OptimusCloud: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud

Ashraf Mahgoub and Alexander Michaelson Medoff, *Purdue University;* Rakesh Kumar, *Microsoft;* Subrata Mitra, *Adobe Research;* Ana Klimovic, *Google Research;* Somali Chaterji and Saurabh Bagchi, *Purdue University*

https://www.usenix.org/conference/atc20/presentation/mahgoub

This paper is included in the Proceedings of the 2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the 2020 USENIX Annual Technical Conference is sponsored by USENIX.

OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud

Ashraf Mahgoub, *Purdue University*
Alexander Medoff, *Purdue University*
Rakesh Kumar, *Microsoft*
Subrata Mitra, *Adobe Research*
Ana Klimovic, *Google Research*
Somali Chaterji, *Purdue University*
Saurabh Bagchi, *Purdue University*

Abstract

Achieving cost and performance efficiency for cloud-hosted databases requires exploring a large configuration space, including the parameters exposed by the database along with the variety of VM configurations available in the cloud. Even small deviations from an optimal configuration have significant consequences on performance and cost. Existing systems that automate cloud deployment configuration can select near-optimal instance types for homogeneous clusters of virtual machines and for stateless, recurrent data analytics workloads. We show that to find optimal performance-per-$ cloud deployments for NoSQL database applications, it is important to (1) consider heterogeneous cluster configurations, (2) jointly optimize database and VM configurations, and (3) dynamically adjust configuration as workload behavior changes. We present OPTIMUSCLOUD, an online reconfiguration system that can efficiently perform such joint and heterogeneous configuration for dynamic workloads. We evaluate our system with two clustered NoSQL systems: Cassandra and Redis, using three representative workloads and show that OPTIMUSCLOUD provides 40% higher throughput/$ and 4.5× lower 99-percentile latency on average compared to state-of-the-art prior systems, CherryPick, Selecta, and SOPHIA.

1 Introduction

Cloud deployments reduce initial infrastructure investment costs and provide many operational benefits. An important class of cloud deployments is NoSQL databases, which allow applications to scale beyond the limits of traditional databases [17]. Popular NoSQL databases such as Cassandra, Redis, and MongoDB, are widely used in web services, big data services, and social media platforms. Tuning cloud-based NoSQL databases for performance[1] under cost constraints is challenging due to several reasons.

[1]We use the standard metrics of throughput and (tail) latency for measuring database performance. Specifically, we target maximizing throughput normalized by price in $, i.e., performance-per-unit-$ or Perf/$ for short.

First, the search space is very large due to VM configurations and database application configurations. For example, cloud services provide many VMs that vary in their CPU-family, number of cores, RAM size, storage, network bandwidths, etc., which affect the VM's $ cost. At the time of writing, AWS provides 133 instance types while Azure provides 146 and their prices vary by a factor of 5,000×. On the NoSQL side, there are many performance-impacting configuration parameters. For example, Cassandra has 25 such parameters and sub-optimal parameter setting for one parameter (e.g., the *Compaction method*) can degrade throughput by 3.4× from the optimal. On the cloud side too, selecting the right VM type and size is essential to achieve the best Perf/$.

Second, there is the need for *joint* optimization while taking into account the dependencies between the NoSQL-level and VM-level configurations. For example, our evaluation shows that the optimal cache size of Cassandra for a VM type `M4.large` (with 8GB of RAM) is 8× the optimal cache size for `C4.large` (with 3.75GB RAM). Additionally, larger-sized VMs do not always provide better Perf/$ [67] as they may overprovision resources and unnecessarily increase the $ cost.

Third, there are many use cases of cloud applications where the workload characteristics change over time, sometimes unpredictably, necessitating reconfigurations [7, 11]. A configuration that is optimal for one phase of the workload can become very poor for another phase of the workload. For example, in Cassandra, with a large working set size, reads demand instances with high memory, while writes demand high compute power and fast storage.

Changing the configuration at runtime for NoSQL databases, which are stateful applications (i.e., with persistent storage), has a performance impact due to the downtime caused to the servers being reconfigured. Therefore, for fast changing workloads, frequent reconfiguration of the overall cluster could severely degrade performance [41]. Consequently, deciding *which* subset of servers to reconfigure is vital to minimize reconfiguration performance hit and to achieve globally optimal Perf/$ while respecting the user's availability requirements. However, changing the configurations of only

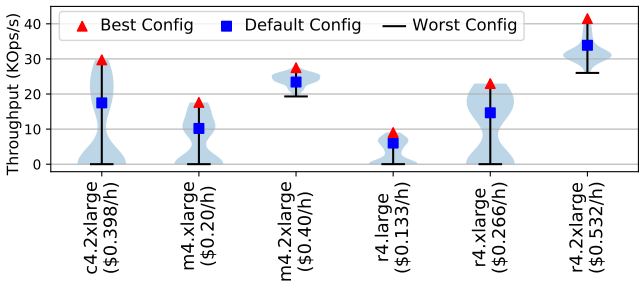

Figure 1: Violin plot showing performance (throughput) of Best, Default, and Worst database configurations across different EC2 VM types.

Feature	Single Server Prediction	Multiple Server Prediction (Homogenous)	Multiple Server Prediction (Heterogeneous)	Application Level Configurations	Cloud Configurations	Dynamic Workloads + Cost Benefit Analysis
Ernest (NSDI'16) CherryPick (NSDI'17)	✓	✓	✗	✗	✓	✗
Selecta (ATC'18)	✓	✓	◐	✗	✓	✗
Rafiki (Middleware'17) / Ottertune (Sigmod'17)	✓	✗	✗	✓	✗	✗
Sophia (Usenix ATC'19)	✓	✓	✗	✓	✗	◐
OPTIMUSCLOUD	✓	✓	✓	✓	✓	✓

✓ Supported ◐ Partially Supported ✗ Not Supported

Table 1:* OPTIMUSCLOUD*'s key features vs. existing systems.

a subset of servers naturally leads to *heterogeneous* clusters, which no prior work is equipped to deal with.

Existing Solutions: State-of-the-art cloud configuration tuners such as CherryPick [5] and Selecta [32] focus mainly on stateless, recurring workloads, such as big-data analytics jobs, while Paris [67] relies on a carefully chosen set of benchmarks that can be run offline to fingerprint which application is suitable for which VM type. Due to their target of static workloads and stateless jobs, a single cloud configuration is selected based on a representative workload and then fixed throughout the operation period. However, small workload changes can cause these "static tuners" to produce drastically degraded configurations. For example, a 25% increase in workload size with CherryPick makes the proposed configuration 2.6× slower than optimal (Section 5.4 in [5]). Also, in our experiments (Sec. 4.3), we find that CherryPick's proposed configuration for the write-heavy phase achieves only 12% of the optimal when the workload switches to a read-heavy phase. Hence, these *prior systems are not suitable for dynamic workloads*.

SOPHIA [41] addresses database configuration tuning for clustered NoSQL databases and can handle dynamic workloads. However, like the static tuner RAFIKI [40], SOPHIA's design focuses only on NoSQL configuration tuning and does not consider cloud VM configurations nor dependencies between VM and NoSQL configurations. Naïvely combining the NoSQL and VM configuration spaces causes a major increase in the search space size and limits SOPHIA's ability to provide efficient configurations (Sec. 3.5). Further, due to its atomic reconfiguration strategy (i.e., either reconfigure all servers or none), it suffers from all the drawbacks of the homogeneity constraint. Table 1 compares key features of OPTIMUSCLOUD to various prior works in this field.

Our Solution: We introduce our system OPTIMUSCLOUD, which jointly tunes the database and cloud (VM) configurations for dynamic workloads. **There are three key animating insights behind the design of** OPTIMUSCLOUD**.** The *first* is that jointly tuning the database and cloud (VM) configurations for dynamic workloads is essential. To show how important this is, we benchmark one Cassandra server with a 30-min trace from one of our three workloads (MG-RAST) on 9 different EC2 VM types[2]. For each type, we use 300 different database configurations selected through grid search. We show the performance in terms of Ops/s for the best, default, and worst configurations in Fig. 1. We see a big variance in performance w.r.t. the database configurations—up to 74% better performance over default configurations (45% on average). Further, the best configurations vary with the VM type and size (for the 6 VM types shown here, there are 5 distinct best DB configurations). This emphasizes the need for tuning both types of configurations *jointly* to achieve the best Perf/$. The *second key insight* is that in order to optimize the Perf/$ for a dynamic workload, it is necessary to perform non-atomic reconfigurations, i.e., for only part of the cluster. Reconfiguration in a distributed datastore is a sequential operation (in which one or a few servers at a time are shutdown and then restarted) to preserve data availability [31, 41]. This operation causes transient performance degradation or lower fault tolerance. Reconfiguration is frequent enough for many workloads that this performance degradation should be avoided, e.g., MG-RAST has a median of 430 significant switches per day in workload characteristics. Accordingly, *heterogeneous configurations* have the advantage of minimizing the performance hit during reconfiguration. Further, in the face of dynamic workloads, there may only be time to reconfigure part of the overall cluster. Also, from a cost-benefit standpoint, maximizing performance does not need *all* instances to be reconfigured (such as to a more resource-rich instance type), rather a carefully selected subset. We give a simple example to make this notion concrete in Section 2. The *third key insight* is that for a particular NoSQL database (with its specifics of data placement and load balancing), it is possible to create a model to map the configuration parameters to the performance of each server. From that, it is possible to determine the overall heterogeneous cluster's performance. OPTIMUSCLOUD leverages performance modeling to search for the optimal cluster configuration.

The workflow of OPTIMUSCLOUD comprises offline training and online prediction and reconfiguration phases as shown in Fig. 2. At runtime, OPTIMUSCLOUD takes user require-

[2]MG-RAST is the largest metagenomics portal and data repository and gets queries from across the globe which cause unpredictable read-write patterns to the backend Cassandra.

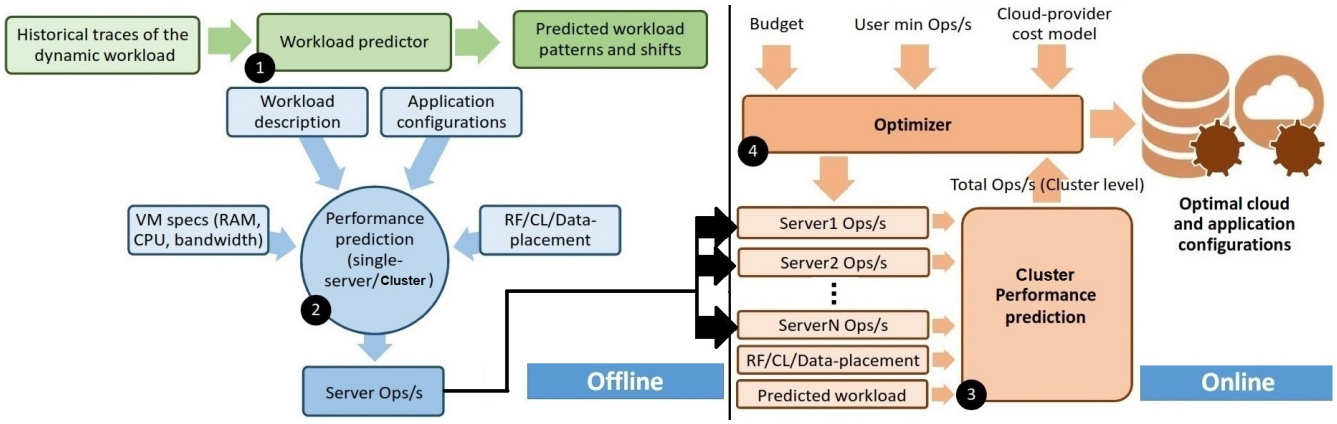

Figure 2: Overview of OPTIMUSCLOUD*'s workflow. First, a workload predictor is trained with historical traces from the database to be tuned. Second, a single server performance predictor is trained to map workload description, VM specs, and NoSQL application configuration to throughput. Third, a cluster-level performance predictor is used to estimate the throughput of the heterogeneous cluster of servers. In the online phase, our optimizer uses this predictor to evaluate the fitness of different VM/application configurations and provides the best performance within a given budget.*

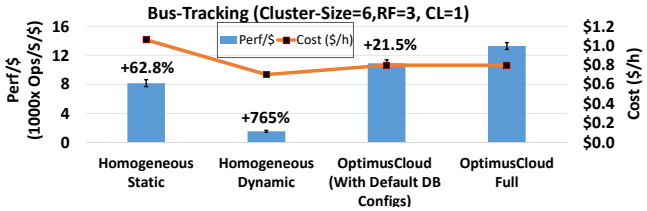

*Figure 3: Importance of creating heterogeneous clusters (*OPTIMUSCLOUD *Full) over homogeneous clusters (both Static and Dynamic) for Bus-Tracking application. Tuning both application and cloud configurations (*OPTIMUSCLOUD *Full) has benefit over tuning only the VM configuration (3rd bar from left). The percentage value on the top of each bar denotes how much* OPTIMUSCLOUD *improves over that particular scheme.*

ments of budget, availability, and consistency. It then combines the performance model with a workload predictor and a cost-benefit analyzer to decide: when the workload changes sufficiently, what should be the new (possibly heterogeneous) configuration. It decides what minimal set of servers should be reconfigured.

Evaluation: We apply OPTIMUSCLOUD to two popular NoSQL databases—Cassandra and Redis—and evaluate the system on traces from two real-world systems, and one simulated trace from an HPC analytics job queue. All three use cases represent dynamic workloads with different query blends. We evaluate the Perf/$ achieved by OPTIMUSCLOUD and compare this to three leading prior works, CherryPick [5], Selecta [32], and SOPHIA [41]. Additionally, we compare ourselves to the best static configuration determined with oracle-like prediction of future workloads and the theoretical best. OPTIMUSCLOUD achieves between 80-90% of the theoretical best performance for the 3 workloads and achieves improvements between 9%-86.5%, 18%-173%, 17%-174%, and 12%-514% in Perf/$ over Homogeneous-Static, CherryPick, Selecta, and SOPHIA respectively without degrading P99 latency (Sec. 4). Fig. 3 shows the improvement in Perf/$ due to OPTIMUSCLOUD's heterogeneous configurations.

We make the following novel contributions in this paper.

1. We design a performance modeling-based technique for efficient *joint optimization* of database *and* cloud configurations to maximize the Perf/$ of a clustered database.

2. We design a technique to identify the minimal set of servers in a clustered database to reconfigure (concurrently) to obtain a throughput benefit. This naturally leads to heterogeneous configurations. To reduce the much larger search space that this causes, we design for a simplification that groups multiple servers that should be configured to the same parameters.

3. We show that OPTIMUSCLOUD generalizes to two distinct NoSQL databases and different workloads, cluster sizes, data volumes, and user-specified requirements for replication and data consistency.

The rest of the paper is organized as follows. Section 2 gives the necessary background for the problem and a quantitative rationale. Section 3 describes the details of OPTIMUSCLOUD's design. We evaluate OPTIMUSCLOUD in Section 4 and survey related work in Section 5.

2 Background and Rationale

To evaluate the generalizability of OPTIMUSCLOUD, we select two popular NoSQL databases with very different architectures.

2.1 Cassandra

Cassandra is designed for high scalability, availability, and fault-tolerance. To achieve these, Cassandra uses a peer-to-peer (P2P) replication strategy, allowing multiple replicas to handle the same request. Other popular datastores such as DynamoDB [20] and Riak [34] implement the same P2P strategy and we select Cassandra as a representative system from

that category. Cassandra's replication strategy determines where replicas are placed. The number of replicas is defined as "Replication Factor" (RF). By default, Cassandra assigns an equal number of tokens to each node in the cluster where a token represents a sequence of hash values for the primary keys that Cassandra stores. Based on this token assignment, a Cassandra cluster can be represented as a ring topology [16]. Fig. 5 shows an example of 4 Cassandra servers and RF of 2.

2.2 Redis

Redis is an in-memory database and serves all requests from the RAM, while it writes data to permanent storage for fault tolerance. This design principle makes Redis an excellent choice to be used as a cache on top of slower file systems or datastores [54]. Redis can operate as either a stand-alone node or in a cluster of nodes [53] where data is automatically sharded across multiple Redis nodes. Our evaluation applies to the clustered mode of Redis. When a Redis server reaches the maximum size of its allowed memory (specified by the `maxmemory` configuration parameter), it uses one of several policies to decide how to handle new write requests. The default policy will respond with error. Other policies will replace existing records with the newly inserted record (the `maxmemory-policy` configuration parameter specifies which records will be evicted). The value of `maxmemory` needs to be smaller than the RAM size of the VM instance and the headroom that is needed is workload dependent (lots of writes will need lots of temporary buffers and therefore larger headroom). Thus, it is challenging to tune `maxmemory-policy` and `maxmemory` parameters with changing workloads and these two form the target of our configuration decision.

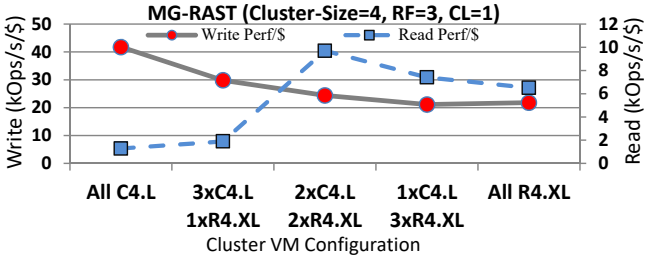

Figure 4: Change in Perf/$ for the write (solid) and read throughput (dotted) as we reconfigure the nodes from C4.large to R4.xlarge.

2.3 Example Rationale for Heterogeneous Configurations

Here we give a motivating example for selecting subset of servers to reconfigure. Consider a Cassandra cluster of 4 nodes with a consistency-level (CL[3]) = 1 and replication-factor (RF[4]) = 3, i.e., any pair of nodes has a complete copy of all the data. Also, assume that we only have two cloud configurations: C4.large, which is compute-optimized, and R4.xlarge, which is memory-optimized. C4.large is cheaper than R4.xlarge by 58% [8], whereas R4.xlarge has larger RAM (30.5GB vs 3.75GB) and serves read-heavy workloads with higher throughput. Now we test the performance of all possible combinations of VM configurations (All C4.L, 1 C4.L + 3R4.XL, ... etc.) for both read-heavy and write-heavy phases of the MG-RAST workload and show the saturation level throughput for each configuration in Fig. 4. The "All C4.large" configuration achieves the best write Perf/$ (41.7 KOps/s/$), however, it has the worst read Perf/$ (only 1.28 KOps/s/$) because reads of even common records spill out of memory. Now if two servers are reconfigured to R4.xlarge, the write Perf/$ decreases (24.4 KOps/s/$), while the read performance increases significantly (9.7 KOps/s/$), showing an improvement of 7.5× for read throughput over the all C4.large configuration. The reason for this huge improvement is Cassandra's design by which it redirects new requests to the fastest replica [19], directing all read requests to the two R4.xlarge servers. *Now we notice that switching more C4.large servers to R4.xlarge does not show any improvement in either reads or writes Perf/$, as the two R4x.large servers are capable of serving the applied workload with no queued requests.* This means that switching more servers will only reduce the Perf/$. Thus, the best Perf/$ is achieved by configuring to all C4.large in write-heavy phases, while configuring only 2 servers to R4x.large in read-heavy phases. **Therefore, heterogeneous configurations can achieve better Perf/$ compared to homogeneous ones under mixed workloads.**

[3]CL: the minimum number of Cassandra nodes that must acknowledge a read or write operation before the operation can be considered successful

[4]RF: the total number of replicas for a key across a Cassandra cluster

3 Design

3.1 Workload Representation and Prediction

OPTIMUSCLOUD uses a query-based model [4] to represent time-varying workloads. This model characterizes the applied workload in terms of the proportion of the different query types and the total volume of queries, denoted by W.

We use a workload predictor to learn time-varying patterns from the workload's historical traces, and predict the workload characteristics for a particular lookahead period. We notate the time varying workload at a given point in time t as $W(t)$. The task of the workload predictor is to provide OPTIMUSCLOUD with $W(t+1)$ given $W(t)$,$W(t-1)$,... , $W(t-h)$, where h is the length of history. OPTIMUSCLOUD then iteratively predicts the workload till a lookahead time l, i.e., $W(t+i), \forall i \in (1,l)$. We execute OPTIMUSCLOUD with a simple Markov-Chain prediction model for both MG-RAST and Bus-tracking workloads while we have a deterministic fully accurate predictor for HPC. We do not claim any novelty in workload prediction and OPTIMUSCLOUD is modular enough to easily integrate more complex estimators, such as

neural networks [37, 39].

3.2 Performance Prediction

Combining NoSQL and cloud configurations produces a massive search space, which is impractical to optimize through exhaustive search. However, it is well known that not all the application parameters impact performance equally [40, 41, 62] and therefore OPTIMUSCLOUD reduces the search time by automatically selecting the most impactful parameters. Further, there exist dependencies among parameters, such as the dependency between the VM type (EC2) and Cassandra's *file-cache-size* (FCS) (Fig. 7). OPTIMUSCLOUD uses *D*-optimal design [48] to optimize the offline data collection process for training our performance model. *D*-optimal design answers this question: "*Given a budget of N data points to sample for a workload, which N points are sufficient to reveal the dependencies between configuration parameters?*". We experimentally determine that the significant dependencies in our target applications are at most pairwise and therefore we restrict the search to linear and quadratic parameters. We create a set of filters for feasible combinations of parameter values by mapping each parameter to the corresponding resource (e.g., *file-cache-size* parameter is mapped to RAM). Afterward, we check that the sum of all parameters mapped to the same resource is within that resource limit of the VM (e.g., the total size of all Cassandra buffer memories should not exceed the VM instance memory). We feed to *D*-optimal design the budget in terms of the number of data points that we can collect for offline training.

After collecting the data points determined by the *D*-optimal design, we train a random forest to act as a regressor and predict the performance of a single NoSQL server for any given set of configuration parameters, both database and VM. The average output of the different decision trees is taken as the final output. We choose random forest over other prediction models because of its easily interpretable results [50] and it has only two hyper-parameters to tune (`max_depth` and `forest_size`) compared to black-box models such as DNNs. OPTIMUSCLOUD trains a second random forest model to predict the overall cluster performance, using the predicted performance for each server, RF, CL and data-placement information. For both random forests, we use 20 trees and a maximum depth of each as 5 as that gives the best result within reasonable times.

3.3 Selection of Servers to Reconfigure

Selecting the right servers to reconfigure in a cluster is essential to achieve the best Perf/$. We introduce the notion of *Complete-Sets* to determine the right subset of servers to reconfigure. **We define a *Complete-Set* as the minimum subset of nodes for which the union of their data records covers all the records in the database at least once.**

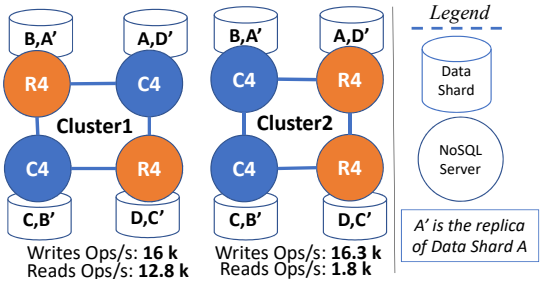

Figure 5: (RF=2, CL=1) Cluster performance depends not just on the configuration of each server, but also on the relative positions of the instances on the token ring. Cluster1 achieves 7× reads Ops/s over Cluster2 with the same VM types and sizes.

To see why the notion of *Complete-Set* is important, consider the two clusters shown in Fig. 5. Both clusters 1 and 2 use 2 C4.large and 2 R4.large and hence have the same $ cost. However, *Cluster1* achieves 7× the read Ops/s compared to *Cluster2*. The reason for the better performance of *Cluster1* is that it has one *Complete-Set* worth of servers configured to the memory-optimized R4.large architecture and therefore serves all read requests efficiently. On the other hand, *Cluster2*'s read performance suffers since all read requests to shard `B` (or its replica `B'`) have to be served by one of the C4.large servers, which has a smaller RAM and therefore serves most of the reads from desk. Accordingly, read requests to shards `B` or `B'` represent a bottleneck in *Cluster2* and cause a long queuing time for the reading threads, which brings down the performance the *entire* cluster for all the shards.

This means that all the servers within a Complete-Set must be upgraded to the faster configuration for the cluster performance to improve. Otherwise, the performance of the Complete-Set will be bounded by the slowest server in the set. OPTIMUSCLOUD partitions the cluster into one or more *Complete-Sets* using the cluster's data placement information. To identify the Complete-Sets, we collect the data placement information for each server of the cluster. OPTIMUSCLOUD queries this information either from any server (such as in Cassandra, using `nodetool ring` command) or from one of the master servers (such as in Redis, using `redis-cli cluster info` command). In Redis, identifying the Complete-Sets is easier since data tokens are divided between the master nodes only, while slaves have exact copies of their master's data. Therefore, a *Complete-Set* is formed by simply selecting a single slave node for every master node.

Maintaining Data Availability: To maintain data availability during reconfiguration of a Cassandra cluster, at least *CL* replicas of each data record must be up at any point in time. This puts an upper limit on the number of *Complete-Sets* that can be reconfigured concurrently as $Count(Complete\text{-}Sets) - CL$.

We show that the number of Complete-Sets in a cluster is *not* dependent on the number of nodes in the cluster, but is a constant factor. This is because when the cluster size increases, the range of keys assigned to every node decreases

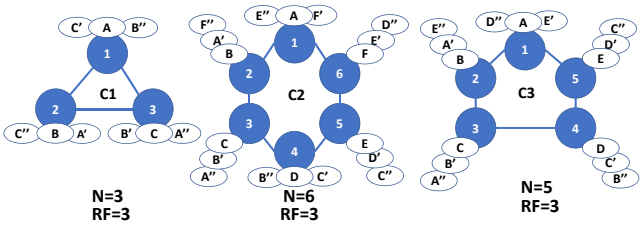

Figure 6: Replication examples with 3 different cluster sizes with RF=3. Cluster C1 has 3 nodes and each node has a complete copy of the data, therefore each node is a Complete-Set. Cluster C2 has 6 nodes and has the following Complete-Sets: [1,4] , [2,5] & [3,6]. Cluster C3 has 5 nodes (not divisible by RF=3), therefore it has two Complete-Sets: [1,3 (or 4)], [2,4 (or 5)].

and therefore the number of nodes that form a Complete-Set increases. This means that since OPTIMUSCLOUD reconfigures the instances in groups of one or more Complete-Sets concurrently, the total time to reconfigure a cluster is a constant factor independent of the cluster size. Figure 6 shows examples of *Complete-Set* for different cluster sizes

Property: OPTIMUSCLOUD **partitions the cluster into** S **Complete-Sets, and** S **is independent of the cluster size** N**.**

Proof. For a cluster of N servers with replication factor RF, there exists a total of RF copies of each record in the cluster, with no two copies of the same record stored in the same server. Assuming each node in the cluster is assigned an equal portion of the data (which NoSQL load-balancers try to achieve [35]), the size of a *Complete-Set* is $Size_{CompSet} = \lceil \frac{N}{RF} \rceil$. Consequently, the number of *Complete-Sets* in the cluster $S = \lfloor \frac{N}{Size_{compSet}} \rfloor$. If RF divides N, then the number of *Complete-Sets* is $S = \frac{N}{Size_{CompSet}} = RF$. Else, say $N\%RF = r$, then $S = \frac{RF}{1-r/N+RF/N}$, which is $\approx RF$ since in practice RF is not large, 3 being a practical upper bound. Thus, the number of *Complete-Sets* is independent of the cluster size and hence the reconfiguration time is also a constant. □

Search Space Size Reduction: Heterogeneous configurations make the search space size much larger than with homogeneous configurations. Consider a cluster of N nodes and I VM options to pick from. If we are to pick a homogeneous cloud configuration for the cluster, we have I options. However, if we are to pick a heterogeneous cloud configuration, our search space becomes I^N. If we assume balanced data placement among the servers in the cluster (as clustered NoSQL databases are designed for), the search space becomes $C(N+I-1, I-1)$ (distribute N identical balls among I boxes). However, this search space size is still too large to perform an exhaustive search to pick the optimal configurations. A cluster of size N=20 nodes and I=15 VM options gives 1.3×10^9 different configurations to select from. One may use domain-specific insights about the domain to reduce the search space for specific applications [36] or for customized distributed strategies [28]. However we aim for generalizability here.

We use one insight about Complete-Sets to reduce the search space. The nodes within each Complete-Set should be homogeneous in their configuration. Otherwise, the performance of the Complete-Set will be equal to that of the slowest node in the set. *This means that the smallest atomic unit of reconfiguration is one Complete-Set.* This insight reduces the search space, while still allowing different *Complete-Sets* to have different configurations. Thus, the search space reduces to $C(S+I-1, I-1)$=680 configurations when $S = RF = 3$. **Also note that the configuration search space is constant rather than growing with the size of the cluster.**

3.4 Selecting the Reconfiguration Plan

3.4.1 Objective Function Optimization

The objective of OPTIMUSCLOUD is to find a reconfiguration plan that maximizes Perf/$ of the cluster under a given budget and with a minimum acceptable throughput. A *reconfiguration plan* C is represented as a time series of a vector of configurations (both NoSQL and VM):

$$\mathbf{C} = \left[\{C_1, C_2, \cdots, C_M\}, \{t_1, t_2, ..., t_M\}\right] \quad (1)$$

Where M is the number of steps in the plan and timestamp t_i represents how long the configuration C_i is applied. The lookahead is $t_L = \sum_{i=1}^{M} t_i$. The optimization problem is defined as:

$$\mathbf{C}^* = _{\mathbf{C}} \frac{f(\mathbf{W}, \mathbf{C})}{Cost(\mathbf{C})} \quad (2)$$

$$\text{subject to } f(\mathbf{W}, \mathbf{C}) \geq \text{minOps } \& \; Cost(\mathbf{C}) \leq \text{Budget}$$

Here, $f(\mathbf{W}, \mathbf{C})$ is the function that maps the workload vector $\mathbf{W}$ and the configuration vector $\mathbf{C}$ to the throughput (the cluster prediction model) and $\mathbf{C}^*$ is the best reconfiguration plan selected by OPTIMUSCLOUD. The two constraints in the problem prevent us from selecting configurations that exceed the budget or those that deliver unacceptably low performance.

The optimization problem described in Equation 2 falls under the category of gradient-free optimization problems [38], in which no gradient information is available nor can any assumption be made regarding the form of the optimized function. For this category of optimization problems, several meta-heuristic search methods have been proposed, such as, Genetic Algorithms (GA) , Tabu Search [64], and Simulated Annealing. We use GA due to two relevant advantages. First, constraints can be easily included in its objective function (i.e., the fitness function). Second, it provides a good balance between exploration and exploitation through crossover and mutation [59]. We use Python `Solid` library for GA [58] and `Scikit-learn` for random forests [55].

3.4.2 Cost-Benefit Analysis

Changing either NoSQL or cloud configurations at runtime has a performance cost due to downtime caused to nodes being reconfigured. We find that most of the performance-impacting NoSQL parameters (83% for Cassandra) necessitate a server restart and naturally, changing the VM type

needs a restart as well. When a workload change is predicted in the online phase, OPTIMUSCLOUD uses its performance predictor to propose new configurations for the new workload. Afterward, OPTIMUSCLOUD estimates the reduction in performance given the expected downtime duration and compares that to the expected benefit of the new configurations. OPTIMUSCLOUD selects configurations that maximize the difference between the benefit and the cost (both in terms of Throughput/$) . This cost-benefit analysis prevents OPTIMUSCLOUD from taking greedy decisions, whenever the workload changes. Rather, it uses a long-horizon prediction of the workload over a time window to decide which reconfiguration actions to instantiate and when.

The benefit of the i^{th} step in the plan is given by:

$$B_{(i+1,i)} = \sum_{t \in t_{i+1}} f\left(W_t, C_{i+1}\right) - f\left(W_t, C_i\right) \quad (3)$$

where $f\left(W_t, C_{i+1}\right)$ is the predicted throughput using the new configuration C_{i+1}. The configuration cost is given by:

$$L_{(i+1,i)} = \sum_{p \in (C_i - C_{i+1})} t_{down} \times \delta_p \times f\left(W_t, C_i\right) \quad (4)$$

where p is any *Complete-Set* that is being reconfigured to move from configuration C_i to C_{i+1}, t_{down} is the expected downtime during this reconfiguration step, and δ_p is the portion of the cluster throughput that p contributes as estimated by our cluster predictor. We then normalize the benefit (Equation. 3) and the cost (Equation. 4) by the difference in price between configurations C_i and C_{i+1}. The value of t_{down} is measured empirically and its average value is 30 sec for NoSQL configurations and 90 sec for VM configurations.

3.5 Distinctions from Closest Prior Work

We describe the substantive conceptual differences of OPTIMUSCLOUD from two recent, related works: Selecta and SOPHIA. OPTIMUSCLOUD provides joint configuration tuning of both NoSQL and cloud VMs, while it considers heterogeneous clusters to achieve the best Perf/$. In Selecta, only heterogeneous cloud storage configurations are permissible (i.e., HDD, SSD, or NVMe). Accordingly, the configuration space in Selecta is much smaller and simpler to optimize using matrix factorization techniques. A simple extension of Selecta to our large search space produces very poor performance due to the sparsity of the generated matrix and the dependency between NoSQL and cloud configurations as we empirically show in Sec. 4.6.

In SOPHIA, only NoSQL parameters are configured and no computing platform parameters such as VM configurations are optimized. Even within NoSQL configurations, it only considers homogeneous configurations. Accordingly, SOPHIA makes a much simpler decision to either configure the complete cluster to the new configuration, or keep the old configuration—correspondingly its cost-benefit analysis is also coarse-grained, at the level of the entire cluster. For fast-changing workloads, it therefore often has to stick to the current configuration since there is not enough time to reconfigure the entire cluster (which needs to be done in a partly sequential manner to preserve data availability). *Similar to Selecta, a simple extension of SOPHIA to VM options cannot achieve the best Perf/$ for dynamic workloads, as it can only create homogeneous configurations across all phases of the workload.* We empirically show this in Sections 4.3 and 4.7.

4 Experimental Setup and Results

In this section, we evaluate OPTIMUSCLOUD under different experimental conditions for the 3 applications. We deploy OPTIMUSCLOUD and the datastore clusters (Cassandra or Redis) in Amazon EC2 in the US West (Northern California) Region. We also deploy a separate set of nodes in the same region to serve as workload generators (i.e., shooters). We vary the number of shooting threads in runtime to simulate the changes in the request rate in the workload trace. The results are averages of 20 runs, with each run using a different subset of the training data. Our evaluation answers four broad questions. (1) How does OPTIMUSCLOUD compare in terms of Perf/$ and P99 latency with three state-of-the-art systems (which can only create homogeneous configurations) and two oracle-based baselines? (2) What is the accuracy of each module of OPTIMUSCLOUD, such as, the workload and the performance predictors? (3) How do application requirements such as RF and CL impact OPTIMUSCLOUD? (4) How does OPTIMUSCLOUD generalize to different applications (we use three), databases (Cassandra and Redis), and levels of prediction errors?

Major Insights: We draw several key insights from our evaluation. **First**, the flexibility afforded by being able to reconfigure different parts of the cluster to different configurations is useful—all three prior protocols being compared (and in fact, all works to date) can only create homogeneous configurations. *Further, the proactive approach of initiating reconfiguration upon predicted workload change helps to handle dynamic workloads and keeps latency low (CherryPick and Selecta are both reactive).* **Second**, OPTIMUSCLOUD's design reduces the heterogeneous configurations search space significantly. Accordingly, it is able to search efficiently and finds higher performing VM and NoSQL configurations than cluster configurations selected by CherryPick, Selecta, or SOPHIA. This improvement persists across applications (highest for the more predictable HPC analytics workload and lowest for the MG-RAST workload) (Fig. 8, 10, 11), different (RF,CL) values (larger values of RF and smaller values of CL) (Fig. 10), and data volumes (benefit stays unchanged) (Fig. 8). **Third**, OPTIMUSCLOUD achieves comparable or better P99 latency than the baselines, thus showing that it does not sacrifice raw performance in search of the performance per unit cost.

MG-RAST			BUS-Tracking		
MC-Order	Lookahead	RMSE	MC-Order	Lookahead	RMSE
First	5m	43.7%	First	15m	6.9%
First	10m	68.7%	**First**	**1h**	**7.4%**
Second	**5m**	**43.4%**	Second	5m	7.12%
Second	10m	68.2%	Second	1h	7.4%

Table 2: (MC stands for Markov Chain). Workload prediction RMSE for MG-RAST and Bus-tracking workloads with different lookahead periods.

4.1 Applications

Here we give the details for our three use case applications, which together span a wide range in terms of predictability and nature of the requests in the workload. **MG-RAST** is a global-scale metagenomics portal [7], the largest of its kind, which allows many users to simultaneously upload their metagenomic data to the repository, apply a pipeline of computationally intensive processes and optionally commit the results back to the repository for shared use. Its workload does not have any discernible daily or weekly pattern, as the requests come from all across the globe and we find that the workload can change drastically over a few minutes. A total of 80 days of real query trace were analyzed, 60 days for training and 20 days for testing. In production, MG-RAST is executed with the values RF=3, CL=1 and it shows abrupt switches in the Read-Ratio (typically from RR=0 to RR=1) and vice versa. The frequency of these switches are 430/day on median. This presents a challenging use case as only 5 minutes of accurate lookahead is possible.

The second workload is the **Bus-Tracking** application [39] where read, scan, insert, and update operations are submitted to a backend database. The data has strong daily and weekly patterns to it. This workload has less frequent switches of 60/day on median. For this workload, 60 days of real query trace were analyzed for the application (40 for training and 20 for testing). The relative proportions of the different kinds of queries are 42.2% updates, 54.8% scans, 2.82% inserts, and 0.18% reads. As shown in Table 2, the prediction accuracy for Bus-Tracking workload is much better compared to the MG-RAST workload, as expected due to the more regular patterns, and here we use a longer lookahead period of 1 hour.

The third use case is a queue of **data analytics jobs** such as would be submitted to an HPC computing cluster. Here the workload can be predicted over long time horizons (order of an hour) by observing the jobs in the queue and leveraging the fact that a significant fraction of the job patterns are recurring. Thus, our evaluation cases span the range of patterns and corresponding predictability of the workloads. We simulate a shared queue of batch data analytics jobs. We modeled the jobs on data analytics requests submitted to a real Microsoft Cosmos cluster [26]. Each job is divided into stages and the workload characteristics of the job change with every stage. The job size is a random variable $\sim$ U(200,100K) operations. The workload switches are 780/day on median, with a level of concurrency of 10 jobs. We achieve accurate prediction over a lookahead duration of 1 hour and we use that for our setting with this use case.

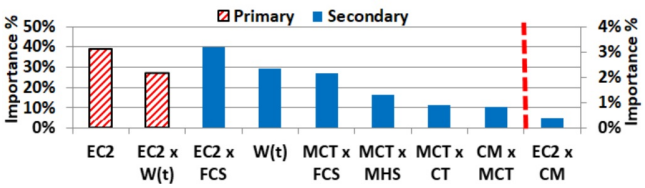

Figure 7: Importance of various parameters, including pairwise combinations. Parameters with black solid bars are w.r.t. the right Y-axis. EC2 configuration, the workload, and top 5 Cassandra parameters describe 81% of data variance, after which there is a significant drop in importance, denoted by the red dotted line. Top parameters are: file_cache_size (FCS), memtable_cleanup_threshold (MCT), memtable_heap_space (MHS), compaction_throughput (CT), and compaction_method (CM)

4.2 Baselines

We compare OPTIMUSCLOUD to the following baselines:

1. *Homogeneous-Static*: We use our cluster predictor to select the single best configuration to use for the entire duration of the predicted workload. The entire workload is assumed to be known in advance, making this an impractically optimistic baseline. Nevertheless, it is a measure of how well a *statically determined* homogeneous configuration can perform when powered by a hypothetically perfect workload predictor.
2. *CherryPick*: We use CherryPick's Bayesian Optimization (BO) to find a heterogeneous cloud configuration which maximize our objective metric. When the workload changes, BO collects 20 points and selects the best cloud configuration. This process takes about 3 minutes with parallelization on a 16-core machine. The reconfiguration is done by restarting servers all at once, thus making data unavailable transiently.
3. *Selecta*: We use Selecta's SVD prediction model to select the optimized homogeneous configuration with workload changes. We populate the SVD matrix with the same training data as used for training of OPTIMUSCLOUD. Further, we give a benefit to Selecta that all the workload characteristics are assumed to be pre-filled in the matrix (or close to it) so that it does not have to execute and profile the workload that arrives at runtime, but can be looked up in the matrix. Both CherryPick and Selecta run reactively with workload changes and neither can change the application configuration. They operate in a greedy manner initiating reconfiguration whenever the workload changes, unlike OPTIMUSCLOUD that optimizes for the workload over a lookahead time window.
4. *SOPHIA*: We use SOPHIA for homogeneous NoSQL configurations while the VM configurations are fixed to the recommended VM types in Cassandra's and Redis' documentations i.e., Compute-Optimize C4.large for Cassandra [2] and Memory-Optimized R4.large for Redis [1].
5. *Theoretical-Best*: This is a baseline that knows what is the best-performing configuration for every workload. It then switches the cluster to this configuration without any downtime cost. Though impractical, this baseline provides a quantitative upper bound for any protocol and is used for normalization of our results.

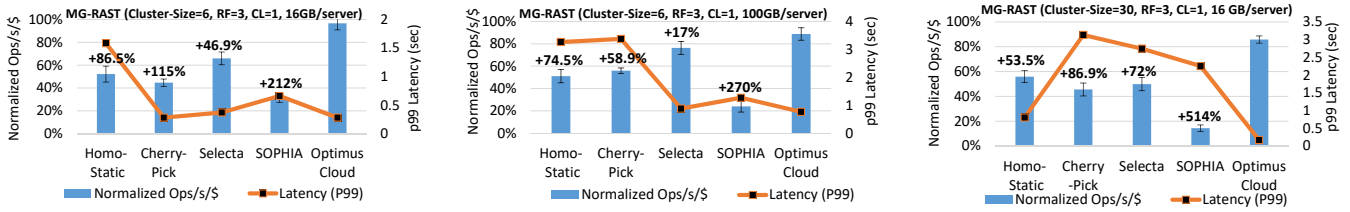

Figure 8: Evaluation of MG-RAST traces in Cassandra using* OPTIMUSCLOUD *vs state-of-the-art tuning systems. The primary Y-axis represents the ratio of the normalized Ops/s/$ achieved by each system to the theoretical-best performance.* OPTIMUSCLOUD *achieves the highest Perf/$ and lowest P99 latency.

4.3 End-to-end System Evaluation

We evaluate how effective OPTIMUSCLOUD and each of the baselines are in selecting the best reconfiguration plan. In Fig. 8 we show the evaluation for the MG-RAST application, the most challenging one for us due to its unpredictable workload characteristics. We use a 1 hour trace from MG-RAST and apply it to a cluster of 6 or 30 servers We show the performance of the different plans with data volume per server of 16GB and 100GB. The performance of each solution is normalized by that of the Theoretical-Best. OPTIMUSCLOUD's plan achieves the highest Ops/s/$ and the lowest latency over all baselines. OPTIMUSCLOUD achieves 86% and 74% improvement over Homogeneous-Static for the 16GB and 100GB cases respectively. This shows there is no single static configuration that can achieve the optimal performance for all phases of the workload. Compared to CherryPick and Selecta, OPTIMUSCLOUD achieves 87% and 45% improvement on average. This is because both systems are striving to create a homogeneous configuration to meet the performance requirement and end up increasing the cost. Further, CherryPick incurs the delay of performing the Bayesian optimization, which takes 3 minutes on average and causes the cluster to operate with sub-optimal configurations during this long delay. The aggressive reconfiguration of CherryPick and Selecta (shutting down all servers and restarting all together) causes a significant performance hit (throughput of zero) for the cluster. Compared to SOPHIA, OPTIMUSCLOUD achieves 212% and 270% improvement in Perf/$. This highlights the benefit of jointly tuning VM and database configurations.

We draw several other conclusions. First, with increasing data volume, the throughput of all protocols goes down because what would once fit in memory (R4.large has 16 GB) now has to go to disk. But the effect on Selecta and CherryPick is smaller because they use expensive and well-resourced memory VMs. Consequently, the performance benefit of OPTIMUSCLOUD decreases. Second, Selecta is achieving better performance than CherryPick, which is consistent with the results reported in [32]. This is because of the longer response time of CherryPick's Bayesian Optimization versus Selecta's matrix lookup. In terms of latency, OPTIMUSCLOUD scales well (comparing the N=6 to N=30), while Selecta and CherryPick both suffer (latency increases of 7.4× and 11×). This is because the control message traffic is very large in these two baselines as they aggressively shut down and restart all the servers together to achieve a reconfiguration, causing the scalability bottleneck. We also notice that SOPHIA scalability is better than the other baselines as it performs sequential reconfiguration. OPTIMUSCLOUD achieves as low tail-latency as Selecta and CherryPick for small scales, and lower at larger scale due to the reason above. Homogeneous-Static has a high latency for all cases due to its inability to adapt to dynamic workloads. The comparison with Selecta and CherryPick shows how important it is to apply *online* reconfiguration to minimize tail-latencies for fast changing workloads.

4.4 Sensitive Parameter Identification

We test the feasibility of pruning the joint configuration search space (i.e., VM and NoSQL), while maintaining the dependencies among the configuration parameters. We collect a total of 3K data points equally from 15 different VM types. We use D-optimal design to decide which data points to collect for building the performance prediction model. Fig. 7 shows the importance of the most impactful parameters, either singly or pairwise, as determined from the regression model. The instance architecture (EC2) and the workload (W(t)) are the most impactful, followed by top-5 database configuration parameters. Note that the costs of changing different configuration parameters are different as it takes about 90 sec to change EC2 type, whereas changing Cassandra's configuration only requires around 30 sec with no impact on the cluster's $ cost. Expectedly, the instance architecture has high inter-dependency with the NoSQL parameters because the architecture controls the physical resources available to the DBMS.

4.5 Single Server Performance Prediction

We evaluate three possible single server prediction models for inclusion in OPTIMUSCLOUD. In each case, we use a Random Forest using 75%:25% for training and prediction.
1. *N-Solitary-Models*: This builds a separate prediction model per architecture. It predicts the performance of a given architecture/configuration combination using previously collected data points from the same architecture.
2. *Combined-Categorical*: This builds a combined model using all points from all architectures, while it represents the architecture as a categorical parameter (with integral values).

Workload	MG-RAST		BUS		HPC	
Metric	R^2	*RMSE*	R^2	*RMSE*	R^2	*RMSE*
N-Solitary -Models	0.2	3401.4	0.127	109.9	0.04	2778
Selecta	-0.14	4149.3	0.66	110.6	0.932	2451
OPTIMUS -Categorical	0.41	1334.2	0.986	21.87	0.983	1172.9
OPTIMUS -Numerical	**0.89**	**1260.9**	**0.988**	**19.77**	**0.986**	**1076.2**

Table 3: Comparison of different Single-server prediction techniques. OPTIMUSCLOUD ***achieves better performance in terms of*** R^2 ***and RMSE over all baselines.***

Thus, knowledge transfer is limited across architectures.

3. *Combined-Numerical*: This also builds a combined model for all architectures. However, it describes the architecture in terms of its resources e.g., C4.large is represented as vCPU 8, RAM 3.75 GB, Network-Bandwidth 0.62 Gbits/s. Thus this allows extrapolating model knowledge across architectures. We test the accuracy of each predictor using the same number of data points (100 points per architecture) and show the result in terms of R^2 (Table 3). We see that using a separate model per architecture gives very poor performance due to the lack of knowledge transfer between architectures. Moreover, the numerical representation shows a significant improvement in prediction performance over the categorical representation due to better knowledge transfer across architectures. Thus, we use the *Combined-Numerical* model in OPTIMUSCLOUD.

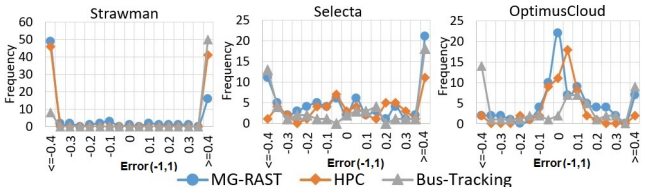

Figure 9: Performance prediction error histogram for heterogeneous clusters. We notice that OPTIMUSCLOUD***'s error percentage is within -15% to +15% for 70% of the test points, with*** R^2 ***value of 0.91 and RMSE of 7.7 KOps/s. On the other hand, the best strawman shows poor performance (RMSE 36 KOps/s) while Selecta performs better (RMSE 21 KOps/s).***

4.6 Cluster Performance Prediction

We evaluate the accuracy of our cluster performance prediction model. We use a cluster of 6 nodes with RF=3, CL=1 and investigate the impact of changing the EC2 architecture of each *Complete-Set*. Thus, the cluster is partitioned into 3 *Complete-Set*s. We use 3 families of EC2's 4th generation (C4, R4, M4) and three different sizes of each family (large, xlarge, 2xlarge). We collect 330 data points covering all combinations of assigning instance types to these 3 *Complete-Set*s.

In Fig. 9, we compare our model with a strawman predictor that uses the sum of Ops/s for each individual server as the overall cluster performance (we also tested Average, Min, and Max and got worse performance). This strawman achieves poor prediction performance with a low R^2 value of 0.08 and an unacceptably high RMSE of 36 KOps/s. We also compare our model with the latent factor collaborative filtering technique, SVD, used in Selecta [32], which we reimplement using the sci-kit `surprise` library [30]. Selecta achieved better R^2 value of 0.69 and a lower RMSE of 21 KOps/s compared to the strawman predictor. However, our model achieves better performance due to the fact that our Random-Forest model can use non-linear combinations of the elementary features (up to quadratic), while SVD is confined to using linear combinations only. The shapes of the error curves are also different—the Selecta and the strawman curves are bathtub-shaped indicating significant overestimation or underestimation, while the OPTIMUSCLOUD curve is bell-shaped with a mean close to zero. The bathtub curves are due to the fact that these protocols are ignorant of the token assignment of the cluster and consider erroneously that each node's throughput has the same contribution to the cluster throughput.

4.7 Evaluation with Diverse Workloads

Now we evaluate the performance for different workloads, cluster scales, and (RF, CL) requirements.

HPC Workload: Figure 10 shows the improvement of OPTIMUSCLOUD over the baselines for the HPC data analytics traces. We first change RF from 1 to 3, holding CL at 1. Then we change CL from 1 to quorum, which is 2, holding RF at 3. OPTIMUSCLOUD's plan achieves the highest Perf/$ and the lowest latency over all baselines for all setups. At RF=3, OPTIMUSCLOUD achieves 20% and 24% better Perf/$ over Homogeneous-Static configuration for CL=1 and CL=Q respectively. This again shows the importance of dynamic reconfiguration to handle workloads with changing characteristics. In comparison to CherryPick, OPTIMUSCLOUD achieves 143% and 89% better performance for CL=1 and CL=Q respectively and 130% and 80% over Selecta. Compared to SOPHIA, OPTIMUSCLOUD achieves 23% and 12.5% better Perf/$. Notice that when RF=1 and CL=1, OPTIMUSCLOUD can no longer perform non-atomic configurations since the *Complete-Set* in this case is the entire cluster. Thus, its improvement over Homogeneous-Static decreases to 9%.

As RF goes up, the amount of data on each node goes up bringing down the absolute performance. Homogeneous-Static is affected more than OPTIMUSCLOUD and therefore the benefit of OPTIMUSCLOUD increases. CherryPick and Selecta are relatively unaffected by increasing RF as they had low throughputs to begin with and they reconfigure all the nodes at once, irrespective of RF and CL. SOPHIA benefits from increasing RF since it can reconfigure more servers concurrently without degrading data availability. As CL goes up, again CherryPick and Selecta are relatively unaffected. The performance of OPTIMUSCLOUD goes down because the maximum number of *Complete-Sets* that OPTIMUSCLOUD can reconfigure at a time is inversely proportional to CL. Thus, its benefit over CherryPick and Selecta reduces. However, we note that for most of our target environments, CL is unlikely

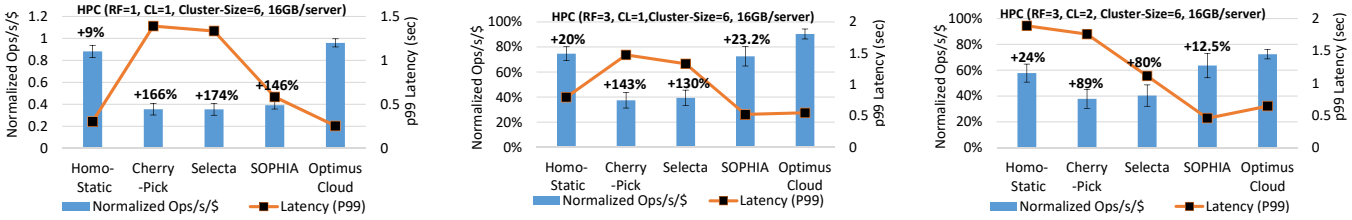

Figure 10: HPC workload evaluation with 10 concurrent jobs, and varying (RF,CL) requirements

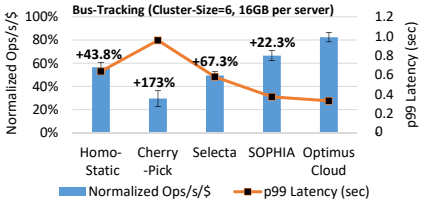

Figure 11: Bus-Tracking workload pattern with RF=3, CL=1

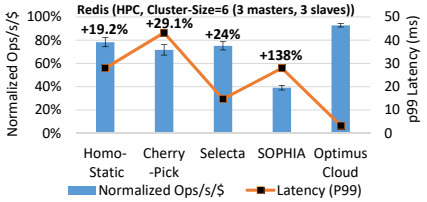

Figure 12: Evaluation on Redis for HPC workload with a cluster of 6 servers

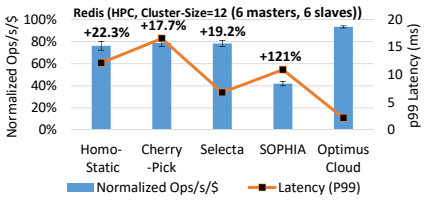

Figure 13: Evaluation on Redis for HPC workload with a cluster of 12 servers

to be higher than 1 as deployments often favor availability and latency over consistency. In terms of latency, OPTIMUSCLOUD achieves the lowest P99 latency across all setups. We also notice that SOPHIA has lower latency than other baselines as it performs a gradual reconfiguration of the different server instances to maintain data availability.

Bus-Tracking Workload: This workload has strong weekly and daily patterns, which allows the workload predictor to provide accurate predictions for long lookahead periods. For a 1 day-trace, the result is shown in Fig. 11. OPTIMUSCLOUD achieves better performance/$ over Homogeneous-Static, CherryPick, Selecta, and SOPHIA by 46%, 178%, 67%, and 28% respectively. As before, OPTIMUSCLOUD achieves the lowest tail latency across all techniques. The tail latency metric is very important for this workload as it represents a user-facing application. We observe that OPTIMUSCLOUD achieves higher gains in performance over CherryPick and Selecta compared to the MG-RAST workload, which shows the benefit of longer accurate predictions for OPTIMUSCLOUD.

4.8 Evaluation with Redis

Redis has a very different architecture than Cassandra and is therefore a suitable target to evaluate the generalizability of OPTIMUSCLOUD. Here we use Redis in clustered mode as a distributed cache (a common use case for Redis)—if the key is found in Redis' memory, it is served by Redis, else, it is served by a slower disk-based database. We apply the HPC analytics workload to a cluster of 6 or 12 Redis servers, keeping the replication degree as 2. We select HPC workload as it has the shortest key-reuse-distance between the three workloads and for which using Redis as a cache is most beneficial [10, 21]. We tune Redis' `maxmemory-policy` and `maxmemory` parameters and observe that changing the cloud configurations for more or less RAM size has an impact on the best value of both parameters. We also found that Redis' Perf/$ is sensitive to workload parameters such as job size, access distribution, and read-to-write. We start by collecting 108 data points for different jobs with varying job sizes, access distributions, and read-write ratios. Job size is a random variable with U(0.5M,1.5M) operations. Access distribution is randomly selected from *Uniform*, *Latest*, and *Zipfian*. Read-write ratio is a random variable with distribution U(0,1). We experiment with traces of 75 jobs which span a total of 5 hours. Our performance predictor achieves an R^2 of 0.922 averaged over 20 runs. From Fig. 12, we see that OPTIMUSCLOUD achieves a better Perf/$ of 19% (Homogeneous-Static), 29% (CherryPick), 24% (Selecta), and 138% (SOPHIA). Also, OPTIMUSCLOUD reduces the tail latency by 8.4X (Homogeneous-Static), 13X (CherryPick), 4.4X (Selecta), and 8.1X (SOPHIA).

We draw the following insights. First, as SOPHIA uses an expensive cluster of R4.large (as recommended in Redis' documentation [1]), it achieves a very low Perf/$. Second, both CherryPick and Selecta switch from R4.large to the less expensive C4.large and M4.large when the workload changes (and less memory is required) and therefore achieve a higher Perf/$. Finally, by using a heterogeneous cluster of the three VM types as well as jointly tuning the application configuration, we achieve the best Perf/$ and the lowest latency among all techniques. To test scalability, we increase the number of servers to 12 (Fig. 13) and note that the normalized performance of each system stays approximately constant.

4.9 Tolerance to Prediction Errors

We investigate how tolerant OPTIMUSCLOUD is to errors in both predictors—performance (throughput) and workload. We add synthetic noise to the output of each predictor separately and then show how does the benefit of OPTIMUSCLOUD change with the amount of synthetic noise for the HPC workload. The percentage of noise is represented as a uniform random variable that is added to (or subtracted from) the output of the predictor. For performance prediction,

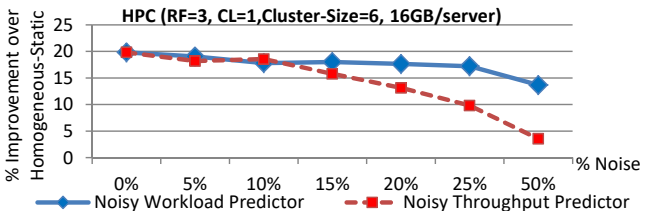

Figure 14: Impact of noisy predictions on* OPTIMUSCLOUD*'s improvement over best Homogeneous-Static configurations

the noise is added to the overall throughput/$ predicted by our multi-server model. For workload prediction, the noise is added to the number of requests/sec in addition to the workload change duration. As shown in Fig.14, OPTIMUSCLOUD's improvement over Homogeneous-Static decreases with increasing levels of noise, as the selected configurations deviate from the best configurations. We note that OPTIMUSCLOUD is more sensitive to errors in the throughput predictor compared to errors in the workload predictor, which is demonstrated in the steeper downward slope in the noisy throughput predictor curve. The reason for this high sensitivity is that OPTIMUSCLOUD uses the throughput predictor to select the best configuration and with increasing levels of noise, the selected configuration more frequently deviates from the optimal. As discussed earlier, a slight deviation from the optimal configuration may cause a significant reduction in Perf/$. On the other hand, slight errors in workload prediction causes OPTIMUSCLOUD to reconfigure earlier or later than it optimally should. However, this has less impact on performance as long as it still switches to the best configuration.

5 Related Work

Configuration tuning for datastores: Several works [6, 31] target online configuration tuning for replicated or geo-replicated datastores. [27, 56] perform online reconfiguration for NoSQL datastores. None of these works address how to optimize for cost-performance benefits by exploiting different cloud VM/instance types. A large body of work focused on the best logical or physical design for static workloads in DBMS [3,12,13,18,29,51,52,61], which are orthogonal to our work. OtterTune [62], BerkeleyDB [60], and iTuned [24] only optimize DBMS configuration, while OPTIMUSCLOUD optimizes both NoSQL configuration and the cluster on which it runs. Pocket [33] optimizes the storage servers for ephemeral data analytics jobs in contrast to handling long-running jobs that are OPTIMUSCLOUD's focus. While OPTIMUSCLOUD can also optimize storage, we choose to restrict our storage servers to elastic block storage (EBS) that are separate from the VMs and thus retain data durability.

Performance predictions: Ernest [63] predicts performance of data analytics applications through system modeling. DBSeer [46] uses linear models to predict resource utilization (e.g. Disk I/O). Myria [49, 66] gives personalized SLAs by predicting query execution times for specific workloads. Recent works [42, 43, 45] improve utilization by *learning* workload characteristics. [44] handles prediction errors for unseen workloads. [23] uses queuing models. Rafiki [40] uses a surrogate model to predict performance of NoSQL datastores. No prior work predicts performance for heterogeneous clusters.

Cache Hit-Rate Maximization: Several works target maximizing cache hit-rates and improving end-to-end latency, either for a single application [14] or multi-tenant deployments [15]. However, neither VM configurations nor heterogeneous cluster configurations are considered to optimize performance/$. [9, 65] propose new cache eviction policies that can be implemented in Redis and then selected by OPTIMUSCLOUD for the appropriate workloads.

6 Discussion

Impact on consistency: OPTIMUSCLOUD exploits the fact that in typical NoSQL deployments, RF>CL as availability and low-latency are favored over consistency [22, 47, 57]. The higher the difference between RF and CL, the smaller the subset of servers that needs reconfiguration, therefore the higher the gain of OPTIMUSCLOUD over baselines (Figure 10). For users whose primary goal is consistency and want to use a high value for CL, one option is to also increase RF and leverage the high gain of OPTIMUSCLOUD. However, increasing RF also increases the total number of copies stored in the cluster, which might negatively impact the cluster's write performance in exchange for higher availability.

Compatibility with other key-value stores: OPTIMUSCLOUD's design assumes that the underlying key-value store implements a protocol to identify and select the fastest replica(s) given a new query. Cassandra achieves this using its dynamic snitching policy, while other popular systems have similar protocols (e.g., Elasticsearch achieves this by its Adaptive Replica Selection policy [25]). If this feature is not implemented in the system, a simple solution is for OPTIMUSCLOUD to provide the system with the ordered list of replicas, using OPTIMUSCLOUD 's performance predictor.

7 Conclusion

For cost-optimal performance of a distributed NoSQL cloud database, it is critical to jointly tune NoSQL and cloud configurations. OPTIMUSCLOUD provides the insight that it is optimal to create heterogeneous configurations and for this, it determines at runtime the *minimum number of servers* to reconfigure. Using a novel concept of Complete Sets, OPTIMUSCLOUD provides a technique to search through the large search space brought out by heterogeneity. Configurations found by OPTIMUSCLOUD outperform those by prior works, CherryPick, Selecta, and SOPHIA, in both Perf/$ and tail latency, across two NoSQL DBMSs, Cassandra and Redis, and all experimental conditions.

References

[1] 5 tips for running redis over aws. https://redislabs.com/blog/5-tips-for-running-redis-over-aws/. [Online; accessed 17-September-2019].

[2] Best practices for running apache cassandra on amazon ec2. https://aws.amazon.com/blogs/big-data/best-practices-for-running-apache-cassandra\-on-amazon-ec2/. [Online; accessed 17-September-2019].

[3] AGRAWAL, S., NARASAYYA, V., AND YANG, B. Integrating vertical and horizontal partitioning into automated physical database design. In *ACM SIGMOD international conference on Management of data* (2004).

[4] AKDERE, M., ÇETINTEMEL, U., RIONDATO, M., UPFAL, E., AND ZDONIK, S. B. Learning-based query performance modeling and prediction. In *2012 IEEE 28th International Conference on Data Engineering* (2012), IEEE, pp. 390–401.

[5] ALIPOURFARD, O., LIU, H. H., CHEN, J., VENKATARAMAN, S., YU, M., AND ZHANG, M. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation NSDI'17* (2017), pp. 469–482.

[6] ARDEKANI, M. S., AND TERRY, D. B. A self-configurable geo-replicated cloud storage system. In *OSDI* (2014), pp. 367–381.

[7] ARGONNE NATIONAL LABORATORY. MG-RAST: Metagenomics Analysis Server. urlhttps://www.mg-rast.org/, 2019.

[8] AWS. Amazon EC2. https://aws.amazon.com/ec2/pricing/on-demand/, May 2018.

[9] BECKMANN, N., CHEN, H., AND CIDON, A. {LHD}: Improving cache hit rate by maximizing hit density. In *15th USENIX Symposium on Networked Systems Design and Implementation NSDI'18* (2018), pp. 389–403.

[10] BEYLS, K., AND D'HOLLANDER, E. Reuse distance as a metric for cache behavior. In *Proceedings of the IASTED Conference on Parallel and Distributed Computing and systems* (2001), vol. 14, pp. 350–360.

[11] CHATERJI, S., KOO, J., LI, N., MEYER, F., GRAMA, A., AND BAGCHI, S. Federation in genomics pipelines: techniques and challenges. *Briefings in bioinformatics 20*, 1 (2019), 235–244.

[12] CHAUDHURI, S., AND NARASAYYA, V. Self-tuning database systems: a decade of progress. In *Proceedings of the 33rd international conference on Very large data bases* (2007), VLDB Endowment, pp. 3–14.

[13] CHAUDHURI, S., AND NARASAYYA, V. R. An efficient, cost-driven index selection tool for microsoft sql server. In *VLDB* (1997).

[14] CIDON, A., EISENMAN, A., ALIZADEH, M., AND KATTI, S. Cliffhanger: Scaling performance cliffs in web memory caches. In *13th USENIX Symposium on Networked Systems Design and Implementation NSDI'16* (2016), pp. 379–392.

[15] CIDON, A., RUSHTON, D., RUMBLE, S. M., AND STUTSMAN, R. Memshare: a dynamic multi-tenant key-value cache. In *2017 USENIX Annual Technical Conference ATC'17* (2017), pp. 321–334.

[16] CLUTCH. Data replication in cassandra. https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archDataDistributeReplication.html, 2019.

[17] COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 143–154.

[18] CURINO, C., JONES, E., ZHANG, Y., AND MADDEN, S. Schism: a workload-driven approach to database replication and partitioning. *VLDB Endowment* (2010).

[19] DATASTAX. Cassandra dynamic snitching. https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archSnitchDynamic.html, 2019.

[20] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review* (2007), vol. 41, ACM, pp. 205–220.

[21] DING, C., AND ZHONG, Y. Predicting whole-program locality through reuse distance analysis. In *ACM Sigplan Notices* (2003), vol. 38, ACM, pp. 245–257.

[22] DIOGO, M., CABRAL, B., AND BERNARDINO, J. Consistency models of nosql databases. *Future Internet 11*, 2 (2019), 43.

[23] DIPIETRO, S., CASALE, G., AND SERAZZI, G. A queueing network model for performance prediction of apache cassandra. In *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools* (2017), pp. 186–193.

[24] DUAN, S., THUMMALA, V., AND BABU, S. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment 2*, 1 (2009), 1246–1257.

[25] ELASTICSEARCH. Improving Response Latency in Elasticsearch with Adaptive Replica Selection. urlhttps://www.elastic.co/blog/improving-response-latency-in-elasticsearch-with-adaptive-replica-selection, 2020.

[26] FERGUSON, A. D., BODIK, P., KANDULA, S., BOUTIN, E., AND FONSECA, R. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings*

of the 7th ACM european conference on Computer Systems (2012), ACM, pp. 99–112.

[27] GHOSH, M., WANG, W., HOLLA, G., AND GUPTA, I. Morphus: Supporting online reconfigurations in sharded nosql systems. *IEEE Transactions on Emerging Topics in Computing* (2015).

[28] GHOSHAL, A., GRAMA, A., BAGCHI, S., AND CHATERJI, S. An ensemble svm model for the accurate prediction of non-canonical microrna targets. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics* (2015), pp. 403–412.

[29] GUPTA, H., HARINARAYAN, V., RAJARAMAN, A., AND ULLMAN, J. D. Index selection for olap. In *IEEE International Conference on Data Engineering (ICDE)* (1997).

[30] HUG, N. Surprise, a python library for recommender systems. `http://surpriselib.com`, 2017.

[31] KEMME, B., BARTOLI, A., AND BABAOGLU, O. Online reconfiguration in replicated databases based on group communication. In *Dependable Systems and Network (DSN)* (2001), IEEE, pp. 117–126.

[32] KLIMOVIC, A., LITZ, H., AND KOZYRAKIS, C. Selecta: heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference ATC'18* (2018), pp. 759–773.

[33] KLIMOVIC, A., WANG, Y., STUEDI, P., TRIVEDI, A., PFEFFERLE, J., AND KOZYRAKIS, C. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2018), pp. 427–444.

[34] KLOPHAUS, R. Riak core: Building distributed applications without shared state. In *ACM SIGPLAN Commercial Users of Functional Programming* (2010), ACM, p. 14.

[35] KONSTANTINOU, I., TSOUMAKOS, D., MYTILINIS, I., AND KOZIRIS, N. Dbalancer: distributed load balancing for nosql data-stores. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), ACM, pp. 1037–1040.

[36] KOO, J., ZHANG, J., AND CHATERJI, S. Tiresias: Context-sensitive approach to decipher the presence and strength of microrna regulatory interactions. *Theranostics 8*, 1 (2018), 277.

[37] KOUSIOURIS, G., CUCINOTTA, T., AND VARVARIGOU, T. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *Journal of Systems and Software 84*, 8 (2011), 1270–1291.

[38] KOZIEL, S., AND YANG, X.-S. *Computational optimization, methods and algorithms*, vol. 356. Springer, 2011.

[39] MA, L., VAN AKEN, D., HEFNY, A., MEZERHANE, G., PAVLO, A., AND GORDON, G. J. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data* (2018), ACM, pp. 631–645.

[40] MAHGOUB, A., WOOD, P., GANESH, S., MITRA, S., GERLACH, W., HARRISON, T., MEYER, F., GRAMA, A., BAGCHI, S., AND CHATERJI, S. Rafiki: A middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (2017), ACM, pp. 28–40.

[41] MAHGOUB, A., WOOD, P., MEDOFF, A., MITRA, S., MEYER, F., CHATERJI, S., AND BAGCHI, S. SOPHIA: Online reconfiguration of Clustered NoSQL Databases for Time-Varying Workloads. In *2019 USENIX Annual Technical Conference ATC'19* (2019), Usenix, pp. 223–240.

[42] MAO, H., ALIZADEH, M., MENACHE, I., AND KANDULA, S. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (2016), pp. 50–56.

[43] MAO, H., SCHWARZKOPF, M., VENKATAKRISHNAN, S. B., MENG, Z., AND ALIZADEH, M. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 270–288.

[44] MITRA, S., BRONEVETSKY, G., JAVAGAL, S., AND BAGCHI, S. Dealing with the unknown: Resilience to prediction errors. In *2015 International Conference on Parallel Architecture and Compilation (PACT)* (2015), IEEE, pp. 331–342.

[45] MITRA, S., MONDAL, S. S., SHEORAN, N., DHAKE, N., NEHRA, R., AND SIMHA, R. Deepplace: Learning to place applications in multi-tenant clusters. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems* (2019), pp. 61–68.

[46] MOZAFARI, B., CURINO, C., AND MADDEN, S. Dbseer: Resource and performance prediction for building a next generation database cloud. In *CIDR* (2013).

[47] NEGRIN, R. Thank you for your help nosql, but we got it from here. https://www.memsql.com/blog/why-nosql-databases-wrong-tool-for-modern-application/, 2018.

[48] OF STANDARDS, N. I., AND (NIST), T. D-optimal designs. `https://itl.nist.gov/div898/handbook/pri/section5/pri521.htm`, 2019.

[49] ORTIZ, J., DE ALMEIDA, V. T., AND BALAZINSKA, M. Changing the face of database cloud services with personalized service level agreements. In *CIDR* (2015).

[50] PALCZEWSKA, A., PALCZEWSKI, J., ROBINSON, R. M., AND NEAGU, D. Interpreting random forest classification models using a feature contribution

method. In *Integration of reusable systems*. Springer, 2014, pp. 193–218.

[51] PAVLO, A., JONES, E. P., AND ZDONIK, S. On predictive modeling for optimizing transaction execution in parallel oltp systems. *VLDB Endowment* (2011).

[52] RAO, J., ZHANG, C., MEGIDDO, N., AND LOHMAN, G. Automating physical database design in a parallel database. In *ACM SIGMOD international conference on Management of data* (2002).

[53] REDIS. Redis cluster tutorial. `https://redis.io/topics/cluster-tutoriallpp`, February 2015.

[54] REDIS. Using redis as an lru cache. `https://redis.io/topics/lru-cache`, 2019.

[55] SCIKITLEARN. scikit-learn: Machine learning in python. `https://scikit-learn.org/stable/`, 2019.

[56] SHIN, Y., GHOSH, M., AND GUPTA, I. Parqua: Online reconfigurations in virtual ring-based nosql systems. In *2015 International Conference on Cloud and Autonomic Computing* (2015), IEEE, pp. 220–223.

[57] SINGH, V. K. Sql vs nosql databases. https://medium.com/system-design-blog/sql-vs-nosql-databases-6896a8cb1800, 2019.

[58] SOLID. Solid:a comprehensive gradient-free optimization framework written in python. `https://github.com/100/Solid`, 2019.

[59] SRINIVAS, M., AND PATNAIK, L. M. Genetic algorithms: A survey. *computer 27*, 6 (1994), 17–26.

[60] SULLIVAN, D. G., SELTZER, M. I., AND PFEFFER, A. *Using probabilistic reasoning to automate software tuning*, vol. 32. ACM, 2004.

[61] VALENTIN, G., ZULIANI, M., ZILIO, D. C., LOHMAN, G., AND SKELLEY, A. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *IEEE International Conference on Data Engineering (ICDE)* (2000).

[62] VAN AKEN, D., PAVLO, A., GORDON, G. J., AND ZHANG, B. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (2017), ACM, pp. 1009–1024.

[63] VENKATARAMAN, S., YANG, Z., FRANKLIN, M., RECHT, B., AND STOICA, I. Ernest: efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation NSDI'16* (2016), pp. 363–378.

[64] W, G. F., AND A, K. G. *Handbook of metaheuristics*, vol. 57. Springer Science & Business Media, 2006.

[65] WALDSPURGER, C., SAEMUNDSSON, T., AHMAD, I., AND PARK, N. Cache modeling and optimization using miniature simulations. In *2017 USENIX Annual Technical Conference ATC'17* (2017), pp. 487–498.

[66] WANG, J., BAKER, T., BALAZINSKA, M., HALPERIN, D., HAYNES, B., HOWE, B., HUTCHISON, D., JAIN, S., MAAS, R., MEHTA, P., ET AL. The myria big data management and analytics system and cloud services. In *CIDR* (2017).

[67] YADWADKAR, N. J., HARIHARAN, B., GONZALEZ, J. E., SMITH, B., AND KATZ, R. H. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing* (2017), ACM, pp. 452–465.