Mimir: Finding Cost-efficient Storage Configurations in the Public Cloud

Hojin Park, Gregory R. Ganger, George Amvrosiadis
Carnegie Mellon University

ABSTRACT

Public cloud providers offer a diverse collection of storage types and configurations with different costs and performance SLAs. As a consequence, it is difficult to select the most cost-efficient allocations for storage backends, while satisfying a given workload's performance requirements, when moving data-heavy applications to the cloud. We present Mimir, a tool for automatically finding a cost-efficient virtual storage cluster configuration for a customer's storage workload and performance requirements. Importantly, Mimir considers all block storage types and configurations, and even heterogeneous mixes of them. In our experiments, compared to state-of-the-art approaches that consider only one storage type, Mimir finds configurations that reduce cost by up to 81% for real-application-based key-value store workloads.

CCS CONCEPTS

• **Information systems** → **Cloud based storage**.

KEYWORDS

public cloud storage, resource provisioning

1 INTRODUCTION

Companies are increasingly moving data-heavy applications to the cloud, often replicating on-premises implementations of integrated data processing and storage backend systems on cloud instances. While researchers have introduced and studied effective approaches for auto-selecting cost-optimized VM instances for computation work [1, 8, 12, 37, 41], less attention has been paid to storage selection. For cold storage, there is usually a clear option (e.g., S3 in AWS or Blob Storage in Azure). For performant storage needs, however, the set of block storage volume types is increasingly diverse in storage characteristics, SLAs and cost structures. Selecting the most cost-effective virtual storage cluster (VSC) configuration for a given data-heavy application deployment is likely beyond all but the most expert user.

Commonly, storage backends (e.g., distributed file systems, key-value stores) are built for use with block storage volumes providing traditional SSD or HDD interfaces. Selecting storage hardware for on-premises deployments is challenging [2, 3, 43], given the wealth of options. The challenge in cloud deployments is similarly difficult, but differently so because of cloud SLA and cost structures. Using AWS as a concrete example, there are three block storage volume types: local-SSD associated with a compute instance, remote-SSD that can be attached to any VM instance, and remote-HDD that can be attached to any compute instance. Making matters worse, each type has multiple options with different costs and different SLA structures regarding cost as a function of performance and capacity required. For example, options include charging per-GB with a fixed budget of IOPS per-GB, providing a specific capacity and performance for a given cost, or charging for a performance budget of MiB/s per-TB. Each customer is best served by a different option, and the most cost-effective may be a mix of options.

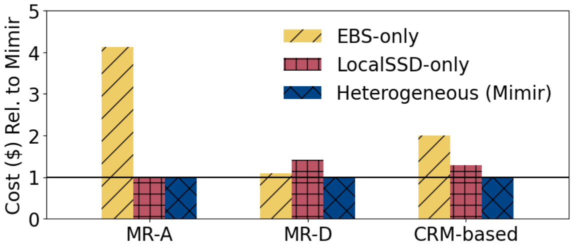


Figure 1: No volume type is most cost-efficient for every workload, and a mix of volume types may be the most cost-effective option. MR-A, MR-D, and CRM-based workloads respectively represent high-throughput, low-throughput, and mixed workloads, and the detailed workload characteristics are described in §5.

Figure 1 illustrates the need to consider many volume types and configurations in selecting a VSC configuration. For each of the three workloads on a distributed storage backend, it shows the cost for the best VSC configuration choice under each of three constraints: considering only local-SSD volume types, only remote storage (EBS) volume types, and arbitrary mixes of both volume types. The most cost-effective configuration is used in each case. We note that: (1) the best single-type choice differs across workloads, and (2) cost is sometimes minimized by mixing volume types.

This paper presents **Mimir**, a tool for finding a cost-effective set of instances, volume types and volume configurations for a distributed storage backend used by a data-heavy application workload. Given high-level workload specifications and performance requirements, as might be produced by profiling an operational version of the system (whether on-premises or using an over-provisioned manual configuration), Mimir considers potentially heterogeneous VSC configurations as shown in Figure 1.

Mimir casts VSC configuration selection as an optimization problem, like most prior tools for automated resource selection. Central to how Mimir achieves its goals is predicting the resources required for the given workload, including both the I/O throughput of the access pattern and the compute and memory needs of the storage

SYSTOR '23, June 5–7, 2023, Haifa, Israel
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9962-3/23/06.
https://doi.org/10.1145/3579370.3594776

software. Using predicted resource requirements and analytically-formulated price-performance cost models of public cloud resources, Mimir determines cost-efficient VSC configurations using dynamic programming. This is in contrast to predicting workload performance for a specific instance type like previous works [1, 24, 30], which allows Mimir to explore heterogeneous VSC configuration options, and find good VSC configurations for workloads composed of multiple access patterns.

Mimir focuses on cost-efficient resource selection for given workload characteristics and requirements, and shows that such resource selection must consider diverse volume types and configurations to minimize costs. In some cases, the workload characteristics and requirements can be determined just once for a stable workload or when provisioning for peak requirements. In other cases, adjusting allocated resources dynamically to match observed variations in the workload can bring further cost benefits. For such cases, Mimir can be used as the resource selection component (replacing less-effective traditional selection components) in a system that monitors the workload variations, intermittently invokes Mimir to suggest new VSC configurations, and enacts configuration changes (and data movement) if the project savings exceeds the cost of changing.

To evaluate Mimir, we used Apache BookKeeper as the distributed storage backend driven by each of two key-value workloads based on discussions with engineers of a top customer relationship management (CRM) service and six workloads based on key-value workloads described by Meta [9]. Our results show significant cost savings arising from Mimir's approach and its ability to consider diverse volume types. For example, compared to a state-of-the-art approach considering only EBS volume type and configurations, Mimir reduces cost by up to 81%. More generally, Mimir consistently and quickly finds cost-effective VSC configurations.

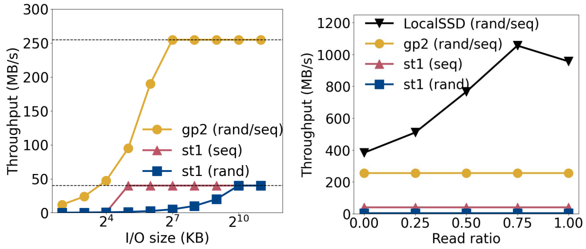
Contributions. We make four primary contributions: (1) We show that finding cost-optimal VSC configurations requires considering diverse volume types and configurations. (2) We describe the architecture and algorithms that allow Mimir to find cost-effective VSC configurations for a distributed storage backend. (3) We demonstrate that Mimir can effectively explore AWS's diverse block storage offerings, reducing cost by up to 81% relative to state-of-the-art approaches. (4) We experimentally demonstrate that Mimir can be used as part of a dynamic reconfiguration system to reduce cost by 74% for diurnal workloads.

2 CLOUD STORAGE CONFIGURATION CHALLENGES

This section motivates the need for tools like Mimir that automate the configuration of virtual storage clusters in public clouds. First, we examine the diversity in performance and cost characteristics of different cloud storage volume types, which complicate manual configurations (§2.1). Second, we examine how the characteristics of cloud storage volume types affect the cost of deploying a scalable storage service in the public cloud (§2.2).

2.1 Public Cloud Storage Characteristics

It is crucial to understand the characteristics of public cloud storage types in order to configure storage systems atop virtual storage cluster in a cost-efficient manner. Mimir formulates the price and performance cost model with the analyzed storage characteristics in this section.



(a) by I/O unit size

(b) by read ratio

Figure 2: Performance characteristics of public cloud storage volume types by (a) I/O unit size and (b) workload read ratio. In (a), both volume types have throughput limits defined by AWS (horizontal lines), and we used a read-only workload.

One of the public cloud storage types we use to build volumes in this paper is *block storage*, such as AWS Elastic Block Store [40], Azure Disk Storage [32], and GCE Persistent Disk [17]. On AWS, there are five different block storage types: local NVMe SSD, remote SSD (gp2, io1), and remote HDD (st1, sc1). Local SSD is served as an SSD locally attached to some instance types, such as i3, c5d, and m5d. It delivers high performance with low latency, but the attached volume capacity is fixed, and it can be an expensive option for data that does not require high throughput. Unlike local SSD, users can attach remote storage volumes (EBS) to the machines they need. The performance of remote storage types is defined as SLAs by the public cloud providers. Though AWS has recently introduced support for EBS Multi-Attach, allowing a single io1 volume to be attached to multiple instances, in this paper, we assume that a single EBS volume can only be attached to one instance since this service is currently available in a limited number of regions and works only for io1 volumes. For instance, AWS currently offers gp2 volumes at 3 IOPS per GiB of provisioned capacity, while it provides 40 MiB/s per TiB of provisioned capacity for st1 volumes.

Figure 2 illustrates the characteristics of 1 TiB of gp2 volume and 1 TiB of st1 volume, in which the performance of each volume is 3000 IOPS and 40 MiB/s, respectively, and local SSD attached at i3.xlarge. We generated the test workloads with the `fio` benchmark [6] varying the access pattern (random/sequential), read ratio, and I/O unit size. For Figure 2(a), we used a read-only workload to evaluate the performance characteristics.

Figure 2(a) shows *how the performance characteristics according to the I/O unit size and access pattern are different for each storage type.* Because gp2 performance is defined in IOPS, as the I/O unit size increases, the throughput of gp2 also increases up to 250 MiB/s, which is the maximum single gp2 volume throughput limited by SLA. In the case of st1, performance is defined in MiB/s, but shows lower throughput for the workloads with random access patterns and I/O units less than 1 MiB [39]. st1 has a throughput limit at 40 MiB/s for 1 TiB st1 volume, in which the limit can be up to 500 MiB/s for the larger st1 volume. The performance of gp2 is the same for both random and sequential data access patterns, while st1 shows better performance for sequential data access than random access.

Figure 2(b) shows *how read ratio affects each volume type's throughput differently*. Throughput of EBS volumes is not affected by the read ratio of the storage workload, as the read ratio is not included in their performance SLAs. The local SSD, however, shows much higher throughput than EBS, and the throughput is affected by the read ratio, while it is not affected by the I/O unit size for requests larger than 32KB.

We have measured the local SSD performance of all the machines we used as candidates of the cost-optimal VSC. The local SSD performance profiling is not a consuming process in terms of time or cost because profiling needs only be performed once. There are other volume types (io1, sc1) we also considered, but we omit them for brevity.

2.2 Apache BookKeeper Use Case

Next, we give a motivating example demonstrating the potential savings of careful machine configuration for an application. Inspired by discussions with engineers from a large customer relationship management (CRM) company shifting from on-premises to cloud, we look at Apache BookKeeper. Apache BookKeeper [21] is a storage system designed for high scalability, fault-tolerance, and low latency. It stores data as streams of log entries in sequences called ledgers, and the ledgers become immutable once the ledger is closed. The primary data access pattern of Apache BookKeeper's storage server is sequential writes and random reads.

We can reduce cost by exploiting heterogeneous resource allocations. Figure 3 shows resource utilization of Apache BookKeeper's storage server running on i3.2xlarge, with a 1.9 TiB local SSD. The workload is write-only and requires 1.8 TiB of data capacity and 360 MiB/s of write throughput at the beginning. After 40 seconds, we increase both requirements by 30%, so the workload's required throughput per TiB remains the same.

For the first 40 seconds, 67% of CPU is idle on average while the storage bandwidth of the local SSD is fully utilized. After 40 seconds, the simplest way to satisfy both increased requirements is to provision another i3.2xlarge which doubles the cost. As Figure 3 shows, however, attaching a 600GiB EBS volume to the original instance instead of provisioning a new instance allows us to store 30% more data while paying only 12% additional cost. It also reduces the cost per data size by 15%, and this heterogeneous allocation allows the workload to utilize 15% more idle computing power.

Therefore, it is crucial to accurately predict how many resources (e.g., CPU, memory, storage bandwidth, etc.) are required for the given workload characteristics to configure cost-efficient heterogeneous virtual storage clusters (§4.3). Also, though we restrict to a single instance type and one workload characteristic in this example, if we consider more instance types and workload characteristics, the gain from the heterogeneity compared to the homogeneous allocation increases (§5.4).

3 PRIOR WORK

In this section, we discuss previous research on the automatic provisioning of public cloud resources and predicting application performance on virtual machines in public clouds.

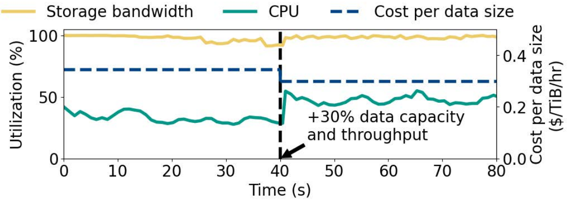


Figure 3: Reducing the cost per data size by exploiting heterogeneous machine allocation. When a storage server uses only local SSD, CPU is underutilized. Attaching an EBS volume can store 30% more data, paying only 12% additional cost.

Configuring storage and VMs in public cloud. Many previous works [8, 27, 36, 45, 48] aim to optimize virtual cluster configuration in public clouds for various workloads. Some studies [1, 24, 25] find near-optimal cloud storage and VM configuration for data analytics workloads, guaranteeing performance and minimizing cost. However, the workloads we target have different nature from the data analytics workloads, e.g., workloads are long-running rather than transient and cannot classify data into input/output and intermediate data, which are common in data analytics applications. Therefore, our research cannot be solved in the same way as previous studies. For example, in the case of data analytics workloads, to reduce the overall cost, the trade-off between using expensive resources for a short duration or simply using cheaper resources should be considered, but our problem does not have this nature.

OptimusCloud [30] jointly optimizes database and VM configurations to find cost-efficient VSC configurations for distributed databases. We consider OptimusCloud as the state-of-the-art to compare with Mimir, but OptimusCloud only considers the EBS volume type, which we show could be a costly configuration compared to a VSC using both local SSD and EBS volume types (§5.4).

Performance prediction on VMs. Numerous previous systems [7, 10, 13, 14, 31, 33–35] studied the method of predicting workload performance on VMs. PARIS [47] uses hybrid offline/online data collection and trains a random forest model to predict the workload performance on VMs. Ernest [44] predicts the performance of large-scale data analytics workloads using statistical modeling. Auto-configuration systems [24, 30] also predict the workloads' performance on VMs using machine learning techniques, such as collaborative filtering and gradient boosting tree.

In contrast, our approach predicts resources required for the given workload performance instead of predicting workload performance on VMs. By predicting resource requirements and knowing the performance SLAs given by the cloud providers, we mathematically formulate a linear programming problem to find the cheapest VSC configuration that has the necessary resources. Still, we can use similar data profiling techniques and prediction approaches that previous works proposed, such as gradient boosting tree.

4 MIMIR DESIGN

Figure 4 shows the workflow of Mimir. First, Mimir takes as input information about multiple workloads' characteristics (§4.1). Storage systems can store data for different workloads, and each workload can have a different data access pattern, such as the data request rate, data access locality, and read/write request ratio.

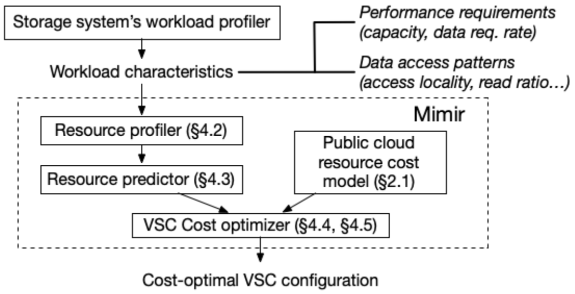

Storage system's workload profiler
Performance requirements
(capacity, data req. rate)
Workload characteristics
Data access patterns
(access locality, read ratio…)
Mimir
Resource profiler (§4.2)
Resource predictor (§4.3)
Public cloud
resource cost
model (§2.1)
VSC Cost optimizer (§4.4, §4.5)
Cost-optimal VSC configuration

Figure 4: Mimir's workflow for optimizing the price of public cloud resources. Mimir profiles the given workloads and learn how many resources (e.g., CPU, memory) are required. The VSC Cost optimizer uses this trained module and cost model of public cloud resources to find the cost-efficient VSC configuration.

Then, our *Resource profiler* profiles each workload and collects data on how many resources are required to run each workload cost-efficiently on the machines in the cloud (§4.2). Using the collected data, the *Resource predictor* learns how to convert each workload specification into the right size of the container to run (§4.3). As demonstrated in §2.2, the heterogeneous single machine configuration is important for the entire VSC's cost-efficiency, and in Mimir, to leverage the heterogeneous single machine configuration, we run multiple storage servers on a single machine by deploying each server in a Docker container, which guarantees the isolation of allocated resources for each storage server. Lastly, the *VSC Cost optimizer* uses the *Resource predictor* and the cost model of public cloud resources to find the cost-efficient VSC configuration of the distributed storage system (§4.4). We assume that the workload characteristics can be profiled (or are known) by Mimir users, and Mimir uses the profiled information as input to its optimization process. One advantage of this design is that any storage system capable of measuring the necessary resource utilization that Mimir uses for optimization can employ Mimir as a resource auto-selector. In our evaluation, we evaluate Mimir only on Apache BookKeeper, but we left extending our evaluation to other storage systems as a future work.

4.1 Input: profiled workload characteristics

Mimir takes workload specifications as input. Table 1 shows the five attributes we use to describe workload characteristics in Mimir. They are divided into two categories: performance requirements and data access patterns.

Data capacity and data request rate are the attributes of the performance requirements that should be satisfied for the given workloads. Performance requirements are also used as profiling knobs and are proportional to the size of *workload fraction*, i.e., a subset of data and data accesses to the subset of data. For example, if a user defines a workload with 1 TiB of data capacity and 10K QPS (queries per second) of data request rate, we expect 3K QPS is required for the 300 GiB of the given workload's data. This can be achieved by load balancing for a clustered storage system, which has been extensively studied [11, 23, 29].

The attributes of the data access pattern describe the behavior of the workloads: temporal/spatial data access locality, read/write ratio, and distribution of object size stored in the storage backend. Unlike performance requirements, Mimir expects the attributes to remain the same for *workload fractions* and uses this assumption in *Resource profiler* to generate a set of *workload fractions* to profile. As future work, Mimir will monitor the actual characteristics of the *workload fraction* on runtime and give feedback to these assumptions.

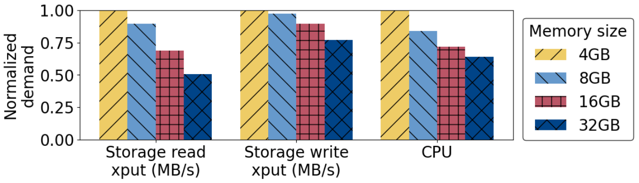


Figure 5: Cost-efficient container sizes for the same workload with different memory sizes. The resource profiler profiles each workload with different memory sizes to collect different resource/performance demands.

Several studies [30, 38] have supported the elastic rightsizing of cloud resources by predicting workload characteristics or in a reactive manner. But elasticity is orthogonal to our work. Instead, we focus on finding the potentially heterogeneous cost-efficient VSC configuration for the mixture of static workloads with different characteristics. Still, we show the extensibility of our tool for dynamic workloads in §5.6.

4.2 Resource profiler

Mimir's goal is to allocate sufficient resources per storage server container (*ContainerSpec*) to satisfy the provided workload performance requirements, while remaining frugal to reduce costs. However, many factors make it challenging to compute the right size of *ContainerSpec* for the arbitrary workload specification analytically. Read/write amplification inherent in the storage servers depends on the implementation and data access pattern; memory size of the storage servers and read/write ratio of workloads affect the necessary storage throughput and computing power to meet the performance requirements. None of these factors can be precisely formulated without the storage system experts and should be reformulated for every storage system to be used. Instead of formulating the cost-efficient size of *ContainerSpec*, Mimir, therefore, profiles and collects data using the *Resource profiler* and predicts the optimal size of containers using the *Resource predictor* trained with the collected data.

***Resource profiler* overview.** The *Resource profiler* runs a storage workload with the given workload characteristics on a benchmark machine to collect the data of the cost-efficient size of *ContainerSpecs*. When profiling, Mimir uses the performance requirement attributes as knobs to get multiple data points. The attributes of *ContainerSpec* we use are: the number of CPUs, memory size, storage bandwidth, storage capacity, and network bandwidth. To get enough profiling data, we chose i3.4xlarge of AWS as the benchmark machine, which has the local SSD with the highest single-storage performance (1900GB NVMe SSD), and sufficient memory and computing resources to profile our evaluation workloads.

There are multiple suitable *ContainerSpecs* for a single workload specification. For example, a read-intensive workload with a high

Type	Attribute	Description	Units
Performance requirement (Profiling knobs)	data capacity	total size of data stored in the storage system	GB
	data request rate	rate of read and write requests arrive at the storage system	QPS
Data access pattern	data request size	mean or distribution of the requested data size	Byte
	read/write ratio	ratio of the read and write request rates	
	access locality	pattern of data access locality	

Table 1: The workload characteristic attributes. The performance requirement attributes are the knobs used by Mimir in profiling to get multiple profile data points, while the data access pattern attributes are not changed.

Algorithm 1 Profiling logic of the *Resource profiler*

```
1:  W: Input workload characteristic
2:  BM: Benchmark machine
3:  procedure PROFILE(W)
4:      p ← MEASUREMAXPERFORMANCE(W, BM)
5:      S ← WORKLOADFRACTIONSETTOPROFILE(W, p)
6:      D ← {}
7:      for wf in S do
8:          u ← MEASURERESOURCEUTILIZATION(wf, BM)
9:          while ¬ ISCONTAINERRIGHTSIZE(wf, u) do
10:             u ← UPDATECONTAINERSPEC(wf, u)
11:         end while
12:         D[wf] ← u
13:     end for
14:     return D
15: end procedure
```

degree of data access locality requires less storage volume performance and computing power with a larger memory size because of memory caching. Figure 5 shows how different the required resources are according to the memory size, even for the same workload specification. Thus, the *Resource profiler* tests different memory sizes to account for multiple *ContainerSpecs* during optimization.

***Resource profiler* logic.** The *Resource profiler* first measures the maximum performance p of the workload on the benchmark machine with the given data access pattern (Algorithm 1, Line 4). Then, it generates a set of N different *workload fractions* (i.e., workloads with $1/N * p, 2/N * p, ..., N/N * p$ performance requirements and given data access pattern) to be profiled (Algorithm 1, Line 5). We used $N = 10$ in our experiments. For each *workload fraction*, *Resource profiler* finds the right container size (Algorithm 1, Line 7-13). It first measures the average resource utilization while running the *workload fraction* on the benchmark machine. However, the container allocated with the average resource utilization may not meet the performance requirements, or it may have been allocated more resources than necessary. So, it iteratively updates the candidate container size by measuring the storage server performance and resource utilization in the container until it finds the cost-efficient container size. Detailed rules for this iterative updates are described below.

Rules for updating *ContainerSpec*. If the current container size satisfies the workload requirements and the average utilization values of some resources are less than the over-provisioning threshold (we used 80%), it is considered those resources are allocated more than necessary. So the profiler reduces their resource allocations. If it does not satisfy the workload requirements, Mimir increases the allocation of the resources with the average utilization higher than the under-provisioning threshold (we used 90%), judging them as bottleneck resources. We used *docker stats*, *iostat*, and *sysfs network interface statistics* to measure CPU, storage/network bandwidth utilizations.

The thresholds we use in this logic can control the cost-efficiency of the container sizes that Mimir selects for the storage servers. For example, if we use a small over-provisioning threshold, Mimir is likely to allocate more resources to the storage servers than they actually require, and vice versa. If we use a smaller under-provisioning threshold, the storage server configuration is more tolerant to a slight increase in workload performance requirements.

4.3 Resource predictor

Based on the data profiled by the *Resource profiler*, Mimir predicts the cost-efficient size of containers for the given workload characteristic. Currently, Mimir provides an implementation using interpolation, but other prediction models, such as a gradient boosting tree [15], could be used instead.

The *Resource profiler* profiled N different *workload fractions*, in which each requires the performance of the $i/N \times p$ (where $i = 1 \cdots N$ and p is the maximum performance on the benchmark machine). Thus, the *ContainerSpecs* for the workload requiring performance less than p can be computed using interpolation. As we noted, we use the large enough instance types as a benchmark machine (i.e., the machine that can profile up to large p) in order to use the interpolation for the *workload fractions* that require high performance.

The interpolation approach allows accurate prediction of the right size of the *ContainerSpecs* as the *Resource profiler* profiles enough data. However, this approach requires a profiling step when the new workload comes in, which requires additional profiling time and cost, although it is cheaper than the cost savings by our tool as we evaluate in §5.7.

4.4 VSC Cost optimizer

Mimir uses dynamic programming (DP) to minimize the cost of the virtual storage cluster while satisfying the performance requirements. Figure 6 shows an example of how we use recursion in the DP problem (OPTCLUSTER) and solve the base cases (OPTSINGLEMACHINE) using a mixed-integer programming. First, the OPTCLUSTER breaks the problem of finding the cost-efficient VSC configuration that can run the entire workload into the smaller problems of finding the ones that can run the *workload fractions*. In order to solve the base cases of the DP problem, the OPTSINGLEMACHINE searches for the cost-efficient resource configuration of a single machine that can execute each *workload fraction*.

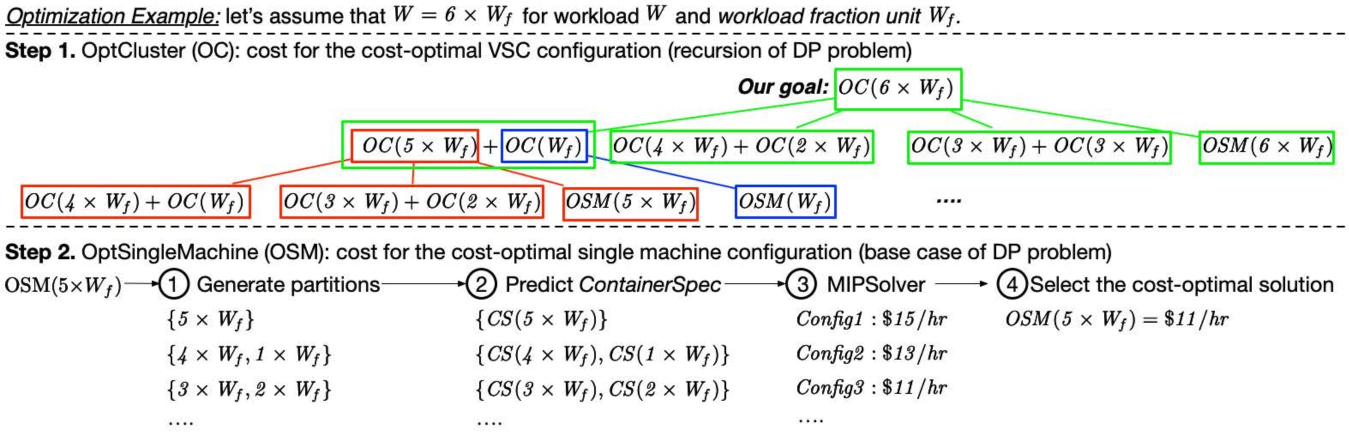

Optimization Example: let's assume that $W = 6 \times W_f$ for workload W and workload fraction unit W_f.
Step 1. OptCluster (OC): cost for the cost-optimal VSC configuration (recursion of DP problem)
Our goal: $OC(6 \times W_f)$
$OC(5 \times W_f) + OC(W_f)$
$OC(4 \times W_f) + OC(2 \times W_f)$
$OC(3 \times W_f) + OC(3 \times W_f)$
$OSM(6 \times W_f)$
$OC(4 \times W_f) + OC(W_f)$
$OC(3 \times W_f) + OC(2 \times W_f)$
$OSM(5 \times W_f)$
$OSM(W_f)$
....
Step 2. OptSingleMachine (OSM): cost for the cost-optimal single machine configuration (base case of DP problem)
$OSM(5 \times W_f)$
① Generate partitions
② Predict ContainerSpec
③ MIPSolver
④ Select the cost-optimal solution
$\{5 \times W_f\}$
$\{4 \times W_f, 1 \times W_f\}$
$\{3 \times W_f, 2 \times W_f\}$
....
$\{CS(5 \times W_f)\}$
$\{CS(4 \times W_f), CS(1 \times W_f)\}$
$\{CS(3 \times W_f), CS(2 \times W_f)\}$
....
Config1 : \$15/hr
Config2 : \$13/hr
Config3 : \$11/hr
....
$OSM(5 \times W_f) = \$11/hr$

Figure 6: Example of the *VSC Cost optimizer*'s optimization algorithm. It uses dynamic programming to break the optimization problem into smaller problems (OptCluster), and mixed-integer programming to solve the base cases (OptSingleMachine).

Algorithm 2 Optimization algorithm of *VSC Cost optimizer*

```
1:  $W$: Profiled workload characteristics
2:  procedure OptCluster($W$)
3:      $S \leftarrow$ WorkloadFractionPairs($W$)
4:      $c \leftarrow \infty$
5:      for Pair<$W_1$, $W_2$> in $S$ do
6:          $t \leftarrow$ OptCluster($W_1$) + OptCluster($W_2$)
7:          $c \leftarrow \min(t, c)$
8:      end for
9:      return min(c, OptSingleMachine($W$))
10: end procedure
11:
12: procedure OptSingleMachine($F$)
13:     $D \leftarrow$ WorkloadFractionPartitions($F$)
14:     $c \leftarrow \infty$
15:     for $d$ in $D$ do
16:         $S \leftarrow$ ResourcePredictor($d$)
17:         $c \leftarrow$ min(MIPSolver($S$), $c$)
18:     end for
19:     return $c$
20: end procedure
```

Algorithm 2 is the pseudocode of the *VSC Cost Optimizer*. We first explain it using a single workload W as input and then expand to using a mixture of workloads as input (§4.5).

Recursion: OptCluster. Mimir first defines the *workload fraction unit* (W_f) of the given workload W, the smallest unit of the workload data stored in the same storage volume. Multiple *workload fraction units* can be stored in the same volume, but a single unit cannot be split. The size of W_f provides the trade-off between the search space size and the optimality of the solution. We empirically evaluated the trade-off and found that using the data size between 50-100 GiB for W_f is generally good in our experiments, e.g., Mimir uses 100 GiB of data size and 1K QPS as W_f for the workload requires 3 TiB of storage capacity and 30K QPS.

The *VSC Cost optimizer* uses DP because the optimization problem has optimal substructure property and overlapping subproblems: *if we found the cost-optimal VSC configuration, then any subcluster of the VSC must have the cost-optimal VSC configuration for the workload fraction running on that subcluster.*

Based on this property, we can argue that the cost-optimal VSC configuration for W is the one that is the cheapest combination of two clusters that one cluster is the cost-optimal for certain amount of *workload fraction* and the other cluster is again the cost-optimal for the remaining *workload fraction* (Algorithm 2, Line 5-9):

$$\textsc{OptCluster}(W) = \textsc{OptCluster}(N \times W_f) = \min(\{\textsc{OptCluster}(i \times W_f) + \textsc{OptCluster}((N-i) \times W_f)\}_{i=1}^{i=\lfloor N/2 \rfloor}, \textsc{OptSingleMachine}(N \times W_f))$$

OptCluster function can be called recursively, until the input of OptCluster becomes W_f, and Mimir uses memoization for the computational efficiency (Figure 6, Step 1). Note that if the cost-optimal VSC configuration has a single machine, we should find the single machine configuration (OptSingleMachine), which is explained below.

Base case: OptSingleMachine. To compute the base cases of the DP problem, OptSingleMachine finds the cost-efficient configuration of a single machine for the given *workload fraction* ($k \times W_f$). ① Within a single machine, there are Partition(k) different ways of distributing the data in storage volumes, where Partition(n) equals the number of possible partitions of n (Figure 6, Step 2-1). For each partition, ② Mimir generates a set of *ContainerSpecs* by predicting them for each *workload fraction* in the partition using the *Resource predictor* (Figure 6, Step 2-2). As each set has the resource requirements of the *workload fractions*, ③ Mimir uses mixed-integer programming (MIPSolver) to minimize the price of a single machine under the resource and the performance requirement constraints (Figure 6, Step 2-3):

$$\begin{aligned}
\underset{Machine,Storage}{\text{minimize}} \quad & \text{Machine[Price]} + \sum_i \text{Storage[i][Price]} \\
\text{subject to} \quad & \sum_{CS} \text{CS[CPU,Mem,...]} \leq \text{Machine[CPU,Mem,...]} \\
& \sum_{CS \in Storage[i]} \text{CS[Storage BW]} \leq \text{Storage[i][BW]}
\end{aligned}$$

Lastly, ④ OptSingleMachine selects the partition with the smallest return value of MIPSolver as the cost-efficient configuration of a single machine (Figure 6, Step 2-4).

4.5 VSC Cost optimizer: multiple workloads

When the input has more than one workload, running the optimization algorithm using all workloads as input at once can find more (or at least the same) cost-efficient results than using separate virtual storage clusters together after finding the cost-efficient VSC configuration for each, because the search space of the former is the superset of one of the latter. For multiple workloads as input, Mimir can use the same optimization algorithm described in §4.4. However, as the number of workloads increases, the complexity of the search space becomes infeasible.

The time complexity (TC) of the recursive loop for the set of workloads $\{W_i\}$, where each workload W_i can be divided into $N_i \times W_{f_i}$, is proportional to the multiplication of N_i.

$$\text{TC of the recursive loop } \propto \prod_i N_i$$

The time complexity of finding the solution for the base cases is proportional to the multiplication of two values: the number of possible partitions of k, which is proportional to the exponential function of the square root of k [4], and the optimization time of the MIPSolver.

$$\text{TC of solving base case } \propto a^{\sqrt{k}} \times T_{\text{MIPSolver}} \text{ (a>1)}$$

Since the total time complexity is the product of these two TCs, it increases exponentially with the number of workloads considered, in which it becomes impractical to use the optimizer even when only three workloads are given as input to the optimizer. To make the time complexity feasible, we use *systematic sampling* and *pairwise workload optimization.*

Systematic sampling: In OptSingleMachine, instead of computing MIPSolver for all the possible partitions, Mimir samples some of the partitions and find the minimum among them. We used systematic sampling rather than random sampling because the order of the partitions we generated has a property that the adjacent partitions tend to have similar configurations. So selecting every *n*th partition allows the Mimir to explore various configurations.

Pairwise workload optimization: As the complexity of the search space increases exponentially with the number of workloads, Mimir runs the optimization algorithm for up to two workloads at once for all pairwise workload combinations. For example, if there are six different workloads as input, rather than giving six of them at once to the optimization function, run pairwise optimization $\binom{6}{2}$ times and find the total cost-efficient VSC configuration using them.

Both approaches provide the trade-off between the optimization execution time and the solution's optimality. We could not directly evaluate the trade-off because the search space is infeasible without these approaches. But, we show that Mimir can find cheaper VSC configuration using these approaches when multiple workloads are considered as an optimization input (§5.4). We also evaluate how fast our approach finds a cost-efficient VSC configuration compared to the naive search algorithm (§5.7).

5 EVALUATION

This section evaluates Mimir using a CRM-based benchmark and Meta's RocksDB key-value workloads [9]. We first describe our experimental setup (§5.1), evaluation benchmarks (§5.2), and baselines to which we compare Mimir (§5.3). We evaluate Mimir to answer the following questions:

- Can Mimir find a cost-efficient VSC to satisfy the requirements of different workloads? (§5.4)
- How effective are key Mimir aspects, including how closely it fits containers to workloads and how important its data partitioning search is? (§5.5)
- Can Mimir be used as part of dynamic resizing? (§5.6)
- How significant are Mimir's overheads? (§5.7)

5.1 Experimental setup

We evaluated Mimir in AWS EC2 US-East-1. We used 55 different instance types for the candidate instance types of a VSC configuration. The candidate instance types include all categories of AWS instance types (except ones with GPUs): "general purpose" instances (m5, m5d) and those "optimized" for compute (c4, c5, c5d), memory (r5, r5d), and storage (i3). For the candidate storage types, we used local SSD (i.e., the SSD in i3, c5d, r5d) and EBS volume types (gp2, io1, st1, sc1). We ran our optimization algorithm on a Xeon E5-2670 2.60GHz CPU with 64 GiB DDR3 RAM, using Gurobi 9.0.1 [18]. We used Apache BookKeeper 4.11.0 as the storage backend where our key-value workloads were run.

5.2 Benchmarks

We evaluated Mimir using two benchmarks (Table 2) on top of Apache BookKeeper: a CRM-based benchmark and a set of workloads similar to the Meta RocksDB key-value workloads described in the paper [9].

Our CRM-based benchmark (**CRM**) is comprised of two workloads: high-throughput workload (`CRM-H`) and low-throughput workload (`CRM-L`). We synthetically generated the CRM-based benchmark based on the discussion with engineers from one of the CRM companies. We used 64 KB of entry size (i.e., data request size) and 2 MB of ledger size, which are the average values the company uses in its BookKeeper storage cluster. Most of their workloads are read-heavy, so the workloads we generated are read-only workloads that read data randomly. For the performance requirements, `CRM-H` and `CRM-L` require 200 MiB/s and 50 MiB/s per TiB of data capacity, respectively, and both have 3 TiB of data. We selected these performance requirements to evaluate how well Mimir finds the cost-efficient VSC configuration for the workloads that have different throughput requirements. The CRM-based benchmark also represents key-value workloads that have large value sizes which are common in real-world [5, 20, 26, 28].

Meta presented detailed characteristics of key-value workloads [9] in their storage cluster, which uses RocksDB as their

Benchmark	Workload	Capacity	Req. rate (QPS)	Req. size	Read req. ratio	Access locality
CRM-based benchmark	H (*High-xput* workload)	3 TiB	9600	64 KB	1.0	Random access
	L (*Low-xput* workload)	3 TiB	2400	64 KB	1.0	
Meta RocksDB benchmark (MR)	A (*Object* in [9])	3 TiB	40K	120 B	0.86	Same as described in [9]
	B (*Object_2ry* in [9])	200 GiB	20K	3 B	0.0	
	C (*Assoc* in [9])	600 GiB	80K	17 B	0.81	
	D (*Assoc_2ry* in [9])	400 GiB	40K	5 B	0.0	
	E (*Assoc_count* in [9])	800 GiB	100K	20 B	0.29	
	F (*Non_SG* in [9])	800 GiB	160K	19 B	0.14	

Table 2: Benchmarks used to evaluate Mimir. CRM-based benchmark consists of a throughput-intensive (CRM-H) and a capacity-intensive workload (CRM-L). Meta RocksDB benchmark consists of 6 real-world workloads [9].

backend storage engine. Among the three production use cases they described, we selected UDB to evaluate Mimir. Because UDB has six workloads that have different characteristics to each other, we can evaluate Mimir for a complicated realistic benchmark. To evaluate UDB-like workload on Apache BookKeeper, we implemented our own benchmark (**MR**) on BookKeeper that has similar characteristics as Meta described. Our benchmark has the same data size distribution, data access locality and count distribution, and average Put/Get request ratio. We used the same distributions presented by Meta, which are General Pareto Distribution [19] for value size distribution and a power model for access count distribution. We implemented only `Put` and `Get` operations, as semantics of other RocksDB operations deviate significantly from available operations in BookKeeper.

5.3 Resource selection baselines

We compare Mimir to three baselines. Whereas Mimir considers all instance and storage types listed in §5.1, when selecting VSC resources, each baseline considers only a subset, and the comparisons show the significance of considering heterogeneous VSC configurations for cost-efficiency.

i3.xlarge-only. The simplest way to configure a VSC on the public cloud is to select one instance type and decide the number of instances to provision from measured storage server performance on the selected instance type. However, since this approach has only a single dimension (i.e., the number of machines) in the search space, it ignores too many potential solutions. In our evaluation, we used `i3.xlarge`, because it is categorized as a storage-optimized instance and provides high-performance local SSD.

Mimir-LocalOnly. Another way to configure the VSC is to use only instance types that have local SSDs, including some compute or memory optimized instance types, such as `m5d`, `c5d` and `r5d`. Local SSD provides high storage performance, but can be costly and may provision more IOPS than necessary. We call this constraint `Mimir-LocalOnly`, because we apply all the Mimir's optimizations to finding the cost-efficient VSC configuration (including mixes of instance types) but limit it to considering only local SSD volumes.

Mimir-EBSonly/OptimusCloud-like. Here, VSCs can only use EBS volumes. EBS volumes can persist data independently from the instance status, and users can provision the volume capacity as much as they need. However, if the workload requires high-performance, it can be more expensive than local SSD. As we explained in Section 3, OptimusCloud [30] restricts the volume type to EBS volumes because of their persistent nature, but our results show that this approach is often much more costly. Like `Mimir-LocalOnly`, this baseline uses all of Mimir's optimizations while only considering EBS volumes. We use the terms `Mimir-EBSonly` and `OptimusCloud-like` interchangeably.

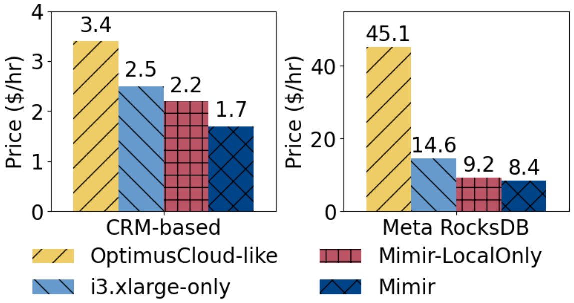


Figure 7: The cost-efficiency analysis of the optimization results of the two benchmarks, CRM and MR. Mimir finds the most cost-efficient VSC configuration compared to the other baselines.

5.4 Cost-efficiency analysis of Mimir

Observation 1: *Mimir finds 2-5.3× cheaper VSC configuration compared to the* `OptimusCloud-like`*, because different workloads prefer different storage types to store data cost-efficiently.*

Figure 7 shows price comparison of the optimization results with Mimir and other baselines for CRM and MR, in which each benchmark is a mix of two and six distinct workloads respectively. Mimir finds cheaper VSC than the other baselines and it is up to 5.3 × cheaper than the `OptimusCloud-like`. In both benchmarks, `Mimir-LocalOnly` finds more cost-efficient VSC configurations than `OptimusCloud-like`. However, it does not mean that every workload data in the benchmarks is more cost-efficient to be stored in local SSD than EBS volume.

Figure 8 shows the storage preference of each workload of CRM and MR. First, CRM-H is the workload that requires high-performance of the storage system. Thus, Mimir finds the VSC configuration that only uses local SSD for the cost-efficient solution. On the other hand, if only EBS volumes are used to store data that requires high storage performance, it should provision much larger storage capacity than it needs to get enough volume IOPS. For example, the cost-efficient VSC configuration of CRM-H searched by `Mimir-EBSonly` uses 15 TiB of gp2 volumes to store only 3 TiB of data to get enough IOPS. It costs 2.5× higher price compared to the result of `Mimir-LocalOnly` or Mimir with no resource constraint.

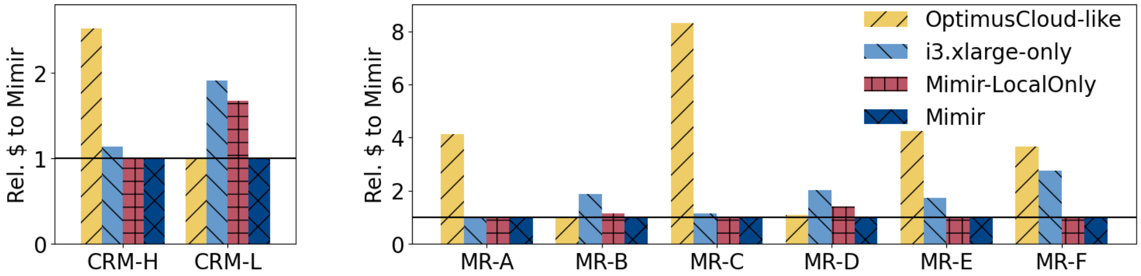


Figure 8: The cost-efficiency analysis of the optimization results of the workloads of the two benchmarks, `CRM` and `MR`. Throughput-intensive workloads (e.g., `CRM-H`, `MR-A,C,E,F`) prefer local SSD as its storage type. In contrast, other workloads (e.g., `CRM-L`, `MR-B,D`) that do not require high throughput prefer EBS volume to local SSD. An `i3` instance type is a costly option for some workloads (e.g., `MR-B,D,E,F`), even if it is categorized as storage optimized instance.

In contrast, `CRM-L` is the workload that does not require high-performance (i.e., capacity-intensive workload). So local SSD is an expensive storage type to store data of `CRM-L`, as it underutilizes storage bandwidth of local SSD. The throughput of `gp2` (i.e., 3 IOPS per provisioned GiB) is enough to support the workload. Figure 8 shows that the cost-efficient VSC configuration optimized by `Mimir-LocalOnly` costs 1.7 × higher price than the one with `Mimir-EBSonly`.

`MR` workloads with different characteristics also show different preferences on the volume type. `MR-A,C,E,F` require 4.13 ×, 8.3 ×, 4.2 ×, 3.6 × higher price with `Mimir-EBSonly` than `Mimir-LocalOnly`, respectively, while `MR-B,D` require 1.1 ×, 1.3 × higher price with `Mimir-LocalOnly`. As Table 2 indicates, `MR-B,D` need lower data request rate and smaller data request size than the other workloads, which makes both workloads well suited to EBS volume type.

Observation 2: *Considering diverse instance types and heterogeneous VSC configurations is crucial for cost-efficiency.*

Not only the volume type, but also the instance type is an important factor that affects the price of the virtual storage cluster. For example, `MR-F` workload requires the second highest storage system throughput per GiB of data among the workloads of `MR`, and `Mimir-LocalOnly` finds more cost-efficient VSC configuration compared to the `Mimir-EBSonly`. However, `i3.xlarge`, a storage optimized instance type, is costly option for the `MR-F` workload (`i3.xlarge-only`). Instead, Mimir and `Mimir-LocalOnly` find the cost-efficient VSC configuration that uses `c5d` instance type, in which `c5d` is a compute-optimized instance type that has a small capacity of SSD. This is because the storage server for `MR-F` needs high computing power (i.e., CPU-intensive) as the workload requires high data request rate. Similarly, `MR-B,D,E` prefer `m5d` or `c5d` to `i3` instance type.

Observation 3: *Considering two workloads together in the optimization algorithm can save cost up to 10.3% compared to using two clusters optimized for each.*

Lastly, we evaluate the pairwise workload optimization of the `MR`'s workloads. We ran the *VSC Cost optimizer* for the $\binom{6}{2}$ number of pairwise combinations of the `MR`'s workloads. Table 3 shows the selected combinations of the workloads that minimize the total price of the virtual storage cluster when the cluster should support all the workloads. (`MR-B`, `MR-E`) pair yields the highest financial gains, 10.3% lower price, when Mimir considers them together to optimize the VSC configuration. `MR-B` and `MR-E` are cost-efficient when data is stored in EBS volume and local SSD, respectively. Thus, Mimir finds the VSC configuration that remote EBS volumes for `MR-B` are attached to the machines for `MR-E` that have local SSDs. In this way, the cost of provisioning instances for `MR-B` could be saved. (`MR-A`, `MR-D`) pair also have the same property (i.e., they prefer different volume types), but there is no financial gain as no computing power left in the machines of `MR-A` to support additional workloads in the same machine. By the pairwise workload optimization, Mimir could save total 4% additional cost compared to using six individual VSCs optimized for each workload.

Optimal Cost/Hour	W_1	W_2	$W_1 + W_2$	Gain
W_1 = MR-A, W_2 = MR-D	$1.86	$0.46	$2.32	0%
W_1 = MR-B, W_2 = MR-E	$0.33	$1.8	$1.91	10.3%
W_1 = MR-C, W_2 = MR-F	$2.25	$2.09	$4.21	3%

Table 3: Pairwise workload optimization of `MR` benchmark. Mimir finds 10.3% cheaper VSC configuration when it optimizes for both workloads at once.

Despite the large search space for the resource heterogeneity and numerous factors to consider (e.g., complex workload and storage characteristic, many price and performance SLAs), Mimir could find the cost-efficient VSC configuration.

5.5 Deep dive into Mimir component effectiveness

In this section, we first evaluate two components of Mimir: *Resource profiler* and *Resource predictor*. And then, we demonstrate how important finding good data partitioning is in finding cost-efficient VSC configurations.

Observation 4: *Mimir selects a cost-efficient container size to run the storage server for the given workload characteristics. Workloads tested utilize at least 83% of allocated resources.*

We evaluate how the *ContainerSpec* profiled by the *Resource profiler* fits for the given workload. Figure 9 shows the example of

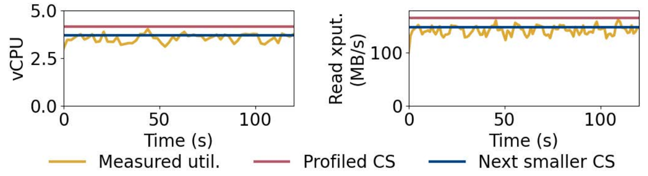


Figure 9: The resource utilization of `MR-A`'s *workload fraction.* The storage server utilizes 85% of the vCPU and storage read throughput (green line) of the allocated resources (red line). If any resource allocation reduces to the next smaller *ContainerSpec* (blue line), the server cannot satisfy the performance requirements.

Workload	Avg % error of the interpolation predictor			
	CPU	Read xput.	Write xput.	Net
MR-A	4.0%	1.1%	7.4%	2.0%
MR-B	2.4%	6.1%	8.4%	1.8%
MR-C	3.1%	0.8%	4.8%	1.1%
MR-D	5.9%	7.7%	1.4%	1.1%
MR-E	2.5%	0.6%	2.7%	1.0%
MR-F	12.9%	2.5%	4.5%	5.1%

Table 4: Percent error of the *ContainerSpec* prediction using interpolation. The interpolation predicts the cost-efficient container size with small percent errors.

how the container with the size of profiled *ContainerSpec* works for the *workload fraction* of `MR-A`. Data access pattern of the *workload fraction* we tested is the same as the one of the original workload, and the performance requirements in this example are 6K QPS of data request rate and 450 GiB of data capacity. The *ContainerSpec* profiled for this *workload fraction* is 4.1 vCPU and 166 MB/s read throughput of the storage volume. To evaluate, we ran a docker container with the profiled amount of resources and measured the CPU and storage throughput of the storage server running on the docker container. As Figure 9 shows, the storage server utilizes 85% of both allocated computing power and storage throughput. We confirmed that the storage server running on the same container (i.e., container with 4.1 vCPU and 166 MB/s read throughput) satisfies the workload requirements, but the storage server on the next smaller container (i.e., container with 3.7 vCPU and 150 MB/s read throughput) following our container size update algorithm (§4.2) cannot meet the performance requirements. We also checked that even a container lack of a single resource failed to satisfy the performance requirements. We ran the same experiment on 300 *ContainerSpecs* we profiled and all the results showed the resource utilization higher than 83% of the resources allocated according to the profiled *ContainerSpecs*.

Observation 5: *Mimir can predict the cost-efficient container size using interpolation with less than 13% error.*

Next, we evaluate our *Resource predictor* using interpolation to see how accurately it predicts the right size of *ContainerSpec*. As a dataset, we use the *ContainerSpecs* that are profiled by the *Resource profiler* for the six workloads of `MR`.

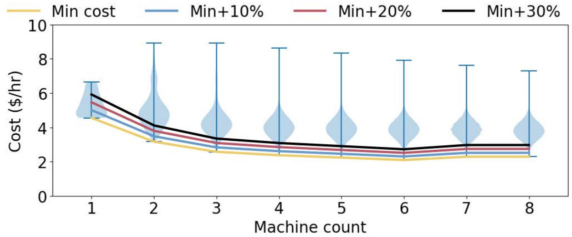


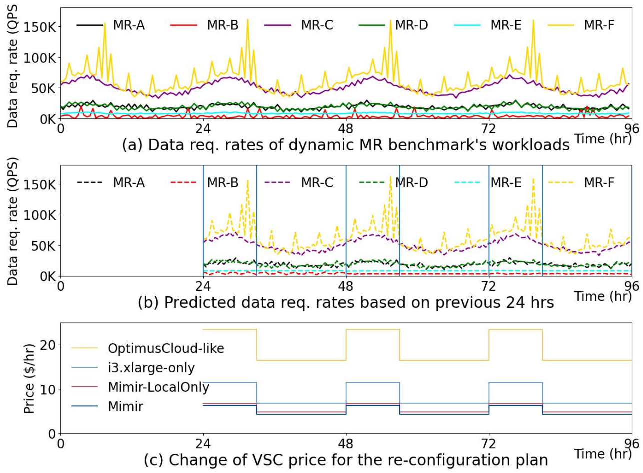
Figure 10: Violin plot of `MR-F` showing the distribution of the cost-efficient VSC configuration price for all possible data partitions. A tiny portion of data partitions can find the near cost-efficient VSC configuration.

As Mimir profiles multiple data points with different performance requirements for each workload, we used the profiled data as a test dataset to evaluate. For instance, the maximum data request rate we measured for the `MR-A` workload on `i3.4xlarge` with memory-to-data ratio of 1:16 is 20K QPS. For the measured maximum data request rate and $N = 10$ (in §4.2) we used, the *Resource profiler* profiled the right size of 10 different *ContainerSpecs* for the *workload fractions* of `MR-A` with the performance requirements of 2K QPS, 4K QPS, ..., 20K QPS. So we evaluate how close the profiled *ContainerSpec* for 4K QPS to the interpolation result of two *ContainerSpecs* for 2K QPS and 6K QPS. Table 4 shows at most 12.9% error for predicting the cost-efficient container size of `MR`'s six workloads.

Observation 6: *The cost-efficient VSC configuration varies greatly depending on how data is distributed. Only 2.4% of data partitions Mimir explored could result in a VSC configuration cheaper than 1.3× of the minimum cost.*

Exploring data partitioning options is an important aspect of Mimir's success. Here data partitioning means how we split the data to be stored in the storage cluster, in which each split is stored in a single storage. For example, consider a workload W that defines its *workload fraction unit* (W_f) as 1/4 of the original size. Then, there are five different partitions of W, which are $\{4W_F\}$, $\{3W_F, W_F\}$, $\{2W_F, 2W_F\}$, $\{2W_F, W_F, W_F\}$, and $\{W_F, W_F, W_F, W_F\}$. Any data partitioning can be used, but we show that only tiny percentage of the possible data partitions can lead to cost-efficient VSC configuration. In this evaluation, we fixed the number of machines to use and ran the optimization algorithm of Mimir for each data partition. So we could compare the minimum price of VSC configuration of each data partition in that the algorithm finds the cheapest VSC configuration of each data partition.

Figure 10 is a violin plot of `MR-F` showing the distribution of the cost-efficient VSC configuration price for each data partition with a fixed number of machines. Mimir finds the most cost-efficient VSC configuration with 6 nodes at the price of 2.09$/hr. For 6 nodes, out of 6043 possible data partitions, only two data partitions could be used for the VSC configuration cheaper than 1.1 × of the minimum price, which is 2.3$/hr, i.e., if we distribute data using one of the remaining 6041 data partitions, we cannot find any VSC configuration that is cheaper than 1.1 × of the minimum price. Even for 1.2 × and 1.3 × of the minimum price, only 24 and 144 data partitions can be used for the VSC configuration cheaper than



(a) Data req. rates of dynamic MR benchmark's workloads

(b) Predicted data req. rates based on previous 24 hrs

(c) Change of VSC price for the re-configuration plan

Figure 11: Using Mimir for dynamic MR benchmark. With dynamic re-configuration, between 8-24hr each day, one can save 32% price compared to using static VSC configuration for the peak performance requirements.

the respective prices. In other words, only 2.4% of all possible data partitions can unearth the VSC configuration cheaper than 1.3 × of the minimum cost.

As we demonstrated, although there are many data partitions and only a few of them can find near cost-efficient VSC configurations, Mimir successfully finds the most cost-efficient one using its optimization algorithm.

5.6 Mimir in dynamic workloads

Observation 7: *If Mimir is used by an auto-scaler to select VSC configurations at changepoints in a dynamic workload, it can reduce daily VSC price by 74% compared to using* OptimusCloud-like *resource selection.*

In this evaluation, we describe how Mimir can be used by an auto-scaling system for dynamic workloads, showing that heterogeneity is still important to find cost-efficient VSC configurations. We used the MR benchmark for the evaluation, but unlike our other experiments, each workload has the diurnal pattern of data request rate described in the corresponding paper [9]. As described earlier, Mimir is a resource selector, not an adaptive auto-scaler, so it would fit as a component of an auto-scaling system that identifies changepoints and dynamically reconfigures as indicated by the selector. Here, we simulate the change of VSC price computed by Mimir’s *VSC cost optimizer* were used in such an auto-scaler.

The solid lines in Figure 11(a) illustrate each workload’s data request rate over time generated by the dynamic MR benchmark. MR-A, MR-C, MR-D, and MR-F have strong diurnal patterns, and MR-B and MR-E are static and bursty, respectively. The dashed lines in Figure 11(b) are the predicted data request rates learned from the previous day’s historical data – the workload behavior of *day*(i + 1) is predicted based on that of *day*(i) – using the SARIMAX [42] forecasting model. We focus on evaluating the cost-efficiency of Mimir’s optimization results for the predicted workload characteristics in dynamic workloads and showing that such optimization must consider diverse storage types and configurations at any phase of dynamic workloads to minimize costs. So evaluating the accuracy of the workload prediction model is out of the scope of this study, and other prediction models [16, 22, 30, 46] can be used instead of SARIMAX.

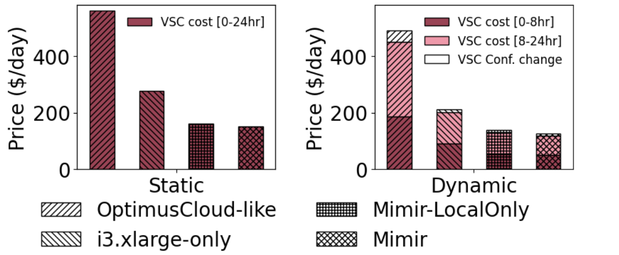


Figure 12: Daily VSC prices for running clusters with static and dynamic configurations. Even if the cost for changing VSC configuration is considered, Mimir is effective as a resource auto-selector for dynamic VSC reconfiguration: Mimir can save 21% daily cost.

Based on the predicted workload behavior, we can plan ahead for the dynamic change of the VSC configuration. We spot 0 and 8 o’clock every day as the change points and used the maximum data request rates between 0-8 hr and 8-24 hr as the performance requirement constraints for each period during the three days we predict the workload behavior.

Figure 11(c) shows the change of VSC price if VSC configurations are adaptively reconfigured using Mimir’s resource selections. Using Mimir as a resource auto-selector, the optimal VSC price reduces by 32% during 8-24 hr compared to the one during 0-8 hr. One reason for the cost reduction is that some workloads, such as MR-C, that prefer local SSDs during 0-8 hr do not require high throughput storage volumes during 8-24 hr and use EBS volumes instead, which are cheaper storage volumes. Another reason is that the computing power required by some workloads, such as MR-F, decreases during 8-24 hr, so fewer machines are used.

We evaluate the cost-benefit analysis by comparing the financial benefit gained by using a cheaper configuration during 8-24 hr with the cost need to be paid for the extra resources during the reconfiguration, i.e., both old and new storage clusters should be running while transferring the data. Figure 12 shows the daily cost comparison between running a cluster with a static configuration that satisfies peak performance requirements for a day and running a cluster that changes its configuration at 0 and 8 o’clock. Using Mimir, the data transfer takes 48 and 38 minutes for the reconfiguration at 0 and 8 o’clock, respectively, which incurs $7.7 additional costs for running the extra cluster while transferring data. Even considering such cost, there is still a 16% gain for changing the VSC configuration based on the Mimir’s optimization results compared to running a static cluster.

Mimir finds more cost-efficient VSC configurations than baselines, saving 8-74% of daily VSC price compared to using the baselines to select auto-scaling system resources.

5.7 Profiling and optimization overheads

Observation 8: Total time and cost overheads incurred by Mimir for MR benchmark are 4.5 hours and $14.1, which are one-time costs for each optimization round, and time overhead can be further shortened if necessary.

We measured how much time and cost overhead Mimir incurs for a single optimization round of the MR benchmark. Profiling step of the *Resource profiler* and optimization step of the *VSC Cost optimizer* mainly induce the overheads.

To profile the six workloads of the MR benchmark, for each workload, we used one i3.4xlarge instance for a storage server and two c5.2xlarge instances for a workload generator. Each workload took 2.8 hours on average, and it costs $14.1 by exploiting the low price of the spot instances. Also, users can further reduce the total profiling time as much as they want for no extra cost just by using more machines because Mimir can profile different *workload fractions* simultaneously as those profiling jobs are independent to each other, i.e., the profiling processes can be parallelized.

To evaluate the time overhead of our optimization algorithm of the *VSC Cost optimizer*, we used three local machines described in §5.1. We ran $\binom{6}{2}$ number of pairwise workload optimizations and it took 1.7 hours in total to find a VSC configuration that costs $8.4/hr. To evaluate how fast our algorithm can find the cost-efficient VSC configuration, we compared it with an algorithm that does not use dynamic programming but use only a mixed-integer programming solver, i.e., instead of breaking the optimization problem into smaller problems as we described in §4.4. We used the same three machines and pairwise workload optimization in this case as well. Without dynamic programming, it failed to find a feasible solution until 30 hours; after 30 hours, the cost-efficient VSC configuration it found costs $10.1/hr; even after 50 hours, the VSC configuration it found costs $9.6/hr, which is still 14% more expensive than Mimir's approach.

6 CONCLUSION

Mimir finds cost-efficient virtual storage cluster (VSC) configurations for distributed storage backends. Given workload information and performance requirements, Mimir predicts resource requirements and explores the complex, heterogeneous set of block storage offerings to identify the lowest-cost VSC configuration that satisfies the customer's need. Experiments show that no single allocation type is best for all workloads and that a mix of allocation types is the best choice for some workloads. Compared to a state-of-the-art approach, Mimir finds VSC configurations that satisfy requirements at up to 81% lower cost for static workloads and 74% lower daily VSC price for dynamic workloads.

ACKNOWLEDGMENT

We thank the anonymous reviewers and Vasily Tarasov, our shepherd, for their help in improving the presentation of this paper. We thank the members and companies of the PDL Consortium: Amazon, Google, Hewlett Packard Enterprise, Hitachi, IBM, Intel, Meta, Microsoft, NetApp, Oracle, Pure Storage, Salesforce, Samsung, Seagate, Two Sigma, and Western Digital for their interest, insights, feedback, and support. This material is based upon work supported by the U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W911NF-20-D-0002. The content of the information does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred. This work was supported by the AWS Cloud Credit for Research program. Hojin Park is supported in part by Korea Foundation for Advanced Studies.

REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 469–482. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[2] Guillermo A. Alvarez, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph A. Becker-Szendy, Richard A. Golding, Arif Merchant, Mirjana Spasojevic, Alistair C. Veitch, and John Wilkes. 2001. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Trans. Comput. Syst.* 19 (2001), 483–518.

[3] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. 2002. Hippodrome: Running Circles Around Storage Administration. In *Conference on File and Storage Technologies (FAST 02)*. USENIX Association, Monterey, CA. https://www.usenix.org/conference/fast-02/hippodrome-running-circles-around-storage-administration

[4] George E. Andrews. 1976. *The Theory of Partitions.* Cambridge University Press.

[5] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-Scale Key-Value Store. In *ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '12, London, United Kingdom, June 11-15, 2012* (London, England, UK) *(SIGMETRICS '12)*. Association for Computing Machinery, New York, NY, USA, 53–64. https://doi.org/10.1145/2254756.2254766

[6] Jens Axboe. 2022. *Flexible I/O Tester.* https://github.com/axboe/fio

[7] Vasanth Balasundaram, Geoffrey Fox, Ken Kennedy, and Ulrich Kremer. 1991. A Static Performance Estimator to Guide Data Partitioning Decisions. In *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Williamsburg, Virginia, USA) *(PPOPP '91)*. Association for Computing Machinery, New York, NY, USA, 213–223. https://doi.org/10.1145/109625.109647

[8] Muhammad Bilal, Marco Canini, and Rodrigo Rodrigues. 2020. Finding the Right Cloud Configuration for Analytics Clusters. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 208–222. https://doi.org/10.1145/3419111.3421305

[9] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H.C. Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. USENIX Association, Santa Clara, CA, 209–223. https://www.usenix.org/conference/fast20/presentation/cao-zhichao

[10] Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy. 2004. Estimating progress of execution for SQL queries *(SIGMOD '04)*. Association for Computing Machinery, New York, NY, USA, 803–814. https://doi.org/10.1145/1007568.1007659

[11] Yue Cheng, Aayush Gupta, and Ali Raza Butt. 2015. An in-memory object caching framework with adaptive load balancing. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*, Laurent Réveillère, Tim Harris, and Maurice Herlihy (Eds.). ACM, 4:1–4:16. https://doi.org/10.1145/2741948.2741967

[12] Andrew Chung, Jun Woo Park, and Gregory R. Ganger. 2018. Stratus: cost-aware container scheduling in the public cloud. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 121–134. https://doi.org/10.1145/3267809.3267819

[13] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. *SIGPLAN Not.* 49, 4 (Feb. 2014), 127–144. https://doi.org/10.1145/2644865.2541941

[14] Salvatore Dipietro, Giuliano Casale, and Giuseppe Serazzi. 2017. A Queueing Network Model for Performance Prediction of Apache Cassandra. In *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools* (Taormina, Italy) *(VALUETOOLS'16)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 186–193. https://doi.org/10.4108/eai.25-10-2016.2266606

[15] Jerome Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (10 2001), 1189–1232. https://doi.org/10.2307/2699986

[16] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. 2007. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *IEEE 10th International Symposium on Workload Characterization, IISWC 2007, Boston, MA, USA, 27-29 September, 2007*. IEEE Computer Society, 171–180. https://doi.org/10.1109/IISWC.2007.4362193

[17] Google. 2022. Google Compute Engine Persistent Disks. https://cloud.google.com/compute/docs/disks

[18] Gurobi Optimization, LLC. 2021. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[19] Jonathan R. M. Hosking and Jamie Wallis. 1987. Parameter and quantile estimation for the generalized pareto distribution. *Technometrics* 29 (1987), 339–349.

[20] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Ling Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: A Raft-based HTAP Database. *Proc. VLDB Endow.* 13 (2020), 3072–3084.

[21] Flavio P. Junqueira, Ivan Kelly, and Benjamin Reed. 2013. Durability with BookKeeper. *SIGOPS Oper. Syst. Rev.* 47, 1 (Jan. 2013), 9–15. https://doi.org/10.1145/2433140.2433144

[22] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. 2012. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium, NOMS 2012, Maui, HI, USA, April 16-20, 2012*, Filip De Turck, Luciano Paschoal Gaspary, and Deep Medhi (Eds.). IEEE, 1287–1294. https://doi.org/10.1109/NOMS.2012.6212065

[23] Markus Klems, Adam Silberstein, Jianjun Chen, Masood Mortazavi, Sahaya Andrews Albert, P. P. S. Narayan, Adwait Tumbde, and Brian F. Cooper. 2012. The Yahoo!: cloud datastore load balancer. In *Proceedings of the Fourth International Workshop on Cloud Data Management, CloudDB 2012, Maui, HI, USA, October 29, 2012*, Xiaofeng Meng, Adam Silberstein, and Fusheng Wang (Eds.). ACM, 33–40. https://doi.org/10.1145/2390021.2390028

[24] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2018. Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 759–773. https://www.usenix.org/conference/atc18/presentation/klimovic-selecta

[25] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 427–444. https://www.usenix.org/conference/osdi18/presentation/klimovic

[26] Chunbo Lai, Song Jiang, Liqiong Yang, Shiding Lin, Guangyu Sun, Zhenyu Hou, Can Cui, and Jason Cong. 2015. Atlas: Baidu's key-value storage system for cloud data. In *IEEE 31st Symposium on Mass Storage Systems and Technologies, MSST 2015, Santa Clara, CA, USA, May 30 - June 5, 2015*. IEEE Computer Society, 1–14. https://doi.org/10.1109/MSST.2015.7208288

[27] Viktor Leis and Maximilian Kuschewski. 2021. Towards Cost-Optimal Query Processing in the Cloud. *Proc. VLDB Endow.* 14 (2021), 1606–1612.

[28] Yongkun Li, Zhen Liu, Patrick P. C. Lee, Jiayu Wu, Yinlong Xu, Yi Wu, Liu Tang, Qi Liu, and Qiu Cui. 2021. Differentiated Key-Value Storage Management for Balanced I/O Performance. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 673–687. https://www.usenix.org/conference/atc21/presentation/li-yongkun

[29] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. DistCache: Provable Load Balancing for Large-Scale Storage Systems with Distributed Caching. In *17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019*, Arif Merchant and Hakim Weatherspoon (Eds.). USENIX Association, 143–157. https://www.usenix.org/conference/fast19/presentation/liu

[30] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, USA, 189–203. https://www.usenix.org/conference/atc20/presentation/mahgoub

[31] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, Jianping Wu and Wendy Hall (Eds.). ACM, 270–288. https://doi.org/10.1145/3341302.3342080

[32] Microsoft. 2022. Azure Disk Storage. https://azure.microsoft.com/en-us/services/storage/disks

[33] Subrata Mitra, Shanka Subhra Mondal, Nikhil Sheoran, Neeraj Dhake, Ravinder Nehra, and Ramanuja Simha. 2019. DeepPlace: Learning to Place Applications in Multi-Tenant Clusters. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys 2019, Hangzhou, China, Augsut 19-20, 2019*. ACM, 61–68. https://doi.org/10.1145/3343737.3343741

[34] Kristi Morton, Magdalena Balazinska, and Dan Grossman. 2010. ParaTimer: A progress indicator for MapReduce DAGs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, Ahmed K. Elmagarmid and Divyakant Agrawal (Eds.). ACM, 507–518. https://doi.org/10.1145/1807167.1807223

[35] Barzan Mozafari, Carlo Curino, and Samuel Madden. 2013. DBSeer: Resource and Performance Prediction for Building a Next Generation Database Cloud. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2013/Papers/CIDR13_Paper52.pdf

[36] Oliver Niehörster, Alexander Krieger, Jens Simon, and André Brinkmann. 2011. Autonomic Resource Management with Support Vector Machines. In *12th IEEE/ACM International Conference on Grid Computing, GRID 2011, Lyon, France, September 21-23, 2011*, Shantenu Jha, Nils gentschen Felde, Rajkumar Buyya, and Gilles Fedak (Eds.). IEEE Computer Society, 157–164. https://doi.org/10.1109/Grid.2011.28

[37] Andrew Or, Haoyu Zhang, and Michael J. Freedman. 2020. Resource Elasticity in Distributed Deep Learning. In *MLSys*. mlsys.org, USA.

[38] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. 2018. Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey. *ACM Comput. Surv.* 51, 4, Article 73 (July 2018), 33 pages. https://doi.org/10.1145/3148149

[39] Amazon Web Services. 2022. Amazon EBS volume types. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#hard-disk-drives

[40] Amazon Web Services. 2022. Amazon Elastic Block Store. https://aws.amazon.com/ebs

[41] Supreeth Shastri and David E. Irwin. 2017. HotSpot: automated server hopping in cloud spot markets. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*. ACM, USA, 493–505. https://doi.org/10.1145/3127479.3132017

[42] Manish Shukla and Sanjay Jharkharia. 2013. Applicability of ARIMA Models in Wholesale Vegetable Market: An Investigation. *Int. J. Inf. Syst. Supply Chain Manag.* 6, 3 (jul 2013), 105–119. https://doi.org/10.4018/ijisscm.2013070105

[43] John D. Strunk, Eno Thereska, Christos Faloutsos, and Gregory R. Ganger. 2008. Using Utility to Provision Storage Systems. In *6th USENIX Conference on File and Storage Technologies, FAST 2008, February 26-29, 2008, San Jose, CA, USA*, Mary Baker and Erik Riedel (Eds.). USENIX, USA, 313–328. http://www.usenix.org/events/fast08/tech/strunk.html

[44] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 363–378. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/venkataraman

[45] Haoyu Wang, Haiying Shen, Qi Liu, Kevin Zheng, and Jie Xu. 2020. A Reinforcement Learning Based System for Minimizing Cloud Storage Service Cost. In *49th International Conference on Parallel Processing - ICPP* (Edmonton, AB, Canada) *(ICPP '20)*. Association for Computing Machinery, New York, NY, USA, Article 30, 10 pages. https://doi.org/10.1145/3404397.3404466

[46] Peng Wang, Haixun Wang, and Wei Wang. 2011. Finding semantics in time series. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.). ACM, 385–396. https://doi.org/10.1145/1989323.1989364

[47] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. 2017. Selecting the Best VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, California) *(SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 452–465. https://doi.org/10.1145/3127479.3131614

[48] Peipei Zhou, Jiayi Sheng, Cody Hao Yu, Peng Wei, Jie Wang, Di Wu, and Jason Cong. 2021. MOCHA: Multinode Cost Optimization in Heterogeneous Clouds with Accelerators. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 273–279. https://doi.org/10.1145/3431920.3439304