Eigen: End-to-end Resource Optimization for Large-Scale Databases on the Cloud

Ji You Li*
Alibaba Group
jiyou.ljy@alibaba-inc.com

Jiachi Zhang*
Alibaba Group
zhangjiachi.zjc@alibaba-inc.com

Wenchao Zhou
Alibaba Group
zwc231487@alibaba-inc.com

Yuhang Liu
Alibaba Group
johan.lyh@alibaba-inc.com

Shuai Zhang
Alibaba Group
xuanluo.zs@alibaba-inc.com

Zhuoming Xue
Alibaba Group
xuezhuoming.xzm@alibaba-inc.com

Ding Xu
Alibaba Group
xuding.xu@alibaba-inc.com

Hua Fan
Alibaba Group
guanming.fh@alibaba-inc.com

Fangyuan Zhou
Alibaba Group
fory@alibaba-inc.com

Feifei Li
Alibaba Group
lifeifei@alibaba-inc.com

ABSTRACT

Increasingly, cloud database vendors host large-scale geographically distributed clusters to provide cloud database services. When managing the clusters, we observe that it is challenging to simultaneously maximizing the resource allocation ratio and resource availability. This problem becomes more severe in modern cloud database clusters, where resource allocations occur more frequently and on a greater scale. To improve the resource allocation ratio without hurting resource availability, we introduce Eigen, a large-scale cloud-native cluster management system for large-scale databases on the cloud. Based on a resource flow model, we propose a hierarchical resource management system and three resource optimization algorithms that enable *end-to-end resource optimization*. Furthermore, we demonstrate the system optimization that promotes user experience by reducing scheduling latencies and improving scheduling throughput. Eigen has been launched in a large-scale public-cloud production environment for 30+ months and served more than 30+ regions (100+ available zones) globally. Based on the evaluation of real-world clusters and simulated experiments, Eigen can improve the allocation ratio by over 27% (from 60% to 87.0%) on average, while the ratio of delayed resource provisions is under 0.1%.

PVLDB Reference Format:
Ji You Li, Jiachi Zhang, Wenchao Zhou, Yuhang Liu, Shuai Zhang, Zhuoming Xue, Ding Xu, Hua Fan, Fangyuan Zhou, and Feifei Li. Eigen: End-to-end Resource Optimization for Large-Scale Databases on the Cloud. PVLDB, 16(12): 3795 - 3807, 2023.
doi:10.14778/3611540.3611565

1 INTRODUCTION

In the past decade, we have witnessed the rapid emergence of cloud-native databases, where users can purchase computing and storage resources, as well as data management services on demand, without having to manage the infrastructures themselves. To host these cloud-native databases, cloud vendors deploy cluster management systems (e.g., Mesos [11], YARN [29], Kubernetes [13], Fuxi [32], Borg [30], Twine [28]) on clusters of machines (nodes) to run jobs or tasks (e.g., database instances). At the center of a cluster management system is a resource scheduler which decides when and how cloud resources are allocated to different jobs. Typically, cloud vendors use two critical metrics to assess whether a resource scheduler is running "well": 1) *resource allocation ratio* refers to the proportion of allocated resources (out of the total cluster resources) that has been allocated by the scheduler to a job; 2) *resource availability* refers to the proportion of resource requests (from a job) that can be fulfilled within a given period of time. Naturally, cloud vendors aim for a high resource allocation ratio (directly leading to lower operation costs) and high resource availability (directly leading to better customer experience).

However, it is intuitively difficult to simultaneously maximize these two metrics. For example, a classic cluster of algorithms (e.g., best-fit, first-fit and their variants) aims to fill each machine in the cluster as tightly as possible, to maximize resource allocation ratio. However, a high resource allocation ratio indicates limited idle resources that are readily available for allocation; therefore, incoming resource requests, especially requests with large resource needs (e.g., 256GB memory), inevitably have an increased probability of failing. Furthermore, heterogeneous resource requirements observed in practical scenarios can cause resource stranding [1] along different resource dimensions (e.g., CPU, memory, disk), and further exacerbate the problem. Another cluster of algorithms chooses to spread tasks across all machines (e.g., worst-fit, E-PVM [1]). Its effectiveness, in terms of resource allocation ratio and resource availability relies on whether the cluster size is set correctly. In fact, there does not exist a "best" resource scheduling algorithm, rather, it depends on the application scenario and requires balancing the optimization of the resource allocation ratio and resource availability.

Resource schedulers face more challenging scenarios in modern database clusters, especially with recent advances in serverless databases [3, 4, 8] which supports auto-scaling based on the real-time workloads. More specifically, resource schedulers face the following two challenges:

*Both authors contributed equally to this research.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611565

[1] Stranded resources refer to the idle resources (e.g., memory) that cannot be effectively utilized due to the exhaustion of other types of resources (e.g., disks).

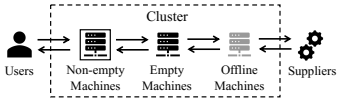

Figure 1: Three-stage resource flow.

1. Frequent and variant resource requests. The database workload varies in response to changes in the user's data size, data distribution, query types, or more generally, application-layer characteristics. Unlike traditional database services, where the users choose the configuration of their database instances, and revisit it on a weekly or even monthly basis, decisions on scaling up/down are made every several seconds or minutes.

2. Low tolerance for delayed resource allocation. Resource requests often reflect the necessary resources to retain SLAs (e.g., query throughput and latency), and therefore, are highly time-sensitive. Resource allocation decisions and the actual allocation need to be made within strong latency (e.g., sub-second) guarantees. The first challenge means that the problem of resource stranding (in the best-fit scheduling algorithm) becomes more prominent, which further triggers a decrease in the allocation availability. On the other hand, an even resource distribution (in the worst-fit scheduling algorithm) will have a penalty on large resource requests: their accommodation requires a large portion of resources in a single machine, which is not readily available and requires time-consuming migration or adding new machines to the cluster. Meanwhile, allocation availability also depends on the cluster size (i.e., the total amount of resources we can provide), and it is impractical to add new machines from the supply chain instantaneously. This drives service providers to deliberately overestimate cluster size, resulting in wasted power consumption of underutilized machines, and an increased carbon emission footprint.

Addressing these challenges, we adopt a cascading resource flow model that divides nodes into three types (shown in Figure 1): non-empty nodes, empty nodes, and offline nodes. To simultaneously maximize the resource allocation ratio and resource availability, we believe resources in the resource flow should be simultaneously optimized (i.e., *end-to-end resource optimization*):

- **Resource optimization of non-empty machines.** We discover two major challenges when adopting classic heuristic bin packing algorithms for resource scheduling of non-empty machines. First, cloud database instances require multi-dimensional and heterogeneous resources, which makes resource scheduling complex. Suboptimal allocations can cause skewness of resource dimensions and consequential stranded resources. Second, migration cost is not negligible when clusters are consolidated. We discuss optimal consolidation solutions under migration cost constraints.
- **Resource optimization of empty machines.** To guarantee high resource availability, especially with the strict latency constraint (see Challenge 2), it is reasonable to maintain a pool of empty machines as a safety net to handle requests that cannot be accommodated in the non-empty machines. However, it is a waste, in terms of power consumption, to maintain a large pool of online machines while a significant portion of it is empty machines. We, therefore, want to maintain *just enough* empty machines online to avoid degradation in resource availability.
- **Resource optimization of offline machines.** Offline machines are a shared resource pool for all database products. From a cloud vendor's perspective, the maintenance and optimization of offline machines are also important (from the perspective of operational cost), but rarely discussed. In practice, cloud vendors purchase machines from suppliers, and the whole process, from placing an order to having the machines delivered, typically takes weeks to months. It is challenging to evaluate the optimal size of offline machines over a long-term period.

Based on our experiences in resource management for Alibaba's database services, we build Eigen, a large-scale, cloud-native cluster management a system that features end-to-end resource optimization. We summarize the contributions of this paper are as follows:

- Based on the resource flow model, we propose a *hierarchical resource management system* which enables three novel resource optimization algorithms: **1.** Vectorized Resource Optimization, a heuristic bin packing algorithm that consolidates non-empty machines in the course of resources allocation; **2.** Exponential Smoothing with Smoothed Adaptive Margins which proactively scales up/down empty machines in a short-term period; **3.** Temporal CNN (TCN) with Minimum-stock Policy which optimizes the number of offline machines in the long-term period.
- We introduce optimizations for fast scheduling, which include master-agent collaborative scheduling and cold instance eviction.
- We evaluate the proposed algorithms, and the overall performance of Eigen on large-scale production clusters with real-world workloads. The evaluation results show that the proposed algorithms significantly increase resource allocation ratios of cloud databases with a negligible rise in the fail/delayed ratio of allocation requests.

2 BACKGROUND AND MOTIVATION

2.1 Resource Scheduling and Optimization

Resource scheduler is a fundamental component in cluster management systems, responsible for allocating resources (such as CPUs, Memory, Disks) to jobs. Notable examples include Kubernetes' Kube-scheduler [14], Twine [28]'s allocator, Borg [30]'s scheduler. Take the Kube-scheduler for instance, it is responsible for assigning a newly created pod to a machine according to some predetermined scheduling policy. Kube-scheduler consists of two steps: *filtering* and *scoring*. The filtering step searches all feasible machines to schedule a pod, and the scoring step chooses the machine deemed most suitable for the pod (based on the scores calculated using a scoring strategy). It allows users to customize scoring strategies for different resource scheduling algorithms.

Resource optimization usually aims to maximize resource allocation ratios (or utilization ratios). Among the previous works, there are roughly two types of resource optimization approaches. The first reduces the resource optimization to the classic bin packing problem. For example, the Kube-scheduler provides two scoring strategies (both are variants of best-fit) to support bin packing of resources [15]. Another approach utilizes statistical and machine learning techniques, notably time series forecasting, to improve utilization ratios by predicting future workloads and reclaiming underutilized resources. For example, Autopilot [24] uses an

exponentially-smoothed sliding window to predict future workloads. In the following sections, we discuss the limitations of previous bin packing and time series forecasting techniques.

2.2 Bin Packing

Resource allocation is typically discussed as a *d-dimensional vector bin packing (d-VBP)* problem, i.e., allocating a set $\mathcal{I}$ of database instances $I_1, I_2, ..., I_n \in [0,1]^d$, where each dimension represents requested resources among a pool of machines. The goal is to minimize the number of machines $M_1, M_2, ..., M_m$, where each machine M_j has a capacity constraint $\| \sum_{I \in M_j} I \|_\infty \leq 1$. d-VBP problem has an online version and an offline version. The online version assumes that resource requests arrive sequentially, and the scheduler decides for each incoming request instantaneously. On the contrary, the offline version decides after all the requests are received.

The d-VPB problem is known to be NP-hard, therefore, it is normally solved by approximation algorithms [7]. Roughly speaking, there are two types of approximation algorithms. The first is based on Integer Linear Programming (ILP). A prominent example, known as *Configuration LP* [23] [5] [12], casts the bin packing problem to a set covering problem, and solves the problem through relaxation and rounding. So far, they have been proven to have the best approximation guarantees. However, based on our and others' experience [26], they do not scale well under large-scale, heterogeneous workloads.

The second uses greedy heuristics, such as first-fit, best-fit, and first-fit decreasing (FFD) and their variants. The greedy heuristic algorithms compute efficiently, therefore, they are more popular in the industrial field. However, their performance has two limitations: Firstly, online heuristic algorithms are highly susceptible to resource skewness. Well-adopted online heuristic algorithms (e.g., Borg [30] and Kubernetes [15]) cause stranded resources under multi-dimensional heterogeneous workloads. Secondly, offline heuristic algorithms hardly consider migration costs. In practice, we want to improve the resource allocation ratio with as little migration cost as possible.

2.3 Time Series Forecasting

Time series forecasting is a technique, which estimates future values for one or more points (termed *forecasting horizon*) based on current and past values of a time series. There are two types of forecasting techniques: statistical models and deep neural network models. Traditional statistical models, such as Exponential Smoothing (ES), Moving Average (MA), Auto-Regressor (AR) and Sparse Periodic Auto-Regressor (SPAR) [27], takes advantage of statistical features/patterns of historical time series data, such as mean, variance, trend, and seasonality, to forecast future values. The statistical models compute efficiently and perform decently when the forecasting horizon is short (i.e., short-term forecasting). Therefore, they have been widely adopted by some well-known systems in the industrial field (e.g., Autopilot [24]., P-Store [27] and MoneyBall [20]). Nevertheless, the statistical models have two limitations: Firstly, the accuracy of long-term forecasting is not satisfying. Secondly, forecasting sharp spikes is challenging.

Deep Neural Networks (DNN) are known for the ability to extract hidden features/patterns of multi-dimensional input data and simulate extremely complex input-output mappings. Recently, there have been works that use DNNs for time series forecasting. Some models, such as DeepAR [25], TCN [6], TFT [18], DSSM [22], have been proven to outperform statistical models when forecasting long-term time series. Some models, such as DILATE [16], aim to forecast sharp rise/drop accurately. Nevertheless, compared to statistical models, DNN-based forecasting is more time-consuming and might lead to delayed resource optimization. Therefore, in this paper, we adopt statistical models for short-term forecasting and DNNs for long-term forecasting.

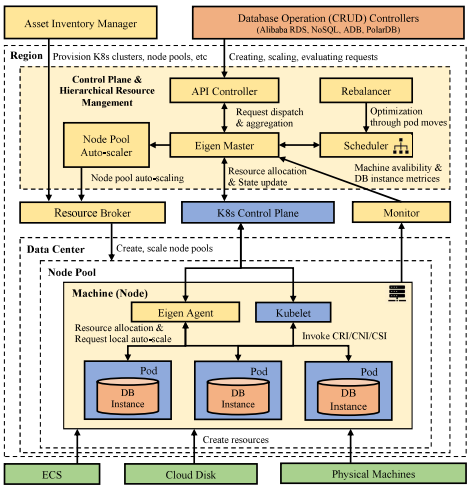

Figure 2: Eigen's architecture.

3 SYSTEM OVERVIEW

3.1 Architecture of Eigen

A large-scale public cloud service provider (e.g., Alibaba Cloud) hosts worldwide database services across geo-distributed *regions*, where each region consists of multiple *data centers*. In a data center, each machine belongs to a *node pool*, a logical cluster designated to host a specific database product (e.g., RDS MySQL, Redis, PolarDB). In each region, we build multiple Kubernetes (K8s) clusters to manage containerized database instances and Eigen oversees the operation of these K8s clusters and interacts with K8s components. Figure 2 depicts the architecture of Eigen.

We highlight the key components of Eigen as follows:

API Controller is a cross-region platform that processes requests from all users. Particularly, resource requests include *creating* (to build new database instances), *scaling* (to scale up/down existing database instances), and *evaluating requests* (to evaluate whether there are enough resources, but never request to allocate). Resource requests originate from an end-user interface, *Database Operation (CRUD) Controllers*. *API Controller* also acts as a resource management gateway: it dispatches requests to the *Eigen Masters* of their corresponding regions, and aggregates the responses.

Eigen Master handles requests from *API controller*, and manages states of nodes and database instances across K8s clusters. Eigen Master has a collection of replicas that use etcd [9] to elect a single leader and to persist the states. The single elected leader is the only

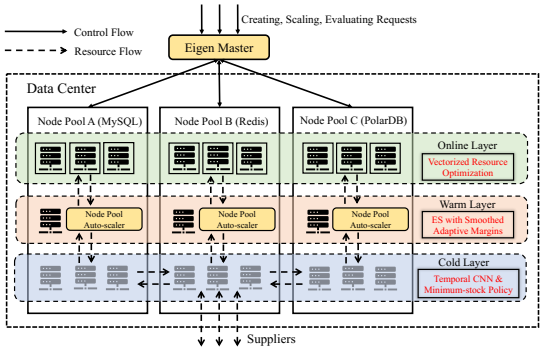

Figure 3: Eigen's hierarchical resource management system.

state mutator, and the non-leader replicas only serve read-only requests. Each machine deploys a *Eigen Agent*, which communicates with *Eigen Master*, handles local auto-scaling requests and manages states of local resources & database instances. The *Eigen Agent* is important for optimizations such as master-agent collaborative scheduling and cold instance eviction (details in Section 5).

Scheduler is the key component for resource scheduling and optimization. It works closely with the *Rebalancer* to improve the resource allocation ratio. Specifically, the Scheduler decides the online placement of database instances; and the *Rebalancer* periodically consolidates clusters through an offline bin packing algorithm which suggests pod migrations (details in Section 3.2). To improve scheduling throughput, the *Scheduler* runs many instances in parallel and uses optimistic concurrent control to resolve conflicts. Eigen adopts techniques, such as scheduling isolation between node pools (similar to Twine's sharding [28]) and an in-memory snapshot of etcd, to further accelerate the scheduling process. They have to be scalable, i.e., *Eigen Master* and *Scheduler* need to support scheduling on up to 100K nodes per region in a typical public cloud setting.

Resource Broker creates, and manages all resources from resource suppliers (e.g., *ECS/EC2, Cloud Disk, and Physical Machines*). Through *Resource Broker*, the *Asset Inventory Manager* provisions and manages the lifecycles of different types of resources for subsystems, such as K8s clusters, node pools, etc. *Node Pool Auto-scaler* supports node pool auto-scaling which elastically tunes the online and offline machines of each node pool (details in Section 3.2), based on the machine statistics collected from the *Monitor*.

3.2 Hierarchical Resource Management

Based on the resource flow in Figure 1, we propose a hierarchical resource management system for end-to-end resource optimization. Take a data center demonstrated in Figure 3 for instance, we divide machines of each node pool into three layers: online layer (green box), warm layer (orange box), and cold layer (blue box). We describe different optimization problems in each layer as follows:

Online layer consists of non-empty machines (online machines). In this layer, the optimization problem is to allocate database instances with heterogeneous resource requests on as few as possible online machines. We design Vectorized Resource Optimization (VRO), which consists of an online version and an offline version. The online version of VRO is implemented in *Scheduler* to schedule resource allocations for online requests. The offline version of VRO is implemented in *Rebalancer*, which periodically rebalances the cluster through pod moves (i.e., migrations of database instances). In addition to consolidating clusters, the offline version of VRO focuses on reducing the number of migrations for lower rebalance costs. We discuss details in Section 4.1.

Warm layer consists of empty machines (warm machines) which work as "buffers" to support high resource availability. The optimization problem in this layer is to evaluate the minimum of resources which will not cause delayed requests in short-term time periods (e.g., ten seconds, one minute, ten minutes). We design Exponential Smoothing with Smoothed Adaptive Margins, a short-term time series forecasting algorithm for resource usage. It is implemented in *Node Pool Auto-scaler*, and is able to automatically scale up/down warm machines. We discuss the details of this algorithm in Section 4.2.

Cold layer consists of offline machines (cold machines). In this layer, the optimization problem is to evaluate the minimum of cold machines which will not cause failed requests in a long-term time period (e.g., one week, three weeks, one month). We train and deploy probabilistic time-series forecasting models on *Node Pool Auto-scaler* to predict long-term daily resource consumption (i.e., a daily difference of allocated resources). Based on the predictions, we design a Minimum-stock Policy that suggests adding or removing cold machines. We discuss details in Section 4.3.

In Figure 3, resource flows depict how machines move between layers and node pools. *Schedulers* uses VRO to allocate resources of online machines to users (User ← Online). Users may migrate their data and release allocated resources (User → Online). Warm machines will be promoted to online machines once *Scheduler* fails to allocate resources due to a lack of resources for online machines (Online ← Warm). Online machines can be emptied by *Rebalancer* (Online → Warm). Based on short-term resource usage prediction, *Node Pool Auto-scaler* proactively tunes warm buffers by configuring/deconfiguring cold machines (Warm ← Cold & Warm → Cold). According to long-term resource consumption prediction and stock management suggestions, the cold machines are either purchased from suppliers (Cold ← Supplier) or exchanged to other node pools (Cold ↔ Cold). All resource flows work simultaneously.

4 RESOURCE OPTIMIZATION ALGORITHMS

In this section, we present the intuitions and technical details of three resource optimization algorithms designed for Eigen's hierarchical resource management system: Section 4.1 describes Vectorized Resource Optimization, online and offline scheduling algorithms in the online layer. Section 4.2 describes Exponential Smoothing with Smoothed Adaptive Margins, a short-term time series forecasting algorithm in the warm layer. Section 4.3 describes TCN with Minimum-stock Policy, long-term time series forecasting and stock management algorithm in the cold layer.

4.1 Vectorized Resource Optimization

4.1.1 Online Allocation. In order to mitigate resource skewness of traditional online heuristic *d*-VBP algorithms (e.g., first-fit, best-fit), we propose VRO online allocation, a *2-step* best-fit variant which combines a *loss-first* heuristic and a *skewness-first* heuristic:

Step 1: The first step uses a loss-first heuristic based on a novel loss function (i.e., scoring strategy) which evaluates machines:

$$l(r) = \alpha \, \|r\| + \beta \sum_{1 \le i \le d} r(i) \bmod f_i \tag{1}$$

where $r \in [0,1]^d$ represents the scaled residual capacity (i.e., allocatable resources) vector of a machine, $r(i)$ denotes the ith resource dimension of r (e.g., CPU, memory, disk), α and β are user-specified weights of two terms. In Equation 1, the first term is a *Norm* (e.g., *L1* and *L2*) of r which evaluates the sizes of residual capacities. The second term evaluates the fragmentation of r, and f_i denotes a fragmentation parameter, which is tuned based on historical resource requests. For example, if we observe the majority of memory requests are 4GBs, the fragmentation parameter of memory will be the scaled number of 4. In practice, we improve bin packing performance through tuning the *Norm* and weights in Equation 1

Given a database instance $I \in [0,1]^d$, we select a set $\mathcal{M}'$ of machines which I fits in. If $\mathcal{M}'$ is empty, open a new bin (i.e., allocate I to an empty machine). Otherwise, we select candidate machines with the smallest losses (Equation 1). Formally, the residual capacity vector of each candidate machine r_c satisfies:

$$l(r_c - I) - \min_{1 \le i \le |\mathcal{M}'|} l(r_i - I) < \delta \tag{2}$$

where δ is a user-specified small number.

Step 2: The second step uses a skewness-first heuristic. Among the candidate machines, we select the best machine using any of the following policies which differ in measurements of resource skewness:

- **Diagonal Direction Policy**. This policy measures resource skewness by the angle θ between $r - I$ and P (we want θ to be as small as possible). Since $\theta \in [0, \frac{\pi}{2})$, we directly compute and select the machine with the minimum θ:

$$r_s = \arg\min_r \theta(r - I) \tag{3}$$

- **Bottleneck Resource Policy**. This policy measures resource skewness by bottleneck resources. We define the bottleneck resource as the minimum component of $r - I$. Since the bottleneck resource is more likely to be run out, this heuristic selects the machine with the maximum bottleneck resource:

$$r_s = \arg\max_r \min_{1 \le i \le d} (r - I)(i) \tag{4}$$

- **Dot Product Policy**. This policy measures resource skewness by the dot product of r and I. Inspired from [19], this heuristic selects the machine with the maximum dot product:

$$r_s = \arg\max_r I \cdot r \tag{5}$$

Case Study We use a case study to demonstrate the intuition behind VRO online allocation. Figure 4 (a) is a contour line graph of Equation 1 in a 2-dimension space (memory and disk). The dotted curves are the contour lines whereas the darker curves represent larger losses. The blue vector P is the diagonal direction vector. Green circles represent three machines M_1, M_2 and M_3 after loading a database instance I. We use r_1, r_2 and r_3 to denote their residual capacities, respectively. Although r_1, r_2 and r_3 vary a lot in different dimensions, their losses reside within a small range

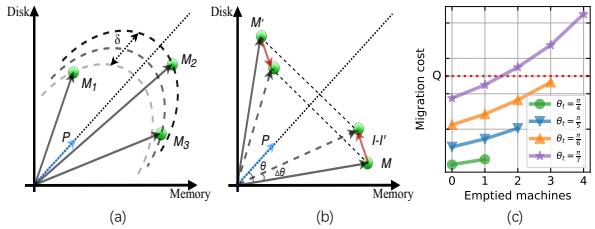

Figure 4: Examples of online allocation (a), instance exchange (b), and binary search of θ_t (c).

δ (Formula 2). In this case, loss-first heuristics lead to a suboptimal solution. Specifically, M_1 has the smallest loss, even though its residual capacity becomes too skewed. To solve this problem, we introduce the second step which uses the skewness-first heuristic to mitigate resource skewness.

To summarize, VRO's online allocation algorithm proposes a novel loss function and inherits the *loss-first* heuristic from best-fit. However, when multiple machines have close losses, skewness becomes the critical criterion. Therefore, in step 2, we switch to the *skewness-first* heuristic and design three policies that measure skewness differently. In this way, the bad case presented in Figure 4 can be solved. For example, when we use the diagonal direction policy, M_2 will be selected since the angle between r_2 and P is the smallest.

4.1.2 Offline optimization. We propose VRO offline optimization, a heuristic offline d-VBP algorithm, which consolidates clusters under migration cost constraints. We formally define the problem of d-VBP with migration cost constraint as follows. Given a cluster $\mathcal{M}$ of m non-empty machines where each residual capacity $r \in [0,1]^d$, a set of n database instances $\mathcal{I}$ where each $I \in [0,1]^d$, and the original instance-machine assignment $A_0 : \mathcal{I} \to \mathcal{M}$, we search the optimal assignment A_{opt} which leads to the minimum non-empty machines under a migration cost constraint $\sum_{I \in \mathcal{I}'} \text{cost}(I) < Q$. In this constraint, the cost is a user-defined metric (e.g., database instance size), $\mathcal{I}'$ represents a set of migrated database instances.

Apparently, it is impractical to directly search for the global optimal solution. The most straightforward heuristic is sorting all machines by the sum of migration costs of the database instances on the machines, then trying to empty one machine by another using d-VBP algorithms. However, this approach does not address migration costs. In this section, we propose VRO offline optimization. Compared to previous works, our method has two advantages. Firstly, instead of traditional d-VBP algorithms, we use VRO online allocation when migrating database instances to reduce resource skewness. Secondly, we introduce instance exchange to further improve bin packing performance, and use binary search to maximize emptied machines under migration cost constraints. Before explaining this algorithm, we introduce two techniques: instance migration and instance exchange.

Instance migration uses a *cost-first* heuristic to empty machines as many as possible. Algorithm 1 describes this algorithm. Given a cluster $\mathcal{M}$ and migration cost constraint Q, we first select the machine M with the lowest migration cost. Next, we try to migrate each instance $I \in M$ and re-allocate it to other non-empty machines

Algorithm 1 Migration phase of VRO offline optimization

```
1: function Migration(M, Q)
2:     while ∃ M ∈ M is not visited do
3:         M ← unvisited machine with the lowest cost in M
4:         for instance I ∈ M do
5:             Migrate and re-allocate I using VRO online
6:         if current cost surpasses Q then                ▷ early terminate
7:             Rollback M
8:             return M
9:         if M is empty then Remove M from M
10:    return M
```

Algorithm 2 Exchange phase of VRO offline optimization

```
1: function Exchange(M, θ_t)
2:     while ∃ M ∈ M is not visited do
3:         M ← unvisited machine with the largest θ in M
4:         while θ > θ_t do
5:             I, I' ← arg max_{I∈M, I'∈M'} Δθ   s.t. Δθ > 0, Δθ' > 0
6:             if ∄ I, I' then break
7:             Exchange instances I (in M) and I' (in M')
8:     return M
```

in $\mathcal{M}$ using VRO online allocation algorithm (Line 4-5). If we find the current migration cost surpasses Q, we roll back M, then terminate (Line 6-8). If M is emptied after instance migration, we remove it from $\mathcal{M}$, then continue to empty the next machine (Line 9).

Instance exchange uses a *skewness-first* heuristic to reduce resource skewness. Algorithm 2 describes this algorithm. Given a cluster $\mathcal{M}$ and an angle threshold θ_t, we first select an unvisited machine M whose r has the largest θ, which evaluates the angle between r and the diagonal direction vector P (Line 3). While θ is greater than θ_t (Line 4), we search a pair of instances $I \in M$ and $I' \in M'$ which leads to an optimal feasible exchange (Line 5). We use $\Delta\theta = \theta(r) - \theta(r + I - I')$ and $\Delta\theta' = \theta(r') - \theta(r' - I + I')$ (i.e., the difference between the θ before exchange and the θ after exchange) to evaluate skewness reductions of M and M'. To improve optimization stability, we constrain that $\Delta\theta > 0$ and $\Delta\theta' > 0$. If no feasible exchange can be found, the skewness of M is already optimized (Line 6); otherwise, we execute this exchange (Line 7).

Figure 4 (b) presents an example of instance exchange. In this figure, black arrows represent residual capacities of machines M and M' before instance exchange, red arrows represent exchanged instances, and black dotted arrows represent residual capacities after instance exchange. In this figure, we can find that after instance exchange, θ (machine M) is reduced by $\Delta\theta$, and the residual capacities of both M and M' move towards P.

Binary search of θ_t. To improve bin packing performance, we run instance exchange before instance migration. To search for the optimal bin packing solution under migration cost constraints, we take advantage of θ_t in Algorithm 2. We observe that smaller θ_t leads to better bin packing performance at the expense of higher migration cost. Therefore, it is critical to find the optimal θ_t. For example, in Figure 4 (c), solid lines represent the migration costs of emptied machines when using different θ_ts, and the dotted line represents the migration cost constraint Q. In this figure, although we can empty more machines when $\theta_t = \frac{\pi}{7}$, under the constraint Q, $\theta_t = \frac{\pi}{6}$ is the best choice (3 emptied machines). In practice, we use binary search to find the optimal θ_t.

Algorithm 3 VRO offline optimization

```
1: function OfflineVRO(M, Q, N, θ_min, θ_max, ϵ)
2:     M^0 ← M
3:     for i ← 1 to N do
4:         M^i ← M^{i-1}
5:         θ_l, θ_r ← θ_min, θ_max
6:         while θ_r − θ_l > ϵ do
7:             M_tmp ← M^{i-1}
8:             θ_t ← (θ_l + θ_r)/2
9:             M_tmp ← Exchange(M_tmp, θ_t)
10:            M_tmp ← Migration(M_tmp, Q)
11:            M^i ← min(M^i, M_tmp)
12:            if migration is early terminated then θ_l ← θ_t
13:            else θ_r ← θ_t
14:        if |M^i| = |M^{i-1}| then break
15:    return M^n
```

Algorithm 3 describes VRO offline optimization. The inputs include a cluster $\mathcal{M}$, migration cost constraint Q, iteration number N, and the binary search parameters $\theta_{\min}$, $\theta_{\min}$ and ϵ. In this algorithm, the outer loop (Line 3-15) iteratively optimizes the cluster, and the inner loop (Line 6-13) searches the optimal θ_t of each iteration. In the inner loop, we initialize the current cluster $\mathcal{M}_{\text{tmp}}$ and θ_t (Line 7-8), try to consolidate $\mathcal{M}_{\text{tmp}}$ through instance exchange and instance migration (Line 9-10), and always retain the smallest consolidated cluster (Line 11). If we find the migration phase is early terminated due to violation of migration cost constraint, it indicates that θ_t should be increased (Line 12). Otherwise, θ_t should be decreased for better bin packing performance (Line 13).

4.2 Management for Warm Resources

In order to optimize warm machines in the warm layer, we propose Exponential Smoothing (ES) with Smoothed Adaptive Margin, a forecasting model optimized for sharp spikes, to predict short-term resource allocation (e.g., memory allocation in the next minute) on node pools. In the following paragraphs, we present our intuition of smoothed adaptive margins.

ES with Constant Margins. Traditional statistic time-series forecasting (e.g., ES, MA, AR) does not predict sharp spikes well. Inspired by Autopilot [24], we adopt a variable *Margin*, and add it to the ES prediction result. In practice, *Margin* is a variable tuned by statistic models. For example, given a time interval T, we collect differences of allocated memory every (i.e., delta memory), fit a probability distribution of T, and select a *Margin* from the distribution (e.g., 90%th quantile). We call this method ES with Constant Margin, and summarize it in the following equations:

$$F_{t+1} = S_{t+1} + Margin \tag{6}$$

where F_{t+1} represents target warm memory at future timestamp $t+1$, S_{t+1} represents ES prediction result:

$$S_{t+1} = \alpha O_t + (1-\alpha) S_t \tag{7}$$

where O_t represents actual allocated memory at current timestamp t, S_t represents prediction result of ES at $t-1$, and α is a user-specified smoothing factor of ES.

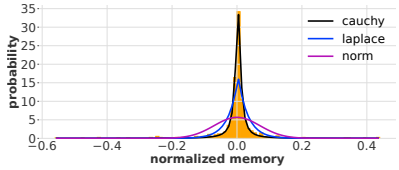

Figure 5: Probability distribution fitting results when adopting different probability distribution models.

ES with Adaptive Margin The drawback of constant margin is lack of flexibility. Observing that the allocated memories rise with acceleration rates, we consider using an adaptive margin where the margin changes based on real-time acceleration rates (i.e., the second-order derivative of allocated memory O''). When O'' is small, the time series is stable, and we use a smaller margin to reduce the cost of warm resources. When O'' is large, based on empirical experience, we assume the time series is likely to retain the current acceleration rate and use a larger margin to improve resource availability. We summarize this method as follows:

$$F_{t+1} = S_{t+1} + (1 + R_{t+1}) \times Margin \tag{8}$$

where R_{t+1} is called adaptive slope and is defined by

$$R_{t+1} = (\frac{O''_t}{O''_{max}})^{\gamma} \tag{9}$$

where O''_t represents current second order derivative of allocated memory, O''_{max} represents the maximum value of historical O''s in a sliding window, and $\gamma \in [0, 1]$ is a user-specified parameter that amplifies R_{t+1}.

ES with Smoothed Adaptive Margin The adaptive margin increases rapidly when the allocated memory is rising with an acceleration rate. However, it also drops rapidly when O''_t is negative and makes the prediction unstable. In practice, we want $Margin$ to decrease smoothly when $O''_t < 0$ since it is risky to sharply reduce warm resources. To solve this problem, based on adaptive margins, we propose ES with Smoothed Adaptive Margin which adopts different adaptive slopes according to signs of O''_t:

$$R_{t+1} = \begin{cases} (\frac{O''_n}{O''_{max}})^{\gamma} & O''_t \geq 0 \\ \beta R_t - (1-\beta)(\frac{|O''_t|}{O''_{max}})^{\gamma} & O''_t < 0 \end{cases} \tag{10}$$

where $\beta \in [0, 1]$ is a user-specified smoothing factor that enables exponential decay of margins when $O'' < 0$. In summary, using ES with Smoothed Adaptive Margins, warm resources cost few when allocated memory is stable (O'' is small), rise rapidly when allocated memory is quickly increasing ($O'' > 0$), and drop smoothly when allocated memory is quickly decreasing ($O'' < 0$).

4.3 Management of Offline Resources

4.3.1 Probabilistic Time Series Forecasting. In order to optimize offline machines in the cold layer, we adopt Temporal Convolutional Network (TCN) [6] to predict long-term resource allocation (e.g., daily memory allocation in the future 21 days) on node pools. Notably, instead of fitting a deterministic curve of future values, our model estimates probability distributions of future values (probabilistic time series forecasting). It captures the uncertainty of the future, and allows users to select the proper quantile of probability distributions. In this section, we introduce two empirical optimizations to improve the performance of TCN:

- **Time series clustering**. Training a model for each time series of a node pool will easily lead to overfitting. On the contrary, the model trained on multiple time series does not perform well due to the heterogeneity of different clusters. In practice, some time series of allocated memory are increasing, some are fluctuating and others are decreasing. In order to solve this problem, we use Dynamic Time Wrapping (DTW) to evaluate the distance between time series and use k-means to cluster them based on DTW evaluations. Finally, we train a model for each cluster.
- **Cauchy Distribution**. Our model estimates probability distributions of the daily difference of memory allocation (i.e., delta memory). Instead of randomly picking a probability distribution, we select three well-adopted candidates and fit their parameters by minimizing the Mean Square Error (MSE) of real-world data. Figure 5 shows the histogram of delta memory (yellow bars) in a node pool, and the fitted probability density functions of Cauchy, Laplace and Normal Distributions. Obviously, Cauchy Distribution best fits the raw data.

4.3.2 Minimum-stock Policy. Based on the predicted time series, we need a mechanism to estimate the optimal cluster size and take corresponding actions (e.g., add or remove machines). This drives us to explore the field of stock/inventory management, a logistics discipline for stocks of goods. However, traditional stock management policies (e.g., Base-stock Policy [2]) rarely consider cases where the stock may instantaneously increase due to "returned" products (e.g., cloud resources are released by end users), therefore, lack mechanisms to remove stocks.

In order to solve this problem, we design Minimum-stock Policy. Algorithm 4 describes this algorithm. In Line 1-4, p and l represent the period time (i.e., time interval) and lead time (i.e., time for the supplier to deliver machines) in days, B represents the current stock size (e.g., cold machines), and pl represents a list of pipeline stock, the resources ordered in the past and will be delivered in the future. In each iteration, we collect the current time T_c (Line 6) and check whether it is time to adjust stocks. If so (Line 7), we firstly compute $s[T_c : T_c + l + p]$, predicted daily delta memory in the future $l + p$ days (Line 8). Next, SimStock computes target stocks and simulated stocks of sub-series $s[T_c : T_c + l]$ and $s[T_c + l : T_c + l + p]$ (Line 9-10).

The function SimStock is described in Algorithm 5. Given predictions of future delta memory, this algorithm simulates how the stock will change if the initial stock is 0. s represents the predicted time series of delta memory, T represents the beginning time of s, and pl represents the pipeline stock list. We initialize two variables: simulated stock sB and minimum stock mB (Line 2-3). Next, we simulate future stocks through traversing s and pipeline stock list pl, and record the minimum of simulated stocks mB (Line 4-6). Finally, we define target stock tB as the opposite of mB (Line 7). It indicates the minimum stock we should maintain in order to prevent insufficient resource events in the future s.size() days. Figure 6 presents an example of simulated stock computing. Solid orange line and solid blue line represent simulated stock series of $s[T_c : T_c + l]$ and $s[T_c + l : T_c + l + p]$. Dotted lines represent simulated stock series if the initial stocks are tB_1 and tB_2.

Algorithm 4 Minimum-stock policy

```
1:  p ← period time
2:  l ← lead time
3:  B ← current stock size
4:  pl ← pipeline stock list
5:  while true do
6:      T_c ← current time
7:      if T mod p = 0 then
8:          s[T_c : T_c + l + p] ← prediction of daily delta memory
9:          tB_1, sB_1 ← SimStock(s[T_c : T_c + l], T_c, pl)
10:         tB_2, sB_2 ← SimStock(s[T_c + l : T_c + l + p], T_c + l, pl)
11:         Δ ← tB_2 − (tB_1 + sB_1)
12:         if Δ ≥ 0 then
13:             if B ≥ tB_1 + Δ then
14:                 B ← tB_1 + Δ                          ▷ remove stocks
15:             else
16:                 pl[T_c + l] ← tB_1 + Δ − B             ▷ add stocks
17:         else
18:             if B ≥ tB_1 then
19:                 B ← tB_1                              ▷ remove stocks
20:             else if B ≤ tB_1 + Δ then
21:                 pl[T_c + l] ← tB_1 + Δ − B             ▷ add stocks
```

Algorithm 5 Compute simulated stock series

```
1: function SimStock(s, T, pl)
2:     sB ← 0                                             ▷ simulated stock
3:     mB ← 0                                             ▷ minimum stock
4:     for i ← T to T + s.size() do
5:         sB ← sB − s[i] + pl[i]
6:         mB ← min(sB, mB)
7:     tB ← −mB                                           ▷ target stock
8:     return tB, sB
```

The intuition behind the Minimum-stock Policy is to maintain minimal stock while preventing insufficient resource events in the future $l+p$ days. Given simulated stocks and Δ (Line 11), we discuss four situations that correspond to four different optimal actions.

- **When $\Delta > 0$**, it indicates the second time period ($[T_c + l : T_c + l + p]$) requires larger stocks than the residual stocks at the end of the first time period ($[T_c : T_c + l]$). For example, in Figure 6, $tB_2 > tB_1 + sB_1$. In order to prevent insufficient resource events in future $l + p$ days, the target stock at T_c should be $tB_1 + \Delta$.
  - **When $B \geq tB_1 + \Delta$**, we should remove $B - (tB_1 + \Delta)$ stocks.
  - **When $B < tB_1 + \Delta$**, we should add $tB_1 + \Delta - B$ stocks, which will be on-hand in l days.
- **When $\Delta \leq 0$**, we only need to ensure that no insufficient resource events will happen in the future l days.
  - **When $B \geq tB_1$**, we remove $B - tB_1$ stocks.
  - **When $B \leq tB_1 + \Delta$**, insufficient resource events are inevitable in the future l days. However, we can add $tB_1 + \Delta - B$ stocks, so that the stocks will be sufficient in the second period.

5 FAST SCHEDULING FOR SERVERLESS WORKLOAD

With the introduction of serverless database services, Eigen is expected to handle requests from both user-specified scaling requests and auto-scaling requests based on metrics collected from serverless database instances, such as their CPU and memory usage. This brings new challenges to the scheduler: how to process resource allocation requests on a more frequent basis, and how to reduce the migration cost caused by cross-node auto-scaling. In this section, we introduce optimizations for overcoming these challenges.

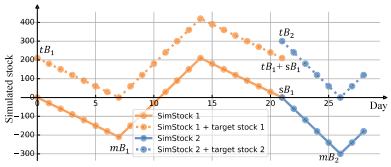

Figure 6: An example of simulated stock series.

5.1 Master-agent Collaborative Scheduling

This design decouples two scheduling procedures: central scheduling and local auto-scaling. The central scheduling processes *creating requests* and *scaling requests* require migration of database instance. As shown in Figure 8 (solid arrows), central scheduling includes the following steps: Step ①, *Eigen Master* receives a creating or scaling request of a database instance. Step ②, based on the node states and the online version of VRO (i.e., allocatable resources of each node), *Scheduler* assigns the database instance to the "best" node. Step ③, *Eigen Master* sends Trylock to request to *Eigen Agent* of the target node. Step ④, *Eigen Agent* checks local allocatable resources. If there are sufficient resources, *Eigen Agent* locks requested resources, then returns success response to *Eigen Master*. If local resources are insufficient, *Eigen Agent* returns fail response. Step ⑤, if Trylock request succeeds, *Eigen Master* executes resource allocation, then builds instances or scales up/down existing instances asynchronously; otherwise, *Eigen Master* asks *Scheduler* to re-schedule.

Local auto-scaling processes the auto-scaling requests that can be fulfilled locally. Similarly, this procedure is depicted in Figure 8 (dotted arrows): Step 1, *Eigen Agent* receives local auto-scaling requests from local database instances. Step 2, *Eigen Agent* checks local allocatable resources. If there are sufficient resources, *Eigen Agent* executes scaling locally. If local resources are insufficient, *Eigen Agent* tries to evict cold instances to release more local resources. After cold instance eviction, if the released resources are still not enough, *Eigen Agent* communicates with *Eigen Master* for cross-node auto-scaling (instance migration). Step 3, *Eigen Agent* asynchronously updates node states to *Eigen Master*.

On the one hand, master-agent collaborative scheduling allows each machine to execute auto-scaling partially without the master node. Therefore, it increases scheduling throughput and reduces the cost of instance migration. However, from the *Scheduler*'s perspective, local auto-scaling may not have the globally optimal decisions. To solve this problem, we implement offline bin packing in *Rebalancer* to periodically consolidate clusters. On the other hand, master-agent collaborative scheduling concerns master-local consistency. To reduce schedule conflict, we implement the Trylock mechanism and require *Eigen Agent* to update node states to *Eigen Master* in real time. First, Trylock ensures that allocated resources

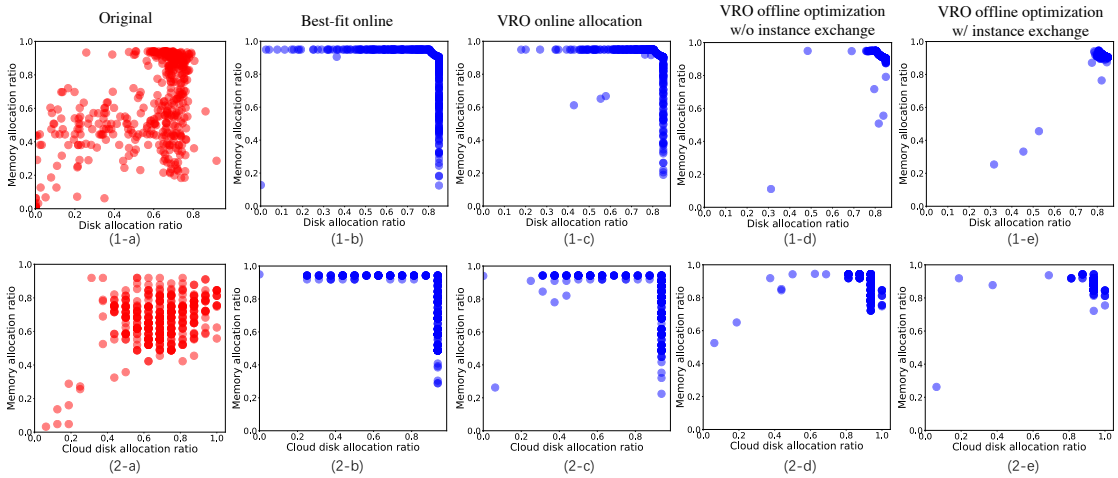

Figure 7: Memory allocation ratios and disk allocation ratios of a physical machine cluster and an ECS machine cluster when using different bin packing algorithms.

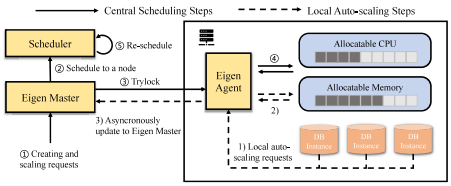

Figure 8: Master-agent collaborative scheduling.

will not be reclaimed by other database instances. Second, the asynchronous update is designed to trade off scheduling efficiency against conflict. In practice, node states are updated every ten seconds, during which Scheduler rarely meets scheduling conflicts caused by lagged versions of node states.

5.2 Cold Instance Eviction

During local auto-scaling, *Eigen Agent* tries to evict cold instances (i.e., migrate less active database instances) if there are no sufficient spare resources on that machine for a scale-up request. We prefer to evict cold instances out of this machine because migrating a "cold" instance costs less than a "hot" one. In practice, we define a metric to reflect the "temperature" of database instances according to multiple factors, such as CPU usage, memory usage, database size, and the number of connections and evaluate the metric periodically. Specifically, we assign the weights to each of the factors and compute the sum of the products of the weight-factor pairs.

6 EVALUATION

In this section, we present evaluation results of Eigen. We launch both simulated and practical experiments on real-world clusters (i.e., node pools) of Alibaba's serverless databases in the production environment. The evaluations of three resource optimization algorithms are demonstrated and discussed in Section 6.1, 6.2, and 6.3 respectively. In Section 6.4, we show how Eigen improves end-to-end resource allocation ratio with little increase of delayed resource requests caused by insufficient resource provision.

6.1 VRO

Experiments of VRO are launched on physical machine clusters and ECS machine clusters. Specifically, on physical machine clusters, multi-dimensional bin packing involves CPU, memory and local disk; on ECS clusters, it involves memory and cloud disk. On ECS clusters, each database instance is associated with several cloud disks (e.g., 1, 2, 4), and each ECS node has a maximum number of cloud disks. On each machine, we constrain that the memory allocation ratio cannot surpass 95%, and the local and cloud disk allocation ratio cannot surpass 85% and 100% respectively.

Resource skewness. We first evaluate VRO by estimating resource skewness when adopting different bin-packing algorithms. Figure 7 presents scatter graphs of a physical machine cluster (upper graphs) and an ECS machine cluster (lower graphs). In each figure, the x-axis represents the local disk or cloud disk allocation ratio, the y-axis represents the memory allocation ratio, and each scatter represents a non-empty machine. Figures 1-a and 2-a show the original clusters without bin packing. Figures 1-b and 2-b show the baseline where the database instances are sequentially requested through a resource scheduler that runs the best-fit online algorithm. Figure 1-c 2-c through Figure 1-e 2-e show the results when the resource scheduler runs different versions of the VRO algorithm. We summarize our observations below: **1.** Bin packing algorithms, even the baseline best-fit online, significantly improve the resource allocation ratios of the whole cluster; **2.** Compared to best-fit, the scatters of machines after using VRO (c, d, e) distribute closer to the diagonal line (particularly the upper right corner). It indicates that VRO can reduce resource skewness and improves allocation ratios. **3.** The VRO offline performs the best, and instance exchange can further reduce resource skewness and improve allocation ratios.

Number of emptied machines. We evaluate the performance of offline bin packing algorithms based on the number of emptied

Table 1: Numbers of emptied machines when using different offline bin packing algorithms, optimal numbers of emptied machines, and total machines of different clusters.

Cluster		First-fit	Best-fit	MostAllocated	RTCR	Diagonal	Diagonal+	Bottleneck	Bottleneck+	DP	DP+	Optimal	Total
Physical machines	1	131	135	131	129	137	140	141	143	136	137	156	445
	2	59	58	59	59	60	60	60	60	63	63	64	158
	3	79	80	80	79	82	82	83	84	84	84	89	211
	4	67	64	64	63	68	69	67	68	71	72	76	203
	5	111	107	107	106	116	117	113	113	111	111	117	298
	6	88	97	97	96	103	103	102	102	101	102	111	350
ECS machines	7	101	99	100	100	108	113	112	117	105	111	141	451
	8	3	3	3	3	4	6	4	6	4	6	20	230
	9	43	41	41	41	45	45	45	46	45	46	52	281
	10	42	29	32	29	43	44	43	43	43	43	96	1476

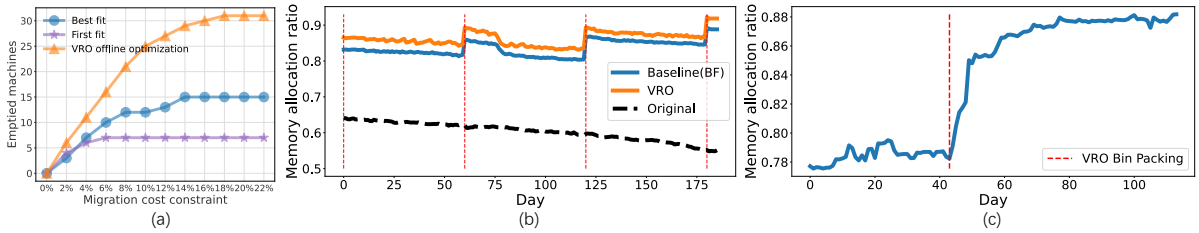

Figure 9: Emptied machines under different migration cost (a), simulated (b) and real-world (c) memory allocation ratios.

machines after consolidating the clusters. Table 1 presents the bin packing results (i.e., emptied machines), the optimal results (i.e., maximum emptied machines), and the total machines of 10 clusters. Specifically, clusters 1-4 are physical clusters that involve two dimensions: memory and disk; clusters 5-6 involve three dimensions: CPU, memory and disk; clusters 7-10 are ECS clusters that involve two dimensions: memory and cloud disk. In each row, gray cells highlight the best bin-packing results of each cluster. We use first-fit, best-fit (*L2 Norm*), and two scheduling strategies from K8s: MostAllocated and RequestToCapacityRatio (RTCR) [15] as baseline policies, then compare them to VRO policies with and without exchange phase. For example, columns "DP" and "DP+" indicate using dot product policy during the migration phase with and without the exchange phase. From this table, we observe that VRO online allocation always outperforms the baseline. Additionally, instance exchange improves bin packing performance regardless of which policy is used during the migration phase.

Next, we evaluate bin packing performance under migration cost constraints. In this experiment, we compare VRO offline optimization with two baseline algorithms, which adopt best-fit and first-fit during the migration phase. Figure 9 (a) presents the number of emptied machines under different migration cost constraints (percentiles of the sum of migration costs of all database instances). We observe that under the same migration cost constraint, VRO offline optimization always outperforms baselines.

Memory allocation ratio of online machines. We show how VRO improves allocation ratios of online machines of real-world clusters. We collect the trace of the actual history of Eigen allocation for six months and simulate the performance of scheduling based on the trace. When computing the simulated memory allocation ratio, we re-allocate database instances using bin packing algorithms. Specifically, we consolidate (e.g., using VRO offline optimization) the cluster every two months (red vertical lines). Between each consolidation, we use online bin-packing algorithms to allocate database instances. Figure 9 (b) presents simulated memory allocation ratios when using best-fit (blue line), VRO (orange line), and the actual allocation ratio (black dashed line) of online machines. We can observe that the consolidation of the cluster (either best-fit or VRO) sharply increases the memory allocation ratio. Compared to the original line, bin packing algorithms can significantly improve the allocation ratio. Compared to best-fit, VRO further improves the memory allocation ratio by reducing resource skewness.

Figure 9 (c) presents the memory allocation ratio of a cluster from our production environment. In this figure, the red vertical line indicates the timestamp when we applied VRO. Explicitly, at that time, we started to use VRO online allocation for real-time resource requests and began rebalancing using VRO offline optimization. Due to the high cost of database instance migration, it is impractical to consolidate the whole cluster instantly. Therefore, we can observe that the memory allocation ratio gradually rises during the first 40 days when we gradually migrated database instances.

6.2 ES with Smoothed Adaptive Margins

Predicted time series Figure 10 presents the memory of actual allocation and prediction of a real-world cluster when using ES with constant margins (a), ES with adaptive margins (b) and ES with smoothed adaptive margins (c). Red circles highlight timestamps when the prediction is surpassed by actual allocated memory (i.e., insufficient resource events). In Figure 10 (a), insufficient resource events happen four times (A, B, C, and D) when the actual allocated memory rapidly rises. When predicting memory allocation of timestamps A, B and C (i.e., at one timestamp earlier), the actual allocation series has large accelerations (i.e., second-order derivatives). Constant margins cannot handle this situation. In Figure 10 (b), ES with adaptive margins overcomes failures at A, B and C. However, its predicted series becomes unstable and is surpassed by the actual allocation at timestamps D and E when the accelerations are negative. In Figure 10 (c), ES with smoothed adaptive margins

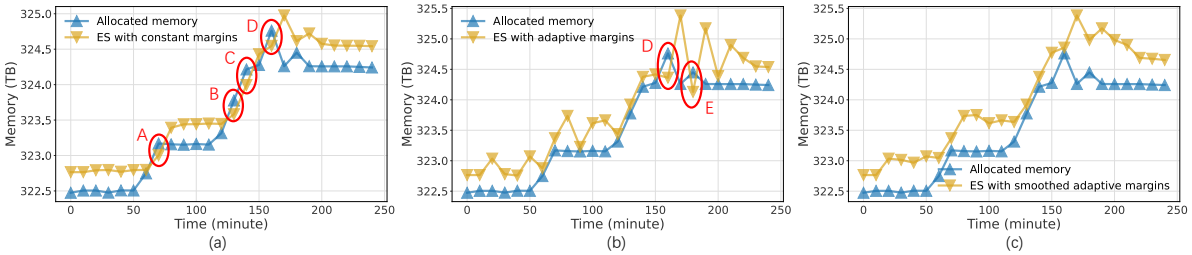

Figure 10: Memory of actual allocation and prediction.

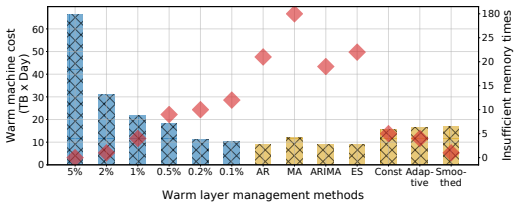

Figure 11: The cost of warm machines (bars) and times of insufficient resource events (dots).

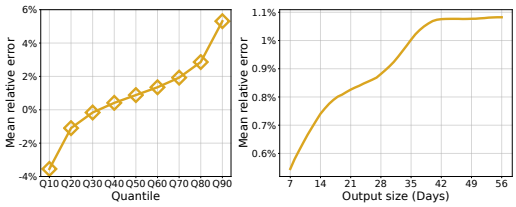

Figure 12: Mean relative error of TCN prediction.

overcomes all five failures. We can observe that the margin drops smoothly when the accelerations are negative (e.g., at timestamp D and E), thus the predicted series become much more smoothing.

Warm resource cost and insufficient resource events We show how ES with smoothed adaptive margins solves the dilemma of maximizing resource utilization or availability in the warm layer. In Figure 11, the left y-axis measures the bars of cost of warm machines (memory times days), and the right y-axis measures the dots of insufficient resource events during a month when using different warm resource management methods. Blue bars represent the methods that maintain warm machines of constant percentages of allocated memory (e.g., 5%, 2%, 1%). Yellow bars represent classic time series prediction methods: AR (AutoRegressor), MA (Moving Average), ARIMA, ES, and Eigen's methods: ES with constant margins, adaptive margins and smoothed adaptive margins. Compared to the methods using constant percentages, ES with smoothed adaptive margins strikes a good balance between reducing the cost and reducing insufficient resource events. Furthermore, out method significantly reduces insufficient resource events with little increase in cost, when compared to other ES variants.

6.3 TCN and Minimum-stock Policy

Prediction accuracy We divide time series data of daily allocated memory into a training set and a testing set (4:1), then evaluate the prediction accuracy of TCN model on the testing set. Figure 12 shows Mean Relative Error (MRE), which measures the deviation of the predictions from the actual data, when using different quantiles (a) and output sizes (b). In Figure 12 (a), we set the output size 28 (days), and we can find that MRE is less than 1% when using 50%th quantile (Q50) of predicted distributions. In practice, to prevent failed requests, we normally select higher quantiles, such as Q60 or even Q70. Next, in Figure 12 (b), we use Q50 of the predicted distributions, and we can find that prediction accuracy mildly decays when the output size increases. MRE is no more than 1.1% when the model predicts allocated memory in the future 50+ days.

Predicted time series Figure 14 (a) shows the memory of actual allocation and prediction of a real-world cluster when using Base-stock Policy and Minimum-stock Policy. Notably, the predicted series is tuned by different stock management policies. In this figure, we set period time $p = 7$ and lead time $l = 21$. We can observe that the predicted series of Minimum-stock Policy proactively adds offline machines before the actual allocation increases, and removes offline machines before the actual allocation decreases. In addition, compared to Base-stock Policy, the prediction series of Minimum-stock Policy is more accurate. In particular, when the actual allocation is decreasing, Base-stock Policy lacks a mechanism to remove offline machines, thus causing much more cost.

Offline resource cost and insufficient resource days We demonstrate how TCN with Minimum-stock Policy solves the dilemma of maximizing resource utilization or availability in the offline layer. Figure 14 (b) presents the costs of offline machines and days of insufficient resource events when using different offline resource management methods for three months. We first compare prediction-based methods to the methods using constant percentages. When the percentages are large (e.g., 5%, 4%), prediction-based methods cost much lower while having comparable insufficient resource days. When the percentages are small (e.g., 0.5%, 0.1%), prediction-based methods have much fewer insufficient resource days while the costs are close. Moreover, compared to Base-stock Policy, Minimum-stock Policy has better performance regarding both cost and insufficient resource days.

6.4 End-to-end Evaluation

We select three large-scale representative clusters, whose memory allocations are decreasing, fluctuating and increasing respectively, to present end-to-end evaluation results of Eigen. Figure 13 presents the memory of actual allocation, configured machines (i.e., online

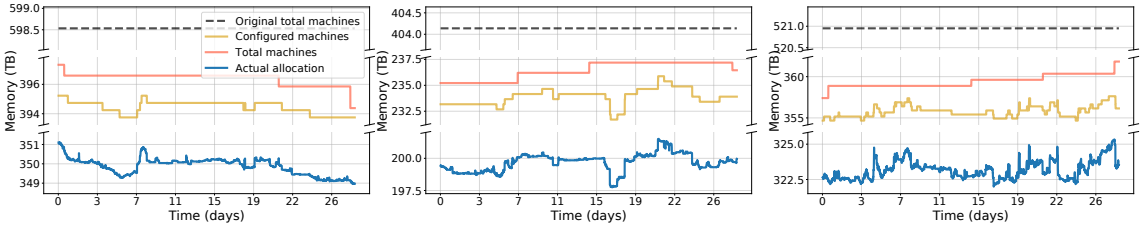

Figure 13: Memory of actual allocation, configured machines, total machines and original total machines.

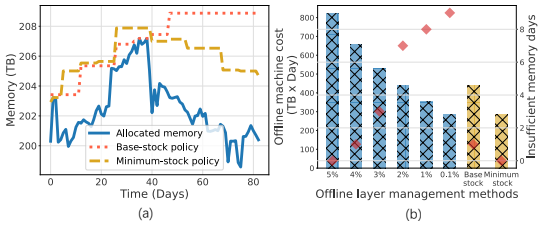

Figure 14: Memory of actual allocation and prediction.

& warm machines), total machines (i.e., online & warm & offline machines) during a month after using Eigen, and the original total machines before Eigen. In these experiments, we compute ES with smoothed adaptive margins every ten minutes and adjust total machines every seven days. From the figures, we can observe that both configured machines and total machines are proactively adjusted before the actual allocated memory changes. Compared to the original total machines, Eigen significantly decreases stranded resources to save the expenses of clusters.

Table 2 presents memory allocation ratios, memory utilization ratios, and ratios of delayed resource requests of three clusters in Figure 13 (Cluster a, b, c) before and after using Eigen. In each cluster, VRO improves allocation ratios and utilization ratios of non-empty machines, and the other two resource optimization algorithms improve allocation ratios by maintaining *just enough* empty and offline machines. On the one hand, after using Eigen, both allocation ratios and utilization ratios are significantly improved. On average, the allocation ratio is improved to 87%, and the utilization ratio is improved to 61.37%. On the other hand, after using Eigen, a few requests were delayed due to the mis-prediction of ES with smoothed adaptive margins. Moreover, no request failed because of insufficient resource provision before and after Eigen. To summarize, Eigen significantly improved resource utilization with a little rise of delayed/failed resource requests.

7 RELATED WORK

There are previous works of *d*-VBP heuristics that discuss resource skewness. Zhang et al. [31] propose an FFD variant that clusters Virtual Machines (VM) by their dominant resources (i.e., the largest component of VM's resource vector) into groups, then allocates each group based on the residual capacities of the whole cluster of Physical Machines (PM). Hieu et al. [10] propose Max-BRU, a bin-centric heuristic that searches for the most appropriate VM for the most "suitable" PM. In this paper, they adopt Resource Balance (RB), a resource skewness metric, as a scoring strategy for PMs.

Table 2: Memory allocation ratios, utilization ratios and delay ratios before and after deploying Eigen.

	allocation ratio		utilization ratio		Delayed ratio	
Cluster	Before	After	Before	After	Before	After
a	58.73%	88.01%	50.41%	75.55%	0%	0.09%
b	61.40%	83.7%	37.34%	51.88%	0%	0%
c	59.01%	89.31%	36.83%	56.68%	0%	0.05%

These two algorithms reduce resource skewness, however, they are offline algorithms and do not address migration costs. Li et al. [17] propose a space partition model for online VM allocation. The model divides PMs into three domains based on the size and skewness of machines' resource usage, where the domains have different scheduling priorities. Compared to this algorithm, VRO online allocation is also a mixture of loss-first and skewness-first heuristics, but does not rely on rigid domain partition.

Some previous works use statistical and machine learning techniques to optimize utilization ratios. Google's Autopilot [24] enables vertical scaling (i.e., tuning the number of resources allocated for jobs/tasks) through an exponentially-smoothed sliding window over historical usage to decide the optimal resource limit of each job/task. FIRM [21] uses a reinforcement learning model (i.e., DDPG) to make dynamic resource provision decisions for microservices. P-Store [27] uses SPAR to predict periodic workloads of database instances. Based on workload prediction, it can proactively add/reduce resource provision before the workload of a database instance changes. Moneyball [20] uses statistic models to learn pause/resume patterns of serverless database instances, therefore supporting proactive resumes to reduce delays of resource provision. Although effective, none of them has optimizations for sharp spikes in short-term forecasting and long-term cloud resource control.

8 CONCLUSION

In this paper, we present Eigen, a large-scale, cloud-native cluster management system for Alibaba's cloud database. Eigen adopts a hierarchical resource management system with three novel resource optimization algorithms: Vectorized Resource Optimization, ES with Smoothed Adaptive Margins and TCN with Minimum-stock Policy, which enable end-to-end resource optimization. In addition, we present Eigen's system optimizations for serverless database services that promote user experience by reducing resource scheduling latencies and improving scheduling throughput. The evaluation results show that Eigen significantly improves the resource allocation ratio without hurting resource availability.

REFERENCES

[1] Yair Amir, Baruch Awerbuch, Amnon Barak, R Sean Borgstrom, and Arie Keren. 2000. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on parallel and distributed Systems* 11, 7 (2000), 760–768.

[2] N Anbazhagan, Jinting Wang, and D Gomathi. 2013. Base stock policy with retrial demands. *Applied Mathematical Modelling* 37, 6 (2013), 4464–4473.

[3] AWS. Amazon Aurora Serverless. https://aws.amazon.com/rds/aurora/serverless

[4] Azure. Azure SQL Serverless. https://learn.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview

[5] Nikhil Bansal, Marek Eliáš, and Arindam Khan. 2016. Improved Approximation for Vector Bin Packing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms* (Arlington, Virginia) *(SODA '16)*. Society for Industrial and Applied Mathematics, USA, 1561–1579.

[6] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. 2020. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing* 399 (2020), 491–501. https://doi.org/10.1016/j.neucom.2020.03.011

[7] Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. 2017. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review* 24 (2017), 63–79. https://doi.org/10.1016/j.cosrev.2016.12.001

[8] Alibaba Cloud. Alibaba Cloud RDS MySQL Serverless. https://help.aliyun.com/document_detail/411291.html

[9] etcd. etcd. https://etcd.io/

[10] Nguyen Trung Hieu, Mario Di Francesco, and Antti Ylä Jääski. 2014. A virtual machine placement algorithm for balanced resource utilization in cloud data centers. In *2014 IEEE 7th International Conference on Cloud Computing*. 474–481. https://doi.org/10.1109/CLOUD.2014.70

[11] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for {Fine-Grained} Resource Sharing in the Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.

[12] Rebecca Hoberg and Thomas Rothvoss. 2017. A Logarithmic Additive Integrality Gap for Bin Packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. Society for Industrial and Applied Mathematics, USA, 2616–2625.

[13] Kubernetes. Kubernetes. https://kubernetes.io/

[14] Kubernetes. Kubernetes Scheduler. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

[15] Kubernetes. Resource Bin Packing. https://kubernetes.io/docs/concepts/scheduling-eviction/resource-bin-packing/

[16] Vincent Le Guen and Nicolas Thome. 2019. Shape and time distortion loss for training deep time series forecasting models. *Advances in neural information processing systems* 32 (2019).

[17] Xin Li, Zhuzhong Qian, Sanglu Lu, and Jie Wu. 2013. Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center. *Mathematical and Computer Modelling* 58, 5-6 (2013), 1222–1235.

[18] Bryan Lim, Sercan Ö. Arık, Nicolas Loeff, and Tomas Pfister. 2021. Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting* 37, 4 (2021), 1748–1764. https://doi.org/10.1016/j.ijforecast.2021.03.012

[19] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. 2011. Heuristics for Vector Bin Packing. (January 2011). https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/

[20] Olga Poppe, Qun Guo, Willis Lang, Pankaj Arora, Morgan Oslake, Shize Xu, and Ajay Kalhan. 2022. Moneyball: Proactive Auto-Scaling in Microsoft Azure SQL Database Serverless. *Proc. VLDB Endow.* 15, 6 (feb 2022), 1279–1287. https://doi.org/10.14778/3514061.3514073

[21] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 805–825. https://www.usenix.org/conference/osdi20/presentation/qiu

[22] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. Deep State Space Models for Time Series Forecasting. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/5cf68969fb67aa6082363a6d4e6468e2-Paper.pdf

[23] Thomas Rothvoß. 2013. Approximating bin packing within o (log OPT* log log OPT) bins. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 20–29.

[24] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. 2020. Autopilot: Workload Autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 16, 16 pages. https://doi.org/10.1145/3342195.3387524

[25] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.

[26] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. 2010. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and distributed Computing* 70, 9 (2010), 962–974.

[27] Rebecca Taft, Nosayba El-Sayed, Marco Serafini, Yu Lu, Ashraf Aboulnaga, Michael Stonebraker, Ricardo Mayerhofer, and Francisco Andrade. 2018. P-Store: An Elastic Database System with Predictive Provisioning. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 205–219. https://doi.org/10.1145/3183713.3190650

[28] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: A Unified Cluster Management System for Shared Infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 787–803. https://www.usenix.org/conference/osdi20/presentation/tang

[29] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–16.

[30] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-Scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[31] Yan Zhang and Nirwan Ansari. 2013. Heterogeneity aware dominant resource assistant heuristics for virtual machine consolidation. In *2013 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1297–1302.

[32] Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. 2014. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. In *Proceedings of the VLDB Endowment*, Vol. 7. VLDB Endowment Inc., 1393–1404.