Achieving Cost-efficient, Data-intensive Computing in the Cloud

In submission – do not distribute

Michael Conley
University of California, San Diego
mconley@cs.ucsd.edu

Amin Vahdat
Google Inc.
University of California, San Diego
vahdat@cs.ucsd.edu

George Porter
University of California, San Diego
gmporter@cs.ucsd.edu

Paper type: Research. Paper length: Long.

Abstract

Cloud computing providers have recently begun to offer high-performance virtualized flash storage and virtualized network I/O capabilities, which have the potential to increase application performance. Since users pay for only the resources they use, these new resources have the potential to lower overall cost. Yet achieving low cost requires choosing the right mixture of resources, which is only possible if their performance and scaling behavior is known.

In this paper, we present a systematic measurement of recently introduced virtualized storage and network I/O within Amazon Web Services (AWS). Our experience shows that there are scaling limitations in clusters relying on these new features. As a result, provisioning for a large-scale cluster differs substantially from small-scale deployments. We describe the implications of this observation for achieving efficiency in large-scale cloud deployments. To confirm the value of our methodology, we deploy cost-efficient, high-performance sorting of 100 TB as a large-scale evaluation.

1. Introduction

Cloud providers such as Amazon Web Services (AWS) [3], Google Cloud Platform [8] and Microsoft Azure [4] offer nearly instantaneous access to configurable compute and storage resources that can grow and shrink in response to application demands, making them ideal for supporting large-scale data processing tasks. Yet supporting the demands of modern Internet sites requires not just raw scalability, but also cost- and resource-efficient operation: it is critical to minimize the resource budget necessary to complete a particular amount of work, or conversely to maximize the amount of work possible given a particular resource budget.

Minimizing cloud costs requires choosing a particular combination of resources tailored to a given application and workload. There have been several measurement studies of the performance of cloud resources [16, 17, 29], and several efforts aimed at automatically selecting a configuration of cloud resources suited to a given workload [13, 14, 31]. This is no easy task, as the diversity within public cloud platforms has rapidly accelerated over the past half decade. For example, as of this writing, Amazon offers 47 different types of VMs, differing in the number of virtual CPU cores, the amount of memory, the type and number of local storage devices, the availability of GPU processors, and the available bandwidth to other VMs in the cluster. The above-mentioned provisioning tools have shown promise, especially for resources such as CPU time and memory space, which can be precisely divided across tenant VMs located on the same hypervisor. On the other hand, shared resources, such as network bandwidth and storage, have proven to be a much bigger challenge [9, 30].

Recently, providers have begun introducing I/O-virtualization at the storage and network layers to enhance performance. Cloud nodes increasingly have access to high-speed flash-based solid state drives (SSDs), which can be virtualized by the hypervisor across multiple guest VMs. These virtualized SSDs can provide high throughput and thousands of IOPS to multiple tenants. Likewise, the data center network fabric is also virtualized, enabling guest VMs to access a "slice" of resources from the network through technologies such as SR-IOV [25]. These virtualized networks enable throughputs and latencies previously unattainable with then-available network technologies. The result is that VMs have access to significantly higher bandwidths than before, e.g., 10 Gb/s VM-to-VM.

These advances in virtualized I/O have the potential to improve efficiency, thereby reducing the number of resources a user needs. Because users pay only for the resources they use, greater efficiency leads to lower costs. However, choosing the right set of resources in this environment is harder than ever, given that the configuration space is

Submission to SoCC '15, August, 2015, Waikoloa, HI, USA.

now even larger than before. Further, as the size of the cluster increases, overall cluster utilization and efficiency can drop, requiring more VMs to meet performance targets and driving up overall cost [6]. Thus an understanding of the scaling behavior of virtualized cloud network and storage resources is key to achieving cost-efficiency in any large deployment.

In this paper, we present a systematic measurement of the scaling properties of recently-introduced virtualized network and storage resources within the AWS public cloud. Our aim is to determine the optimal price points for configuring clusters for data-intensive applications, specifically applications that are I/O-bound. We deploy Themis [19], our in-house implementation of MapReduce, as a case study of I/O-bound data processing applications under a variety of efficiency and data durability assumptions. We give a large-scale evaluation of our methodology using jobs drawn from the annual 100 TB "GraySort" sorting competition [24].

We find that despite newly-introduced I/O virtualization functionality, AWS clusters still have scalability limitations, leading to larger cluster sizes than would be otherwise predicted from the performance of small numbers of nodes. We further find that the choice of cloud resources at scale differs significantly from predicted configurations measured at smaller scale. Thus the actual deployment cost shifts dramatically from estimates based on small-scale tests.

We further show that, by measuring performance at scale, it is possible to provision highly efficient clusters withing the AWS public cloud. As a demonstration of this point, we deploy Themis MapReduce to an AWS cluster consisting of 100s of nodes and 100s of terabytes of virtualized SSD storage, and set three new world records in the GraySort competition at very low cost. We compare our sorting results to other record winners, and find several commonalities between the winning entries, further supporting the results of this work.

The contributions of this paper are:

1. A systematic methodology for measuring the I/O capabilities of high-performance VMs in the public cloud via application-level benchmarks.
2. A measurement of the current AWS offerings at scale, focusing on virtualized I/O.
3. A large-scale evaluation of cost-efficient sorting on 100s of nodes and 100s of terabytes of data informed by this measurement methodology.
4. Three new world records in sorting speed and cost-efficiency based on our evaluation results.

2. Related Work

While many previous works have studied performance in the public cloud, we note that our work is unique in that it has the following three aspects:

- We measure clusters composed of 100s of VMs

Type	vCPU	RAM	Storage	Net.
`m1.small`	1	1.7 GB	160 GB	Low
`m3.xlarge`	4	15 GB	80 GB*	High
`hs1.8xlarge`	16	117 GB	49 TB	10G
`i2.8xlarge`	32	244 GB	6.4 TB*	10G

Table 1: Four example EC2 instance types with various CPU, memory, storage, and network capabilities. Some types use flash devices(*) rather than disk.

- We measure VMs offering high-performance virtualized storage and network devices
- We measure workloads making use of 100s of terabytes of cloud-based storage

We now discuss several related studies.

Measurement: Many have measured the public cloud's potential as a platform for scientific computing. Walker [29] compared Amazon Elastic Compute Cloud (EC2) to a state-of-the-art high-performance computing (HPC) cluster. Mehrotra et al. [17] performed a similar study four years later with NASA HPC workloads. Both came to the same conclusion that the network in the public cloud simply is not fast enough for HPC workloads.

Others have identified this problem of poor I/O performance and have studied the impact of virtualization on I/O resources. Wang and Ng [30] measure a wide variety of networking performance metrics on EC2 and find significantly more variance in EC2 than in a privately owned cluster. Ghoshal et al. [9] study storage I/O and find that EC2 VMs have lower performance and higher variability than a private cloud designed for scientific computation.

Variability in the cloud extends to CPU and memory resources as well. Schad et al. [22] measure the variability of a wide variety of VM resources and find that among other things, heterogeneity in the underlying server hardware dramatically increases performance variance. Two VMs of the same type may run on different processor generations with different performance profiles.

In a somewhat different line of study, Li et al. [16] measure inter-cloud variance, that is, the difference in performance between cloud providers. They compare Amazon EC2, Microsoft Azure, Google AppEngine and RackSpace CloudServers across a variety of dimensions and find that each cloud provider has its own performance profile that is substantially different from the others, further complicating the choice of resource configuration in the public cloud.

Configuration: One goal of measuring the cloud is optimal, automatic cluster configuration. Herodotou et al. [13] describe Elasticizer, a system that profiles Hadoop MapReduce jobs and picks an optimal job configuration on EC2. Wieder et al. [31] construct a similar system, Conductor, that combines multiple cloud services and local servers in a single deployment.

Scheduling around deadlines in shared clusters is another common line of work. ARIA [27] is a scheduler for

Hadoop that meets deadlines by creating an analytical model of MapReduce and solving for the appropriate number of map and reduce slots. Jockey [7] is a similar system for more general data-parallel applications. Bazaar [14] translates these efforts to the cloud by transforming the typical resource-centric cloud API to a job-centric API whereby users request job deadlines rather than collections of VMs. In this model, the cloud provider applies the job profile to an analytical model to compute the cheapest way to meet the job's deadline.

Scale: In the public cloud, users are often presented with a choice of whether to use a larger number of slow, cheap VMs or a smaller number of fast, expensive VMs. The choice to scale out or scale up often depends on the technology available. Michael et al. [18] compared a scale-up SMP server to a scale-out cluster of blades and found the scale-out configuration to be more cost effective. Half a decade later, Appuswamy et al. [2] revisited this question in the context of Hadoop and found the opposite to be true: that a single scale-up server is more cost-effective than a larger scale-out configuration.

While the relative costs of either approach change over time, scale-out configurations must be cautious to avoid excessive variance. Dean and Barroso [6] study tail latency in Web services at Google and demonstrate very long-tailed latency distributions in production data centers. They specifically call developers to build *tail tolerance* into their systems to avoid performance loss. Xu et al. [35] take a pragmatic approach and develop a system to screen for and remove outlier VMs in the long tail.

At the same time, Cockcroft [5] demonstrates how Netflix takes advantage of scale-up VMs on EC2 to reduce costs while substantially simplifying cluster configuration. Cockcroft relies on newer SSD-based VMs, indicating that available hardware drives the choice of whether to scale out or scale up. Of course, the software must also be capable of taking advantage of scale-up. Sevilla et al. [23] describe an optimization to MapReduce that alleviates I/O bottlenecks in scale-up configurations.

Modern, large-scale data-processing systems are now being designed to eliminate as much extraneous I/O as possible. Spark [36] is highly optimized for iterative and interactive workloads that can take advantage of small working sets and large memories. While this target workload is different, the spirit of this work is the same as ours. In fact, Databricks, using Apache Spark, set a world record for sorting using the same AWS VM configuration we derive in this paper [32–34]. In the same sorting contest, Baidu [15] set a record using an implementation of TritonSort [21], further highlighting the need for efficient I/O processing.

3. Background

We now present a brief overview of Amazon Web Services (AWS)'s I/O resources, and then describe our application model.

3.1 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2) is a cloud computing service that provides access to on-demand VMs, termed *instances*, at an hourly cost. There are many types of instances available, each with a particular mixture of virtual CPU cores (vCPU), memory, local storage, and network bandwidth. Table 1 lists a few examples.

VM instances are located in *availability zones*, which are placed across a variety of geographically distributed regions. VMs within the same region are engineered to provide low-latency and high-bandwidth network access to each other. The cost of individual VMs varies by instance type, as well as over time, as new hardware is deployed within AWS. In this paper, we only consider "on-demand" pricing, representing the cost to reserve and keep instances during a given job. Finally, although the cloud offers the abstraction of unlimited computing and storage resources, in reality the number of resources in a given availability zone is limited. This complicates cluster provisioning because the most economical cluster for a given job might not be available when the user needs it. In our experience, launching even 100 VMs of a specific type required two weeks of back and forth communication with engineers within Amazon. Even then, we were only permitted to allocate the virtual machines in a short window of a few hours.

3.2 Virtualized I/O

Recent advances in I/O-virtualization technology have made the cloud an attractive platform for data-intensive computing. Here we discuss three types of virtualized I/O available in the cloud.

3.2.1 Virtualized Storage

In 2012, Amazon introduced the first EC2 VM with solid-state storage devices. Prior to this, all VM types available on EC2 ran either on disk or persistent network-attached storage. Over the next two years, more and more VMs with SSDs became available. By mid 2014, Amazon began highlighting its SSD offerings, relegating the disk-based VMs to the "Previous Generation" of VMs. Other cloud providers have followed suit in the race for newer and faster storage technologies. Google recently added a local SSD offering to its Compute Engine [10] cloud. Microsoft Azure's new G-series VMs include large amounts of local SSD storage [4].

Because offered bandwidth is so high and access times are so low, significant effort is required to support these devices in a virtualized environment at full speed. If the hypervisor spends too much time processing I/O on shared devices, performance will suffer. Recent virtualization tech-

nologies, such as Single Root I/O Virtualization (SR-IOV), enable providers to expose a high-speed I/O device as many smaller, virtualized devices [25]. With SR-IOV, the hypervisor is out of the data path, enabling faster guest VM access to these devices.

3.2.2 Virtualized Network

Today, high-speed networks are common in public cloud platforms. EC2 has offered VMs connected to a 10 Gb/s network as early as 2010, although these VMs were primarily targeted at scientific cluster computing. More recently, 10 Gb/s networks have been rolled out to VM types targeting more general workloads. While achieving maximum network performance is difficult on dedicated hardware, virtualization adds another level of complexity that needs to be addressed for achieving efficiency. As in the case of storage, technologies such as SR-IOV can reduce virtualization overheads and make the most of the high speed network. In a shared environment, SR-IOV can be used to slice the 10 Gb/s interface so each VM receives a portion of the bandwidth. In the case of a single guest VM, eliminating overhead makes 10 Gb/s transfer speeds possible.

Amazon offers SR-IOV through a feature called *enhanced networking*. Though not all VMs support enhanced networking, a large portion of the newer VMs can access the feature. These include not only the VMs that support 10 Gb/s, but also their smaller counterparts, which are likely carved up from larger instance types using SR-IOV to efficiently share a single 10 Gb/s NIC.

Enhanced networking also enables VMs to launch in a *placement group*. Placement groups instruct EC2 to provision VMs strategically in the network to increase bisection bandwidth. Given that oversubscription is common in large data center networks [12], placement groups play an important role in delivering high performance to the user.

3.2.3 Network-Attached Storage

A third type of virtualized I/O, network-attached storage, is a common way to implement persistent storage in cloud environments. The local storage devices described in Section 3.2.1 are typically erased after a VM shuts down or migrates. To store persistent data, users are directed to separate storage services, such as Amazon Simple Storage Service (S3) or Amazon Elastic Block Store (EBS). These services are accessed remotely by a variety of interfaces. For example, S3 supports a RESTful API and can be accessed via HTTP, while EBS is exposed as a standard block device. When evaluating persistent storage in this work, we consider EBS because its interface is similar to a local storage device, thereby supporting unmodified applications. To access EBS, a user simply attaches a volume to a running instance. Volumes can be created with near arbitrary size and IOPS requirements, backed either by disks or SSDs.

Achieving high performance on persistent, network-attached storage brings its own complexities. On the back-

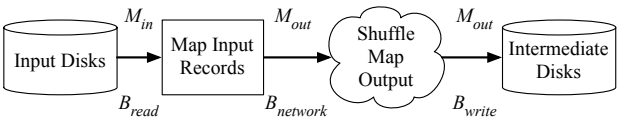


Figure 1: Themis phase 1: `map()` and shuffle.

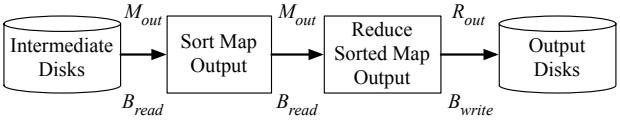


Figure 2: Themis phase 2: sort and `reduce()`.

end, the storage service must be provisioned with enough storage devices to suit users' needs and also have an efficient way of carving them up into volumes. Typically these storage services are also replicated, resulting in additional complexity. On the client's side, an application wants to issue an optimal pattern of I/O's while simultaneously knowing nothing about storage system's internal characteristics or preferred I/O patterns. Finally, congestion in the network or interference from co-located VMs can reduce the performance observed by certain VMs in an unpredictable way.

3.3 Application Models

In this work, we focus on the performance of I/O-bound jobs and deploy Themis, our in-house MapReduce [19, 21]. Themis implements MapReduce as a two pass, pipelined algorithm. In its first *map and shuffle* pass (Figure 1), Themis reads input data from disk into small, in-memory buffers. It then applies the `map()` function to records in these buffers, and the resulting map output, or *intermediate*, data is divided into *partitions*. Unlike traditional MapReduce systems, which write intermediate data to local disk, Themis streams intermediate data buffers over the network to remote nodes before writing to partition files on the remote node's local disks. This implementation eschews traditional task-level fault tolerance in favor of improved I/O performance.

In the second *sort and reduce* pass (Figure 2), Themis reads entire intermediate partitions from local disk into memory. It then sorts these partitions and applies the `reduce()` function. Finally, the resulting records are written to output partition files on local disk. In the rare event that partitions do not fit in memory, a separate mechanism handles these overly large partitions.

We now model the performance of Themis MapReduce under several assumptions about I/O efficiency and data durability.

3.3.1 2-IO

Because Themis eschews traditional task-level fault tolerance, it exhibits the 2-IO property [19], which states that each record is read from and written to storage devices exactly twice. In this paper, we consider data sorting as our mo-

tivating application. For external sorting, Themis achieves the theoretical minimum number of I/O operations [1]. This property not only makes Themis efficient, but it also yields a very simple computational model. When we restrict our focus to I/O-bound applications, the **processing time** of the *map and shuffle* phase can be modeled as:

$$T_1 = \max\left(\frac{M_{in}}{B_{read}}, \frac{M_{out}}{B_{network}}, \frac{M_{out}}{B_{write}}\right) \quad (1)$$

where M_{in} and M_{out} represent the per-node map input and output data sizes, and B_{read}, B_{write}, and $B_{network}$ represent the per-node storage and network bandwidths. For clarity, we have labeled these variables in Figures 1 and 2. In the particular case of sorting, map input and output are the same, and if we ensure that storage read and write bandwidths are the same, we are left with:

$$T_1 = \max\left(\frac{D}{B_{storage}}, \frac{D}{B_{network}}\right) \quad (2)$$

where D is data size to be sorted per node. Next we compute the processing time of *sort and reduce* phase. Because this phase involves only local computation, storage is the only I/O bottleneck:

$$T_2 = \max\left(\frac{M_{out}}{B_{read}}, \frac{R_{out}}{B_{write}}\right) \quad (3)$$

where R_{out} is the reduce output data size. Again in the case of sort, this is equal to D, the per-node data size, so the processing time is:

$$T_2 = \frac{D}{B_{storage}} \quad (4)$$

In practice, it may not be the case that read and write bandwidths are equal, in which case we have:

$$B_{storage} = \min(B_{read}, B_{write}) \quad (5)$$

Therefore the final processing time of the sort is:

$$T = T_1 + T_2 = \max\left(\frac{D}{B_{storage}}, \frac{D}{B_{network}}\right) + \frac{D}{B_{storage}} \quad (6)$$

Finally, we account for the VM's hourly cost C_{hourly} to compute the total dollar cost of the sort:

$$C = C_{hourly}\left[\max\left(\frac{D}{B_{storage}}, \frac{D}{B_{network}}\right) + \frac{D}{B_{storage}}\right] \quad (7)$$

3.3.2 Application-Level Replication

The 2-IO model discussed in Section 3.3.1 represents the upper-bound of cost-efficiency and performance for I/O-bound applications. In practice, storing exactly one copy of the data dramatically reduces durability. We now consider the case where the application makes a remote replica of each output file for improved data durability.

We augment the *sort and reduce* phase with output replication as shown in Figure 3. In addition to writing output partitions to local output disks, the system creates a replica of each output file on a remote node's local output disks.

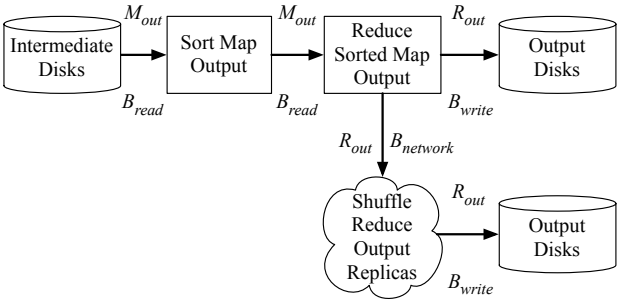


Figure 3: Sort and `reduce()` with Application-Level Replication.

This incurs an extra network transfer and disk write for each output partition file. This online replication affects the total processing time of the *sort and reduce* phase:

$$T_2 = \max\left(\frac{M_{out}}{B_{read}}, \frac{R_{out}}{B_{network}}, \frac{2R_{out}}{B_{write}}\right) \quad (8)$$

In the case of sort, this becomes:

$$T_2 = \max\left(\frac{D}{B_{read}}, \frac{D}{B_{network}}, \frac{2D}{B_{write}}\right) \quad (9)$$

Notice there is now an asymmetry in the storage bandwidth requirements between the *map and shuffle* phase (Equation 2) and the *sort and reduce* phase (Equation 9). This asymmetry will necessitate storage configuration changes, as we will see in Section 5.2.

3.3.3 Infrastructure-Level Replication

Implementing Application-Level Replication as described in Section 3.3.2 adds significant complexity and cost. Cloud providers typically offer infrastructural services to reduce the burden on application developers.

To illustrate the use of Infrastructure-Level Replication, we consider running Themis MapReduce on Amazon EC2 using the EBS storage service described in Section 3.2.3 for input and output data, and local disks for intermediate data only. The time for the *map and shuffle* phase becomes:

$$T_1 = \max\left(\frac{M_{in}}{B_{readEBS}}, \frac{M_{out}}{B_{network}}, \frac{M_{out}}{B_{write}}\right) \quad (10)$$

Similarly, the time for *sort and reduce* is:

$$T_2 = \max\left(\frac{M_{out}}{B_{read}}, \frac{R_{out}}{B_{writeEBS}}\right) \quad (11)$$

We consider the performance and cost implications of these three models in the following sections. Section 4 thoroughly explores the 2-IO model, while Section 5 describes a large-scale evaluation of all three models.

4. Profiling AWS Storage and Networking

We now turn our attention to choosing a cluster configuration on EC2 for I/O-bound applications. As we will show, it is not simply enough to know the VM specifications. The scaling behavior of each VM must be taken into account.

To this end, we design a series of experiments to estimate the performance of I/O-bound jobs on EC2. First, we measure the per-VM bandwidth of local storage devices (Section 4.2). This approximates the performance of instance types where the network is not the bottleneck ($B_{network} = \infty$ in our models). Next, we measure the network performance of each instance type (Section 4.3). Together, these metrics give a performance estimate that accounts for either bottleneck, but assumes that network performance scales perfectly. Then, we measure the actual scaling behavior of the network at the largest cluster sizes that we can reasonably allocate to get a more realistic performance estimate. Finally, we combine the above results with the published hourly costs of each instance type to select the most cost-effective instance type for carrying out a large-scale 100 TB sort job under the 2-IO model described in Section 3.3.1.

The data we use in this analysis comes from a pair of custom-built microbenchmark tools: (1) *DiskBench*, which measures the overall throughput of the storage subsystem within a single node, and (2) *NetBench*, which measures network performance by synthetically generating data without involving local storage. In sections that follow, we describe each of these microbenchmarks in detail and analyze the resulting measurements.

4.1 Measurement Limitations

A common concern when conducting measurements of the public cloud is variance. Resource sharing between customers, either co-located on the same machine or utilizing the same network, increases variance and makes measurement more difficult. Getting a completely fair assessment of the performance of the cloud is complicated by diurnal workload patterns that necessitate measuring at different times of day. Jobs launched during the work week cause different days of the week to experience different performance levels as well. Less-frequent, periodic jobs may even lead to changes based on week of the month or month of the year.

In addition to user-created variance, the infrastructure of the cloud itself is constantly changing, meaning that any attempted measurement is just a snapshot of the cloud in its current state. For example, in the time between the experiments in this paper and the current writing, Amazon has added 10 new instance types to EC2, all of which can alter the performance of the shared network that connects them. Variance can even exist between different data centers belonging to the same provider. Different data centers may contain I/O devices with different performance characteristics, as Schad et al. [22] have shown.

While we acknowledge the amount of variance that exists in the public cloud, we admit that our ability to quantify variance is limited. Despite partial support from Amazon's educational grant program, the experiments described in this paper totaled more than $50,000 in AWS costs, and so we were not able to continue studying AWS in enough detail to account for these forms of variance.

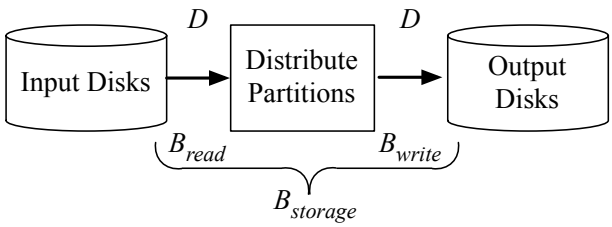


Figure 4: The *DiskBench* storage microbenchmark runs locally on a single node without involving the network.

Furthermore, in many of the more interesting cases, it is often not possible to allocate a large number of on-demand VMs. The large-scale evaluations in Section 5 were only possible after weeks of back-and-forth communication with AWS engineers. When we were finally able to allocate the VMs, we were instructed to decommission them after only a few hours, proving further measurement impossible. For these two reasons, a comprehensive study of variance in the cloud is not presented in this work.

4.2 Local Storage Microbenchmarks

We begin our measurement study by profiling the local storage available on each EC2 VM type with DiskBench. Because local storage devices are often faster than network-attached storage, these measurements are typically an upper-bound on storage performance. We revisit the choice of local versus remote storage in Section 4.4.

4.2.1 The DiskBench Microbenchmark

DiskBench, shown in Figure 4, is a pipelined application that reuses many of the components of Themis MapReduce (Section 3.3). The goal of DiskBench is to isolate the storage subsystem of the *map and shuffle* phase. As such, data records are read from disk, but no `map()` function is applied. Records are randomly assigned to partitions on the same node, and are written back to local disk without involving a network shuffle.

In the measurements that follow, we configure DiskBench to use half of a node's local disks for reading and the other half for writing when more than one device is available. This configuration is typically ideal for local storage devices, and is in fact the configuration used in our earlier experience with high speed sorting [21]. As a result, the bandwidths reported by DiskBench measure a simultaneous read/write workload, and in many cases are approximately half of the bandwidth available in read-only or write-only workloads.

4.2.2 Experimental Design

We begin by running DiskBench on each of the VM types offered by AWS. For each type, we instantiate two to three VMs in the `us-east-1a` availability zone, and we run DiskBench on each of those instances three times. From these six to nine data points, we compute the average per-node storage bandwidth, $B_{storage}$, measured in megabytes per

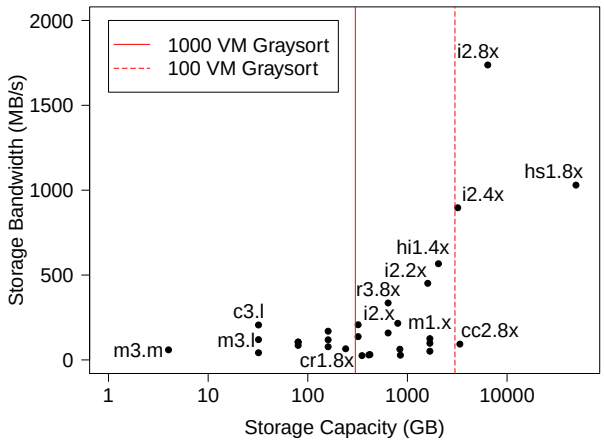


Figure 5: Storage performance of EC2 VM reported by *DiskBench*. Vertical lines cluster VM types into those requiring more than 100 or 1,000 instances to sort 100 TB.

second (MB/s). We run DiskBench on multiple instances to account for natural variance in performance between VMs.

4.2.3 Analysis

The results of DiskBench are shown in Figure 5. We report the mean storage bandwidth across the measured data points, as well as the offered per-VM storage capacity. Recall that storage bandwidth as measured by DiskBench is a read/write workload that approximates half of the read-only or write-only bandwidth of the devices. We have used vertical bars to group VM instance types into regions based on the number of instances needed to sort 100 TB of input data; the rightmost region represents instance types needing fewer than 100 instances. The middle region represents types needing between 100 and 1,000 instances. Finally, the leftmost region represents instance types needing more than 1,000 instances. We highlight these regions because provisioning a large number of instances is not always possible. For example, we found that even with the help of Amazon's engineers, we were only able to allocate at most 186 instances of `i2.8xlarge` in a single availability zone. Furthermore, as we will show, network performance can degrade significantly with larger clusters.

In Figure 5 we have labeled some of the more interesting instance types. Many of these are on the upper right-hand side of the figure and represent a candidate set of instance types which deliver both high storage performance and host enough local storage to meet the capacity requirements of a 100 TB sort with a small cluster. The highest performing instance type in the sub-100 VM region is `i2.8xlarge`, which contains eight 800 GB SSDs and offers 1.7 GB/s of simultaneous read/write bandwidth as measured by DiskBench. The `i2.4xlarge` instance type has half the number of SSDs, with half as much storage bandwidth as a result. Another interesting instance type is `hs1.8xlarge`, which provides the highest density of storage using HDDs instead of SSDs. The `hs1.8xlarge` instance type includes 24 local HDDs and supports 1.06 GB/s of read/write bandwidth. Because of its high storage density, only seven instances are needed to meet the capacity needs of a 100 TB sort operation.

Instance	Min. nodes required for 100TB sort	Cost	
		Sort ($)	Hourly ($/hr)
c3.large	9,375	28	0.105
m3.large	9,375	65	0.14
m3.medium	75,000	66	0.07
m1.xlarge	179	155	0.35
i2.4xlarge	94	211	3.41
i2.8xlarge	47	218	6.82
hs1.8xlarge	7	248	4.60
cr1.8xlarge	1,250	2,966	3.50

Table 2: Estimated dollar cost of sorting 100 TB on a subset of EC2 instance types based solely on local storage performance.

Estimating the dollar cost of sorting: We next use the results of DiskBench in conjunction with the listed AWS hourly cost to predict the total dollar cost of running a 100 TB 2-IO sort using Themis. Here, we consider only local storage performance ($B_{network} = \infty$), and apply the results from Figure 5 to Equation 7 to estimate the total cost of sorting 100 TB.

A subset of these results, shown in Table 2, is presented in ascending order using this sort cost metric to rank instance types. Note that each configuration has its own storage capacity limitations, and to highlight the impact this capacity limitation has on overall resource utilization, we also include the number of nodes necessary to meet the capacity requirements of a 100 TB sort. Specifically, the cluster must be capable of storing 300 TB between the input, intermediate, and output data sets. However, it is important to note that under the assumption of perfect scalability, the total dollar cost is independent of the number of VMs used. To see this, consider that using twice as many VMs cuts job execution time in half, resulting in exactly the same dollar cost.

From Table 2, we see that `c3.large` is the most economical, with a per-sort cost of $28. However, each VM only has 32GB of storage, so 9,375 instances are required to hold the necessary 300 TB of data. Next are the `m3.large` and `m3.medium` instance types, with a sort cost of approximately $65. Again, scaling to meet capacity requirements is a significant challenge. In fact, it is not until the `m1` instance types that clusters of $O(100)$ nodes will suffice. The first instance types with $O(10)$ node cluster sizes are the `i2` types, which are built with arrays of SSDs. A 100 TB sort can be completed with just 47 `i2.8xlarge` instances at a cost of $218. For reference, the most expensive instance type is `cr1.8xlarge`, a memory-optimized 32-core instance type with two 120GB SSDs, on which a 100 TB sort would cost $2,966, a factor of over 100x more expensive than the

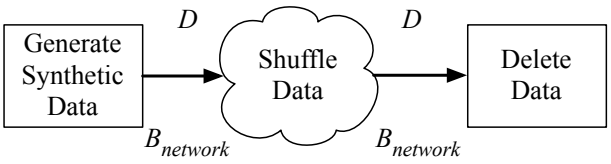


Figure 6: The *NetBench* network microbenchmark measures network scalability and performance using synthetic input data.

cheapest instance type. It is worth noting that two instance types might have hourly costs that are an order of magnitude apart, but the total cost to the user may be very similar, e.g., `m1.xlarge` and `i2.4xlarge`.

Summary: Measuring VM storage bandwidth provides great insight into the total cost of a large-scale data-intensive application. Many high-performance VM configurations can deliver reasonable costs using a small number of nodes.

4.3 Network Microbenchmarks

Next, we measure the performance and scalability of the AWS networking infrastructure. We focus on the subset of instance types that have relatively high performance and high storage capacity as measured in Section 4.2.

4.3.1 The NetBench Microbenchmark

Similar to DiskBench, NetBench (Figure 6) is a pipeline-oriented application derived from Themis. Following the analogy, NetBench aims to isolate the network subsystem from the *map and shuffle* phase. Synthetic data records are generated in-memory and shuffled over the network to remote nodes, which simply delete the data records. NetBench operates entirely in memory and does not touch local disk.

4.3.2 Experimental Design

We perform two experiments to measure the AWS networking infrastructure. The first experiment determines the baseline network bandwidth of each instance type. For each VM type, we allocate a cluster of two nodes in the `us-east-1a` availability zone. On each of these clusters, we run NetBench three times. From these three data points, we compute the average observed network bandwidth, $B_{network}$, which we report in the unconventional unit of megabytes per second (MB/s) for easy comparison with the results of DiskBench. This measurement represents the ideal scaling behavior of the network. When available, we enable the enhanced networking feature and allocate nodes in a single placement group, and we use two parallel TCP connections between nodes to maximize the bandwidth of the high speed VMs.

The second experiment assesses the scaling behavior of the network in a candidate set of VM types. For each type, we allocate increasingly large clusters in the `us-east-1a` availability zone in the following way. We first create a cluster of size two. We then create a cluster of size four by allocating two new VMs and adding them to the existing

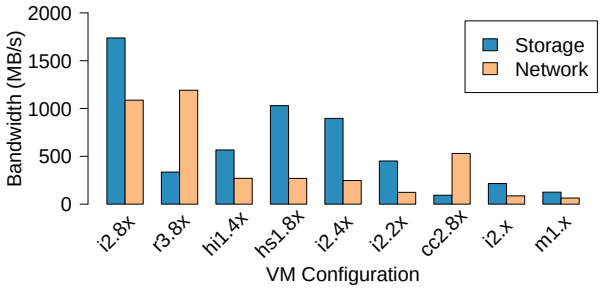


Figure 7: Comparison between storage and network performance of each VM instance type.

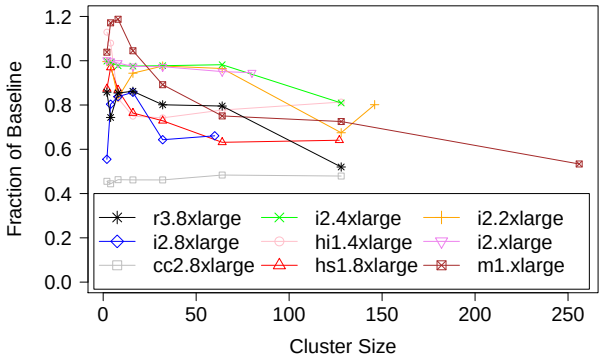


Figure 8: Network performance scalability displayed as a fraction of the baseline network performance given in Figure 7.

cluster. Next we create a cluster of size eight by adding four new VMs. We repeat this process until we reach the end of the experiment. For each cluster size, we run NetBench once, and measure the all-to-all network bandwidth as observed by the slowest node to complete the benchmark.

We note that the largest measured cluster size varies by instance type. In many cases, limits imposed by AWS prevented larger study. For some of the more expensive VMs, we cap the maximum cluster size due to limited funds. We do not use placement groups in this experiment because doing so alters the natural scaling behavior of network and limits cluster sizes. Placement groups also work best when all VMs launch at the same time. This launch pattern is neither representative of elastically scaling applications, nor is it applicable to our experiment setup. Additionally, we use a single TCP connection between VMs because using multiple TCP connections reduces performance at larger cluster sizes, and we are ultimately interested in the performance at scale.

4.3.3 Analysis

The ideal network performance of a select subset of instance types measured in the first experiment is shown in Figure 7. For comparison, we also show the storage performance measured in Section 4.2. For many instance types, the storage and network bandwidths are mismatched. Equation 2 (Section 3.3) suggests that we want equal amounts of storage

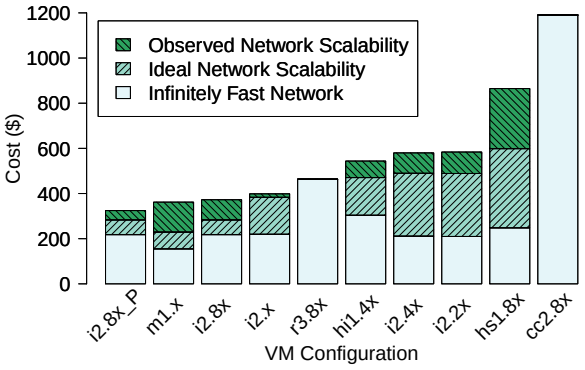


Figure 9: Estimated cost of sorting 100 TB on a subset of EC2 VM types, under various network performance assumptions.

and network bandwidth for the *map and shuffle* phase of sort, but this is often not achieved. For example, the network bandwidth of `i2.8xlarge` is only 63% of its measured storage bandwidth. This mismatch reduces the end-to-end performance of an application that must use both storage and network I/O, resulting in underutilized resources.

Figure 8 shows the network scaling behavior measured in the second experiment. We present the data as a fraction of the baseline bandwidth measured in the first experiment. This comparison is not perfect because the experiments were run on different sets of VMs on different days during a two week period and at different times of day. This perhaps explains how `m1.xlarge` and `hi1.4xlarge` reach speeds that are 20% faster than the baseline at small cluster sizes.

However, the main takeaway is that performance degrades significantly as more nodes are added to the cluster. In eight of the nine VM types measured, performance drops below 80% of baseline during the experiment. One instance type, `cc2.8xlarge`, shows consistently poor performance. We speculate this type resides in a highly congested portion of the network and can only achieve high performance when placement groups are enabled.

The dollar cost of sorting revisited: Finally, we use the results of DiskBench and NetBench to predict the total monetary cost of running a 100 TB 2-IO sort operation on each of the VM instance types. We apply the measured bandwidths to Equation 7 (Section 3.3) to determine the total dollar cost.

This overall cost prediction is shown in Figure 9. For the selected instance types, we show (1) the overall cost assuming that the network is *not* the bottleneck, (2) the cost assuming that the offered network bandwidth scales in an ideal manner, and (3) the cost based on the observed scale-out networking performance. The results show that the lowest-cost instance type for sort is `m1.xlarge`, at $362 per sort followed closely by `i2.8xlarge` and `i2.xlarge`. Interestingly, while the ideal network scalability cost of `i2.8xlarge` is larger than `m1.xlarge`, `i2.8xlarge` has better actual network scaling properties, resulting in very similar overall dollar costs. However, the `i2.8xlarge` instance type supports placement groups, which if employed actually result in a lower overall cost than `m1.xlarge`. We represent this configuration as `i2.8x_P`, with an estimated cost of $325, which is $37 cheaper than `m1.xlarge`.

Summary: Networking performance, particularly at scale, must be accounted for when estimating cost. Poor scaling performance can significantly drive up costs. Better network isolation, e.g. placement groups, can substantially reduce costs. In the case of sort, network isolation results in a savings of $37, or about 10%.

4.4 Persistent Storage Microbenchmarks

We now turn our attention to persistent network-attached storage. While local storage devices typically have higher performance, many cloud deployments will want input and output data sets to persist across VM resets and migrations. We now consider the performance properties of Elastic Block Store (EBS), a persistent network-attached storage service offered by AWS.

4.4.1 Experimental Design

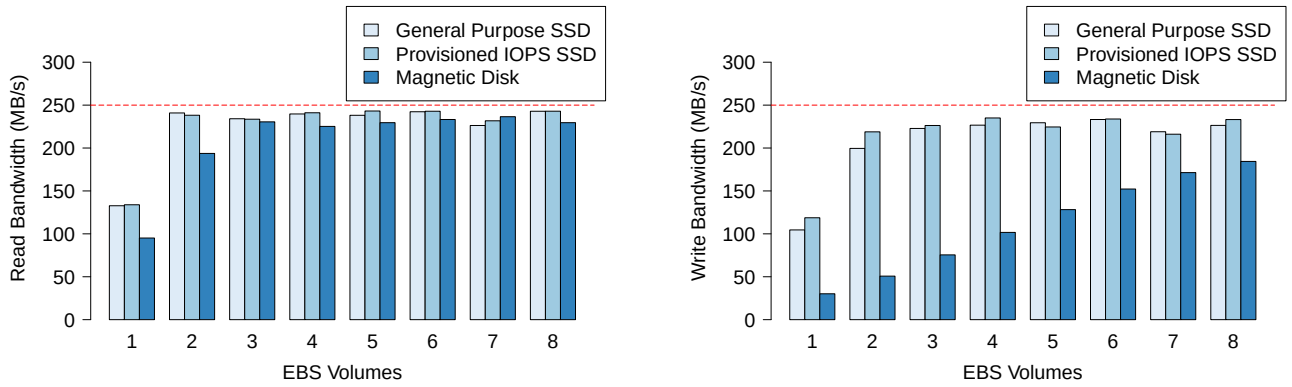
To measure the performance of EBS, we allocate three `i2.4xlarge` instances in `us-east-1a` with the enhanced networking and EBS-optimization features enabled. At the time of the experiment, `i2.4xlarge` was one of the few VM types supporting an EBS throughput of up to 250 MB/s. As of this writing, Amazon offers new classes of instance types, `c4` and `d4`, with speeds of up to 500 MB/s. EBS offers three types of storage volumes: magnetic disk, general purpose SSDs, and IOPS-provisioned SSDs. For each type, we create and attach eight 215 GB EBS volumes to each of the three `i2.4xlarge` instances. We then run DiskBench, and vary the number of EBS volumes used.

We configure DiskBench to run in read-only and write-only modes, but not in the read/write mode described in Section 4.2.1. This more closely resembles an actual EBS-backed application, which will read input data from persistent storage, process it for some period of time using local per-VM storage, and then write output data back to persistent storage. This usage pattern directly corresponds to the Infrastructure-Level Replication model described in Section 3.3.3.

We run each combination of EBS volume type, number of EBS volumes, and DiskBench mode three times on each of the three nodes to get an average bandwidth measurement.

4.4.2 Analysis

Figure 10a shows the read-only DiskBench results, and Figure 10b shows the write-only results. There are four key takeaways. First, a single EBS volume cannot saturate the link between the VM and EBS. Bandwidth increases as more volumes are added up to the 250 MB/s limit. Second, near maximal read performance can be achieved using as few as three volumes of any type. Third, near maximal write perfor-



(a) EBS read performance.

(b) EBS write performance.

Figure 10: EBS performance observed by `i2.4xlarge`. The maximum advertised performance is shown with a dashed line.

mance can be achieved using three or four SSD-based volumes. Finally, the magnetic disk volume type cannot achieve maximal write performance with even eight volumes.

These results are promising in that EBS-optimized instances can actually achieve the maximal read or write bandwidth using the SSD volume types. However, these maximum speeds are quite low relative to the performance of local, per-VM storage. For example, the `i2.4xlarge` instance measured in Section 4.2 is capable of nearly 900 MB/s of read/write bandwidth to its local storage devices, as shown in Figure 5. As such, EBS bandwidth is likely to be a bottleneck in the Infrastructure-Level Replication model (Equations 10 and 11) and will shift the cost analysis quite a bit from that derived in Section 4.3.3.

Summary: Persistent storage systems built from SSDs can deliver reasonable levels of storage performance. However, local, per-VM storage provides far higher levels of performance, so persistent storage will likely be a bottleneck.

5. Evaluation

Thus far we have measured the I/O performance and scalability of several cloud offerings in AWS in the context of the 2-IO model described in Section 3.3.1. We now present a large-scale evaluation of 2-IO, as well as the other models presented in Section 3, Application-Level Replication and Infrastructure-Level Replication. We consider the problem of sorting 100 TB and measure the performance and cost in each case. Each of these evaluations corresponds to one of a larger number of established large-scale sorting benchmarks [24], and thus represents a realistic problem that one might want to solve using the public cloud.

5.1 2-IO

We evaluate the performance and cost of 2-IO by sorting a 100 TB data set that consists of one trillion 100-byte key-value pairs. Each pair consists of a 10-byte key and a 90-byte value. Keys are uniformly distributed across the space of 256^{10} possible keys.

System Name	Cluster Size	Sort Speed (TB/min)	Per-node Speed (MB/s)	Total Cost ($)
Themis	178	6.76	633	299.45
Hadoop	2,100	1.42	11	?
Baidu	982	8.38	142	?

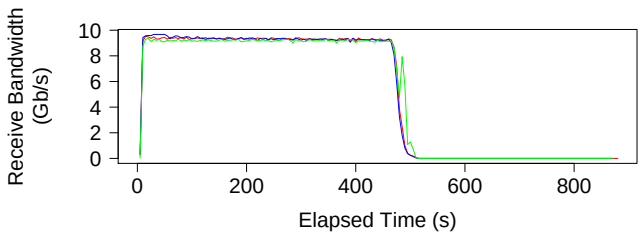
Table 3: Our 100 TB Indy GraySort entry. Past and current record holders are shown for comparison.

Experiment Setup: We allocate 178 on-demand instances of the `i2.8xlarge` VM instance type. All instances belong to a single placement group in the `us-east-1a` availability zone. We use local, per-VM SSDs for input, intermediate, and output data sets.

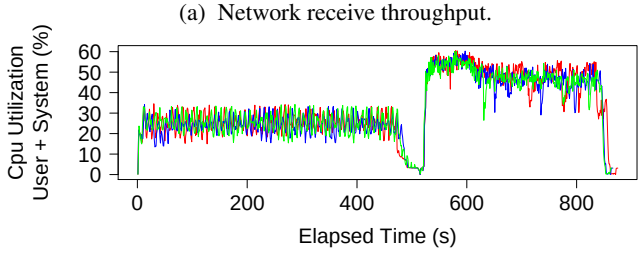
Before running the sort application, we run the DiskBench and NetBench microbenchmarks on the cluster to get a baseline performance measurement, and also to decommission VMs with faulty or slow hardware. DiskBench reports read/write storage bandwidth at 1515 MB/s for the slowest VM, which is 87% of the bandwidth measured in Section 4.2. NetBench yields a network bandwidth of 879 MB/s which is 81% of the ideal bandwidth measured in Section 4.3. We note that this experiment was conducted on a different day than the microbenchmarks described in Section 4, and therefore may have somewhat different performance characteristics.

As in Section 4.2.1 we configure Themis to use four of the eight local SSDs for input and output files, and the remaining four SSDs for intermediate files.

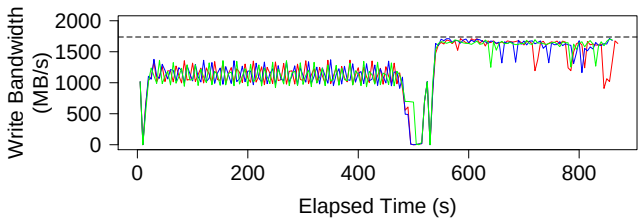
Results: The 100 TB 2-IO sort completes in 888 seconds and requires $299.45. To better understand the bottlenecks and limitations of this particular job, we collect system-level performance metrics using `sar`, `iostat`, and `vnstat` [26, 28]. Using these measurements, we find that during the approximately 500 seconds required to complete the map and shuffle phase, Themis is network-bound. Figure 11a shows the network utilization for three randomly chosen servers as a function of time. The 10 Gb/s network is



(a) Network receive throughput.



(b) CPU utilization.



(c) Disk write bandwidth. The maximum hardware speed is denoted by a dashed line.

Figure 11: System-level metrics collected on 3 of the 178 nodes running the 100 TB 2-IO sort, which shifts from being network-limited to being SSD-limited at $t \approx 500s$.

almost fully utilized, and as a result, the CPU and SSDs are only lightly utilized, as shown in Figures 11b and 11c.

The sort and reduce phase, which begins immediately after the map and shuffle phase completes, is I/O-bound by the local SSDs. Because no network transfer occurs in this phase, Themis can fully utilize the available storage bandwidth, and Figure 11c shows that the disk write bandwidth approaches the limitations of the underlying hardware. Multiple sorting threads allow CPU usage to increase considerably. However the overall system does not become CPU-limited, as illustrated in Figure 11b.

Because the sort job is I/O-limited, the final cost ($299.45) closely resembles the estimated cost given Section 4.3 for `i2.8xlarge` with placement groups ($325). We conclude that the methodology in Section 4 can predict the cost of I/O-bound jobs with reasonable accuracy.

Sort Benchmark: While the analysis thus far has been focused on cost-efficiency, raw performance is also a highly-desired feature. Our 100 TB 2-IO sort conforms to the guidelines of the Indy GraySort 100 TB sort benchmark [24], and achieves an overall throughput of 6.76 TB/min. Our sort is nearly five times faster than the prior year's Indy GraySort record [11] (see Table 3), while still costing less than $300.

System Name	Cluster Size	Sort Speed (TB/min)	Per-node Speed (MB/s)	Total Cost ($)
Themis	186	4.35	390	485.56
Spark	207	4.27	344	551.36
Hadoop	2,100	1.42	11	?

Table 4: Our 100 TB Daytona GraySort record.

We attribute this result to both the methodology in this paper, and also to our Themis MapReduce framework. It is important, however, to note that it is not simply our codebase that yields high performance. In fact, our Indy GraySort speed was surpassed by Baidu [15] by more than 20% using a system derived from TritonSort [20, 21], which also exhibits 2-IO. Thus the 2-IO model of computation has powerful implications for performance as well as cost-efficiency.

5.2 Application-Level Replication

Next we evaluate Application-Level Replication on the same 100 TB data set described in Section 5.1. We run a variant of Themis that supports output replication as illustrated in Figure 3. This particular configuration conforms to the Daytona GraySort benchmark specification [24].

Experiment Setup: This time we allocate 186 on-demand instances of `i2.8xlarge`. As before, we launch all instances in a single placement group. However, due to insufficient capacity in `us-east-1a`, we use the `us-east-1d` availability zone.

As alluded to in Section 3.3.2, the storage requirement asymmetry in Application-Level Replication necessitates a slight change in the configuration of Themis. Here we use five of the eight SSDs for input and output files and the remaining three for intermediate files. This configuration more evenly balances the storage and network requirements of the MapReduce job.

Results: Sorting 100 TB with Application-Level Replication requires 1,378 seconds and results in a total cost of $485.56. While a comparison between this result and the 2-IO result in Section 5.1 is not completely fair due to different sets of resources used in different availability zones on different dates, it is nevertheless interesting to note that the improved data durability increases the cost of the sort from $299.45 measured in Section 5.1 by $186.11.

Sort Benchmark: The performance of our Application-Level Replication surpassed the prior year's record-holder by more than 3x, as seen in Table 4, setting the 100 TB Daytona GraySort record. Apache Spark, run by Databricks, submitted a benchmark result [32–34] that was slightly slower than ours, although our results are close enough to be considered a tie. However, our system is slightly more resource-efficient, resulting in a cost savings of $66, or about 12%.

We note that both results for this sort benchmark use the `i2.8xlarge` VM type on Amazon EC2, despite there being no requirement to use EC2 at all. While we cannot speculate

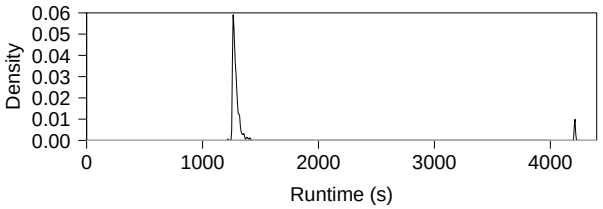


Figure 12: Bimodal elapsed times of reading 100 TB from EBS as seen by a cluster of 326 `c3.4xlarge` VMs.

as to what methodology Apache Spark used to determine the use of `i2.8xlarge`, we can say that the fact that both teams submitted records using this virtual machine validates the conclusions drawn in Section 4.

5.3 Infrastructure-Level Replication

Finally, we evaluate Infrastructure-Level replication on the same 100 TB data set. This time we run the 2-IO implementation of Themis but replace the input and output storage devices with EBS volumes, which provide the desired replication properties. This configuration meets the specifications for the Indy and Daytona CloudSort benchmarks [24]. Incidentally, CloudSort directly measures cost, rather than absolute performance as measured in the GraySort benchmarks, and is more in-line with the spirit of this work.

Preliminary Results: While the analysis in Section 4 suggests `i2.8xlarge` for sorting on local disks, the use of EBS changes the cost analysis substantially. Our measurements indicate the cheapest VM type is `c3.4xlarge`. Therefore we allocate 326 `c3.4xlarge` VMs in a single placement group in the `us-east-1a` availability zone and attach to each four 161 GB general purpose SSD EBS volumes. Unfortunately, this configuration experiences significant variance in read performance. Figure 12 shows a probability distribution function of runtimes across the 1,304 EBS volumes experienced when reading 100 TB from EBS. Approximately 95% of the nodes complete in under 1,400 seconds, but the remaining nodes take three times longer. This long-tailed distribution makes `c3.4xlarge` an ineffective choice for Infrastructure-Level Replication at scale.

Experiment Setup: The next best option after `c3.4xlarge` is `r3.4xlarge`, which is 60% more expensive and offers approximately the same projected performance. We allocate 330 `r3.4xlarge` instances in a single placement group in the `us-east-1c` availability zone. We use a different zone because, as stated earlier in this work, it is often not possible to allocate a large number of instances in a particular zone. To each instance we attach eight 145 GB[1] general purpose EBS volumes. We use EBS for input and output data and local SSD for intermediate data, as suggested in Sections 3.3.3 and 4.4.

[1] Actually 135 GiB. The EBS API uses GiB (2^{30}) rather than GB.

System Name	Cluster Size	Sort Time (s)	Per-node Speed (MB/s)	Total Cost ($)
Themis	330	2981	102	450.84

Table 5: Our 100 TB Indy and Daytona CloudSort record.

Results: We run Infrastructure-Level Replication three times and get completion times of 3094, 2914, and 2934 seconds, yielding an average completion time of 2,981 seconds and an average cost of $450.84 (Table 5). The first point to note is the total runtime, which includes two full rounds of I/O to EBS, is around 3000 seconds. When we compare this to a single round of I/O on `c3.4xlarge`, shown in Figure 12 to be more than 4000 seconds on a cluster of comparable size, we conclude that `r3.4xlarge` does not experience the same long-tailed behavior we see in `c3.4xlarge`. Because EBS is a black-box storage service, we can only guess as to the cause of this behavior. One hypothesis is that the network connecting `c3.4xlarge` to EBS is more congested, and thus more variable, than that of `r3.4xlarge`. It may also be possible that the `us-east-1c` availability zone itself experiences better EBS performance at scale.

Another interesting point is that the per-VM throughput is nearly half of the maximum 250 MB/s throughput to EBS. This indicates that each phase of the sort is running at near-optimal EBS speeds. In fact, Section 4.4 pins the ideal read and write bandwidths at 243 and 226 MB/s, respectively. This suggests an ideal end-to-end throughput of 117 MB/s, so our sort speed is 87% of optimal.

Sort Benchmark: The Infrastructure-Level-Application sort set the world record for both Indy and Daytona CloudSort. Because CloudSort was recently introduced, we do not have prior records to compare against. Further, losing submissions are not published. We can, however, compare to our Daytona GraySort record. We note that although far slower than Daytona GraySort in absolute speed, our CloudSort record actually sorts 100 TB about $35, or about 8%, cheaper with even stronger durability requirements.

6. Conclusions

High-speed flash storage and 10 Gb/s virtualized networks supporting SR-IOV have enabled high performance data-intensive computing on public cloud platforms, and yet achieving efficiency remains challenging for these workloads. We present a systematic methodology for measuring the I/O capabilities of high-performance VMs, and extensively measure these features within EC2 using tools that will be made publicly available at `http://github.com/TritonNetworking/themis` by the time of publication. We find that expected costs rise dramatically due to poor network scaling, altering the optimal choice of VM configurations. By provisioning based on performance measurements at scale, we demonstrate highly efficient sorting on EC2 and set three new world records at very low cost.

References

[1] A. Aggarwal and J. Vitter. The input/output complexity of sorting and related problems. *CACM*, 31(9), Sept. 1988.

[2] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. Scale-up vs scale-out for Hadoop: Time to rethink? In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 20:1–20:13, New York, NY, USA, 2013. ACM.

[3] Amazon Web Services. `http://aws.amazon.com/`.

[4] Microsoft Azure. `http://azure.microsoft.com/`.

[5] A. Cockcroft. Benchmarking high performance I/O with SSD for Cassandra on AWS. `http://techblog.netflix.com/2012/07/benchmarking-high-performance-io-with.html/`.

[6] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, Feb. 2013.

[7] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 99–112, New York, NY, USA, 2012. ACM.

[8] Google Cloud Platform. `http://cloud.google.com/`.

[9] D. Ghoshal, R. S. Canon, and L. Ramakrishnan. I/O performance of virtualized cloud environments. In *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds*, DataCloud-SC '11, pages 71–80, New York, NY, USA, 2011. ACM.

[10] Google Compute Engine. `http://cloud.google.com/compute/`.

[11] T. Graves. GraySort and MinuteSort at Yahoo on Hadoop 0.23. `http://sortbenchmark.org/Yahoo2013Sort.pdf`.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[13] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 18:1–18:14, New York, NY, USA, 2011. ACM.

[14] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Bridging the tenant-provider gap in cloud services. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 10:1–10:14, New York, NY, USA, 2012. ACM.

[15] D. Jiang. Indy Gray Sort and Indy Minute Sort. `http://sortbenchmark.org/BaiduSort2014.pdf`.

[16] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 1–14, New York, NY, USA, 2010. ACM.

[17] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas. Performance evaluation of Amazon EC2 for NASA HPC applications. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 41–50, New York, NY, USA, 2012. ACM.

[18] M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski. Scale-up x scale-out: A case study using Nutch/Lucene. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[19] A. Rasmussen, M. Conley, R. Kapoor, V. T. Lam, G. Porter, and A. Vahdat. Themis: An I/O efficient MapReduce. In *ACM SoCC*, 2012.

[20] A. Rasmussen, M. Conley, G. Porter, and A. Vahdat. TritonSort 2011. `http://sortbenchmark.org/2011_06_tritonsort.pdf`.

[21] A. Rasmussen, G. Porter, M. Conley, H. V. Madhyastha, R. N. Mysore, A. Pucher, and A. Vahdat. TritonSort: A balanced large-scale sorting system. In *NSDI*, 2011.

[22] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endow.*, 3(1-2):460–471, Sept. 2010.

[23] M. Sevilla, I. Nassi, K. Ioannidou, S. Brandt, and C. Maltzahn. SupMR: Circumventing disk and memory bandwidth bottlenecks for scale-up MapReduce. In *Workshop on Large-Scale Parallel Processing (LSPP'14)*, 2014.

[24] Sort Benchmark. `http://sortbenchmark.org/`.

[25] PCI-SIG Single Root IOV. `http://www.pcisig.com/specifications/iov/single_root`.

[26] SYSSTAT. `http://sebastien.godard.pagesperso-orange.fr/`.

[27] A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic resource inference and allocation for MapReduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244, New York, NY, USA, 2011. ACM.

[28] vnStat - a network traffic monitor for Linux and BSD. `http://humdi.net/vnstat/`.

[29] E. Walker. Benchmarking Amazon EC2 for high-performance scientific computing. *LOGIN*, 33(5):18–23, Oct. 2008.

[30] G. Wang and T. E. Ng. The impact of virtualization on network performance of Amazon EC2 data center. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[31] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the deployment of computations in the cloud with Conductor. In *NSDI*, pages 367–381, 2012.

[32] R. Xin. Spark officially sets a new record in large-scale sorting. `http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html`.

[33] R. Xin. Spark the fastest open source engine for sorting a petabyte. `http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html`.

[34] R. Xin, P. Deyhim, A. Ghodsi, X. Meng, and M. Zaharia. GraySort on Apache Spark by Databricks. `http://sortbenchmark.org/ApacheSpark2014.pdf`.

[35] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, pages 329–342, Berkeley, CA, USA, 2013. USENIX Association.

[36] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.