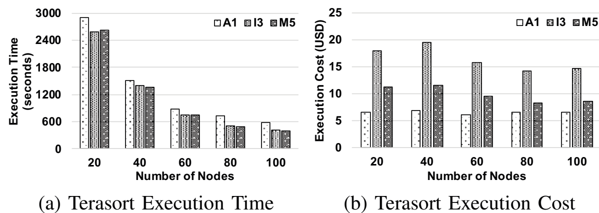
Figure 13 presents the cost of transcoding 1000 video clips on different instance types among the A1, I3 and M5 product family. The I3 product family has the highest transcoding cost. The A1 product family exhibits some modest cost saving as compared to the M5 product family. In large-scale deployments with horizontal scaling, achieving the same transcoding speed with a1.4xlarge can lead to 15% cost saving as compared to m5.4xlarge, and 37% cost saving as compared to i3.4xlarge.

C. Terabyte Scale Sorting

In recent years, big data analysis is steadily gaining attention in the information technology industry, with Hadoop MapReduce being one of the most popular tool for such analysis. Although there isnt an exact size that qualifies a dataset as big data label, many big data repositories are measured in terabytes or petabytes. One of the many problems in big data analysis is the sorting of a very large data set, for example, finding out a few IP addresses that produce the most number of 404 errors from a large amount of web server access logs in a near real-time fashion.

We use the TeraSort application in the Hadoop examples to performing sorting on a 1 TB dataset on Hadoop clusters with a1.4xlarge, i3.4xlarge, and m5.4xlarge instance types. The Hadoop clusters includes one master node and a certain number of worker nodes. Both the master node and the worker nodes have a 500 GB EBS volume, offering up to 1500 IOPS and up to 250 MB/s throughput. The master node runs the YARN resource manager and the HDFS namenode, while the worker nodes run the YARN node manager and the HDFS datanode. We intentionally configure the HDFS replication factor to be 1. The sorting process produces 2 TB disk reads, 2 TB disk writes, with 1 TB data transfer between different data nodes across the network. The data to be sorted is divided into 1024 splits, with 1 GB data in each split.

Figure 14a presents the execution time for sorting 1 TB data with Hadoop Terasort. For all instance types, the execution time decreases when the number of nodes increases. The



(a) Terasort Execution Time (b) Terasort Execution Cost

Fig. 14: Sorting 1 TB data with Hadoop Terasort.

execution time on I3 and M5 are on approximately the same level. This indicates that (a) the sorting algorithm does not fully consume the memory available on the worker nodes, therefore more memory on I3 does not lead to increased performance, and (b) the sorting algorithm is less CPU-intensive, because higher CPU clock rate on M5 does not lead to increased performance. On I3 and M5, the progress of the sorting process is controlled by disk I/O performance. However, the execution time on A1 is significantly greater than the execution time on I3 and M5, which can be attributed to the slower memory access rate and the lack of L3 cache, as described in Section III-A.

Figure 14b presents the execution cost for sorting 1 TB data with Hadoop Terasort. I3 has the highest execution cost, followed by M5, while A1 has the lowest execution cost. As compared to I3, A1 achieves 54% to 65% cost reduction. As compared to M5, A1 achieves 20% to 41% cost reduction. Since I3 and M5 have shorter execution time, this opens the door for further optimization for workloads with both both cost and deadline constraints. That is, one can choose to execute a certain workload on A1 with longer execution time but smaller execution cost, or to execute the same workload on I3 and M5 with shorter execution time but greater execution cost.

V. Related Work

The energy consumption of data centers has become a key sustainability issue [1]–[3]. From the processor architectural point of view, there have been two distinctive approaches. One approach is using lower-power CISC processors such as the Intel Atom product family, which is commonly referred to as "wimpy nodes", e.g., [4], [12]. This is effective in handling I/O-heavy but computationally-lightweight applications such as key-value stores, but encounters performance issues when the applications are compute-intensive [4], [12]–[17].

The other approach is using RISC processors such as the ARM product family. Most of these attempts used 32-bit Cortex-A7/A8/A9 processors, and the evaluations were performed on a single-node [18]–[24]. The above-mentioned research attempts focused primarily on a single-node. Rajovic et. al. [25] introduced Tibidabo – the first experimental HPC cluster with 128 Cortex-A9 nodes. With a total number of 256 CPU cores, the experimental cluster achieved 120 MFLOPS/W on HPL. The authors considered Tibidabo to be "competitive" as compared to AMD Operton 6128 and Intel Xeon X5660-based systems. Aakash et. al. [26] present a video analysis application running on an ARM64 server, which is designed for automatic license plate recognition and urban scene classification. Jayanth et. al. [27] compares the performance of an ARM64 server (AMD Opteron A1100 SoC) with an x64 server (AMD Opteron 3380) for big data workloads. Adrian et. al. [28] adopted a benchmarking approach to evaluate the potential of using ARM processors for HPC applications. The system being evaluated was HPE Apollo 70, an ARMv8 based cluster with up to 1024 CPU cores and 4096 GB memory. The authors concluded that the performance of HPE Apollo 70 was as good as, or better, than that of well-established platforms with Intel Xeon processors. This is by far the largest ARM cluster that has been reported in literature. Daniel et. al. [29] performed a comprehensive literature review on previous attempts in using ARM processors in HPC.

There exists a large collection of literature on evaluating the performance of computer systems in general [9], [10], [30]–[33]. The works done with ARM processors usually focus on comparing the performance and power consumption of ARM32 devices and x64 devices. The ARM devices being used are usually Cortex-A8 and Cortex-A9. The x64 devices used include AMD Operton, Intel Atom, Intel Core2, and Intel Xeon. Aroca et. al. [18] studied web server and database server workloads. They reported that the ARM devices were 3 to 4 time more power efficient than x64 systems under different load situations. Tudor et. al. [19] studied HPC, web server, and financial analytic workloads. They observed that low-power CPU cores did not promise energy-efficient executions for server workloads. On the contrary, significantly longer execution time and higher energy cost were observed in HPC due to resource imbalances. Blem [20] studied mobile client, desktop application, web server and database server workloads. They concluded that micro-architecture – large cache, accurate branch prediction and bigger issue in particular – had the most impact on performance. Ou et. al [21] studied in-memory database, web server, and video transcoding workloads. They observed that the energy-efficiency ratio of the ARM cluster against the Intel workstation varies from 9.5 to 1.2, depending on the type of workload. They concluded that ARM clusters were only feasible for computationally lightweight applications. For compute-intensive workloads, the benefit of using ARM cluster diminished progressively.

Our paper differs from existing literature in that (a) multiple compute-intensive workload are used in the empirical study, (b) the scale of the worker fleet is large enough to handle real-life workloads at scale, and (c) using ARM64 processor on public cloud is an emerging trend, which has not been studied before.

It should be noted that in December 2019 AWS addressed the L3 cache issue in the Amazon Graviton2 Processor [10], which includes a 32 MB L3 cache. At the time of submitting this paper, the authors were not aware of the work done in AWS, nor was AWS aware of the work presented in this paper.

[10]https://aws.amazon.com/about-aws/whats-new/2019/12/announcing-new-amazon-ec2-m6g-c6g-and-r6g-instances-powered-by-next-generation-arm-based-aws-graviton2-processors/

VI. Conclusion

In this paper, we study the performance characteristics of the Amazon Graviton Processor, with comparison to Intel Xeon Processors. Using a combination of micro benchmark and Linux performance counters, we demonstrate that the lack of its L3 cache and the slower memory access speed can prevent the Amazon Graviton Processor from achieving better performance for compute-intensive workload with intensive demand on memory.

We use multi-tier web service, video transcoding, and terabyte scale sorting as cases studies to evaluate the potential of using the A1 product family on AWS EC2 for large-scale workloads, with comparison to the I3 and M5 product families. In our large-scale evaluations, the largest cluster includes 1600 vCPU cores, which is by far the lartest ARM64 cluster that has been reported. The single-node performance of the A1 product family is not as good as the single-node performance of the I3 and M5 product families. However, satisfactory processing capacity can be achieved by practicing horizontal scaling. We demonstrat that the A1 product family achieves the same price-performance in multi-tier web service, up to 37% cost saving in video transcoding, and up to 65% cost saving in terabyte scale sorting. This opens the door for further optimization for workloads with both both cost and deadline constraints.

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[2] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner, "United states data center energy usage report," 2016, lawrence Berkeley National Laboratory.

[3] L. C. Paolo Bertoldi, Maria Avgerinou, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," 2017, JRC Technical Report, European Union.

[4] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "Fawn: A fast array of wimpy nodes," in *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. ACM, 2009, pp. 1–14.

[5] D. Jaggar, "ARM architecture and systems," *Proceedings of the 1997 Annual IEEE/ACM Internal Symposium on Microarchitecture (Micro)*, vol. 17, no. 4, pp. 9–11, 1997.

[6] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Price elasticity in the enterprise computing resource market," *IEEE Cloud Computing*, vol. 3, no. 1, pp. 24–31, 2016.

[7] A. Gulati, G. Shanmuganathan, A. M. Holler, and I. Ahmad, "Cloud scale resource management: Challenges and techniques." vol. 11, pp. 3–3, 2011.

[8] P. Leitner and J. Cito, "Patterns in the chaosa study of performance variation and predictability in public iaas clouds," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 3, p. 15, 2016.

[9] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[10] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 2.

[11] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Scalable video transcoding in public clouds," in *Proceedings of the 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2019, pp. 70–75.

[12] D. Schall and V. Hudlet, "Wattdb: an energy-proportional cluster of wimpy nodes," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 2011, pp. 1229–1232.

[13] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini, "An energy case for hybrid datacenters," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 76–80, 2010.

[14] W. Lang, J. M. Patel, and S. Shankar, "Wimpy node clusters: What about non-wimpy workloads?" in *Proceedings of the 6th International Workshop on Data Management on New Hardware*. ACM, 2010, pp. 47–55.

[15] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis, "Towards energy-efficient database cluster design," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1684–1695, 2012.

[16] P. Stuedi, A. Trivedi, and B. Metzler, "Wimpy nodes with 10gbe: leveraging one-sided operations in soft-rdma to boost memcached," in *Proceedings of the 2012 USENIX Annual Technical Conference (ATC)*, 2012, pp. 347–353.

[17] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin servers with smart pipes: designing soc accelerators for memcached," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 36–47.

[18] R. V. Aroca and L. M. G. Gonçalves, "Towards green data centers: A comparison of x86 and arm architectures power efficiency," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, pp. 1770–1780, 2012.

[19] B. M. Tudor and Y. M. Teo, "On understanding the energy consumption of arm-based multicore servers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1. ACM, 2013, pp. 267–278.

[20] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures," in *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 1–12.

[21] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Yla-Jaaski, and P. Hui, "Energy-and cost-efficiency analysis of arm-based clusters," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2012, pp. 115–123.

[22] N. Rajovic, P. Carpenter, I. Gelado, N. Puzovic, and A. Ramirez, "Are mobile processors ready for hpc," in *Proceedings of the 2013 IEEE/ACM Supercomputing Conference*, 2013.

[23] A. Gutierrez, M. Cieslak, B. Giridhar, R. G. Dreslinski, L. Ceze, and T. Mudge, "Integrated 3d-stacked server designs for increasing physical density of key-value stores," in *ACM SIGPLAN Notices*, vol. 49, no. 4. ACM, 2014, pp. 485–498.

[24] M. Shahrad and D. Wentzlaff, "Towards deploying decommissioned mobile devices as cheap energy-efficient compute nodes," in *Proceedings of the 9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2017.

[25] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez, "Tibidabo: Making the case for an arm-based hpc system," *Future Generation Computer Systems*, vol. 36, pp. 322–334, 2014.

[26] A. Khochare, P. Ravindra, S. P. Reddy, and Y. Simmhan, "Distributed video analytics across edge and cloud using echo," in *Proceedings of the 2017 International Conference on Service-Oriented Computing (ICSOC)*, 2017.

[27] J. Kalyanasundaram and Y. Simmhan, "Arm wrestling with big data: A study of commodity arm64 server for big data workloads," in *Proceedings of the 24th IEEE International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 203–212.

[28] A. Jackson, A. Turner, M. Weiland, N. Johnson, O. Perks, and M. Parsons, "Evaluating the arm ecosystem for high performance computing," *arXiv preprint arXiv:1904.04250*, 2019.

[29] D. Yokoyama, B. Schulze, F. Borges, and G. Mc Evoy, "The survey on arm processors for hpc," *The Journal of Supercomputing*, pp. 1–34, 2019.

[30] L. W. McVoy, C. Staelin *et al.*, "lmbench: Portable tools for performance analysis." in *Proceedings of the 1996 USENIX Annual Technical Conference (ATC)*. San Diego, CA, USA, 1996, pp. 279–294.

[31] A. Yasin, "A top-down method for performance analysis and counters architecture," in *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 35–44.

[32] M. Becker and S. Chakraborty, "Measuring software performance on linux," *arXiv preprint arXiv:1811.01412*, 2018.

[33] R. Azimi, T. Fox, W. Gonzalez, and S. Reda, "Scale-out vs scale-up: a study of arm-based socs on server-class workloads," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 3, no. 4, pp. 1–23, 2018.