SOPHIA: Online Reconfiguration of Clustered NoSQL Databases for Time-Varying Workloads

Ashraf Mahgoub, *Purdue University;* Paul Wood, *Johns Hopkins University;*
Alexander Medoff, *Purdue University;* Subrata Mitra, *Adobe Research;*
Folker Meyer, *Argonne National Lab;* Somali Chaterji and Saurabh Bagchi, *Purdue University*

https://www.usenix.org/conference/atc19/presentation/mahgoub

This paper is included in the Proceedings of the
2019 USENIX Annual Technical Conference.

July 10–12, 2019 • Renton, WA, USA

ISBN 978-1-939133-03-8

SOPHIA: Online Reconfiguration of Clustered NoSQL Databases for Time-Varying Workload

Ashraf Mahgoub, *Purdue University*
Paul Wood, *Johns Hopkins University*
Alexander Medoff, *Purdue University*
Subrata Mitra, *Adobe Research*
Folker Meyer, *Argonne National Lab*
Somali Chaterji, *Purdue University*
Saurabh Bagchi, *Purdue University*

Abstract

Reconfiguring NoSQL databases under changing workload patterns is crucial for maximizing database throughput. This is challenging because of the large configuration parameter search space with complex interdependencies among the parameters. While state-of-the-art systems can automatically identify close-to-optimal configurations for static workloads, they suffer for dynamic workloads as they overlook three fundamental challenges: (1) Estimating performance degradation during the reconfiguration process (such as due to database restart). (2) Predicting how transient the new workload pattern will be. (3) Respecting the application's availability requirements during reconfiguration. Our solution, SOPHIA, addresses all these shortcomings using an optimization technique that combines workload prediction with a cost-benefit analyzer. SOPHIA computes the relative cost and benefit of each reconfiguration step, and determines an optimal reconfiguration for a future time window. This plan specifies when to change configurations and to what, to achieve the best performance without degrading data availability. We demonstrate its effectiveness for three different workloads: a multi-tenant, global-scale metagenomics repository (*MG-RAST*), a bus-tracking application (*Tiramisu*), and an HPC data-analytics system, all with varying levels of workload complexity and demonstrating dynamic workload changes. We compare SOPHIA's performance in throughput and tail-latency over various baselines for two popular NoSQL databases, Cassandra and Redis.

1 Introduction

Automatically tuning database management systems (DBMS) is challenging due to their plethora of performance-related parameters and the complex interdependencies among subsets of these parameters [45, 64, 17]. For example, Cassandra has 56 performance tuning parameters and Redis has 46 such parameters. Several prior works like Rafiki [45], OtterTune [64], BestConfig [69], and others [17, 62, 61], have solved the problem of optimizing a DBMS when workload characteristics relevant to the data operations are relatively static. We call these "*static configuration tuners*". However, these solutions cannot decide on a set of configurations over a window of time in which the workloads are changing, i.e., what configuration to change to and when. Further, existing solutions cannot perform the reconfiguration of a cluster of database instances without degrading data availability.

Workload changes lead to new optimal configurations. However, it is not always desirable to switch to new configurations because the new workload pattern may be short-lived. Each reconfiguration action in clustered databases incurs costs because the server instance often needs to be restarted for the new configuration to take effect, causing a transient hit to performance during the reconfiguration period. In the case of dynamic workloads, the new workload may not last long enough for the reconfiguration cost to be recouped over a time window of interest to the system owner. Therefore, a proactive technique is required to estimate when executing a reconfiguration is going to be globally beneficial.

Fundamentally, this is where the drawback of all prior approaches to automatic performance tuning of DBMS lies—in the face of dynamic changes to the workload, they are either silent on when to reconfigure or perform a naïve reconfiguration whenever the workload changes. We show that a naïve reconfiguration, which is oblivious to the reconfiguration cost, actually *degrades* the performance for dynamic workloads relative to the default configurations and also relative to the best static configuration achieved using a static tuner with historical data from the system (Figure 3). For example, during periods of high dynamism in the read-write switches in a metagenomics workload in the largest metagenomics portal called MG-RAST [50], naïve reconfiguration degrades throughput by a substantial 61.8% over default.

Our Solution: We develop an online reconfiguration system—SOPHIA—for a NoSQL cluster comprising of multiple server instances, which is applicable to dynamic workloads with various rates of workload shifts. SOPHIA uses historical traces of the workload to train a workload predictor, which is used at runtime to predict future workload

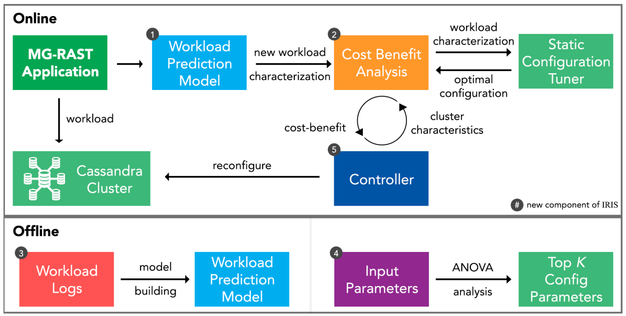


Figure 1: Workflow of SOPHIA *with offline model building and the online operation, plus the new components of our system. It also shows the interactions with the NoSQL cluster and a static configuration tuner, which comes from prior work.*

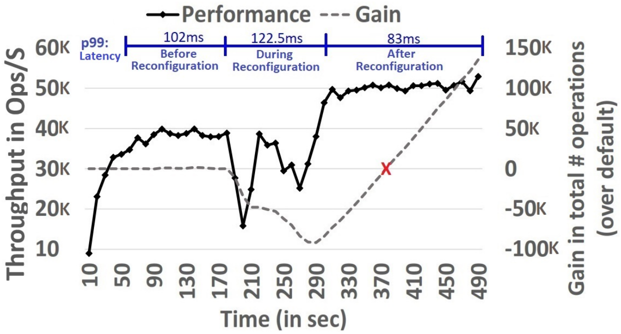


Figure 2: The effect of reconfiguration on the performance of the system. SOPHIA *uses the workload duration information to estimate the cost and benefit of each reconfiguration step and generates plans that are globally beneficial.*

patterns. Workload prediction is a challenging problem and has been studied in many prior works [43, 19, 51]. However, the workload predictor itself is not a contribution of SOPHIA, and it can operate with any workload predictor with sufficiently accurate and long-horizon predictions. SOPHIA searches the vast space of all possible reconfiguration plans, and hence determines the best plan through a novel Cost-Benefit-Analysis (CBA) scheme. For each shift in the predicted workload trace, SOPHIA interacts with any existing static configuration tuner (we use RAFIKI in our work because it is already engineered for NoSQL databases and is publicly available [15]), to quickly provide the optimal *point configurations* for the new workload and the estimated benefit from this new configuration. SOPHIA performs the CBA analysis, taking into account the predicted duration of the new workload and the estimated benefit from each reconfiguration step. Finally, for each reconfiguration step in the selected plan, SOPHIA initiates a distributed protocol to reconfigure the cluster without degrading data availability and maintaining the required data consistency requirement.

During its reconfiguration, SOPHIA can deal with different replication factors (RF) and consistency level (CL) requirements specified by the user. It ensures that the data remains continuously available through the reconfiguration process, with the required CL. This is done by controlling the number of server instances that are concurrently reconfigured. However, this is only possible when RF>CL. In cases where RF=CL, reconfiguring any node in the cluster will degrade data availability as every request will require a response from every replica before it is returned to the user. Therefore, we also implement an aggressive variant of our system (SOPHIA-AGGRESSIVE), which relaxes the data availability requirement in exchange for faster reconfiguration and hence better performance.

Evaluation Cases

We evaluate SOPHIA on two NoSQL databases, Cassandra [39] and Redis [7]. The first use case is based on real workload traces from the metagenomics analysis pipeline, MG-RAST [9, 49]. It is a global-scale metagenomics portal, the largest of its kind, which allows users to simultaneously upload their metagenomic data and use its computational pipeline. The 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. This presents a challenging use case as only 5 minutes or less of accurate lookahead is possible. The second use case is a bus-tracking application where read, scan, insert, and update operations are submitted to a backend database. The data has strong daily and weekly patterns to it. 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 three cases span the range of patterns and corresponding predictability of the workloads.

We compare our approach to existing solutions and show that SOPHIA increases throughput (and decreases tail-latency) under all dynamic workload patterns and for all types of queries, with no downtime. For example, SOPHIA achieves 24.5% higher throughput over default configurations and 21.5% higher throughput over a statically determined idealized optimal configuration in the bus-tracking workload. SOPHIA achieves 38% and 30% higher aggregate throughput over these two baselines in the HPC cluster workload. With SOPHIA's auto-tuning capability, Redis is able to operate through the entire range of workload sizes and read/write intensities, while the *vanilla Redis fails with large workloads*. The main contributions of SOPHIA are:

1. We show that state-of-the-art static tuners when applied to dynamic workloads degrade throughput below the state-of-practice of using the default parameter values and also degrade data availability.

2. SOPHIA performs cost-benefit analysis to achieve *long-horizon optimized performance for clustered NoSQL instances* in the face of dynamic workload changes, including unpredictable and fast changes to the workload.

3. SOPHIA executes a decentralized protocol to gracefully

switch over the cluster to the new configuration while respecting the data consistency guarantees and keeping data continuously available to users.
First, we show the improvement of using SOPHIA to tune a Cassandra cluster. Afterwards, we show how SOPHIA can be used to tune Redis and improve its performance. The rest of the paper is organized as follows. Section 2 provides an overview of our solution SOPHIA. We provide a background on Cassandra and its sensitivity to configuration parameters and on static configuration tuners in Section 3. We describe our solution in Section 4. We provide details of the workloads and our implementation in Section 5. We give the evaluation results in Section 6 and finally conclude.

2 Overview of SOPHIA

Here we give an overview of the workflow and the main components of SOPHIA. A schematic of the system is shown in Fig. 1. Details of each component are in Sec. 4.

SOPHIA runs as a separate entity outside the Cassandra cluster. It measures the workload by intercepting and observing received queries at the entry point(s) to Cassandra. Periodically, SOPHIA queries the **Workload Predictor** (box 1 in figure) to determine if any future workload changes exist that may merit a reconfiguration—no change also contributes information for the SOPHIA algorithm. Also, an event-driven trigger occurs when the predictor identifies a workload change. The prediction model is initially trained from representative workload traces from prior runs of the application and incrementally updated with additional data as SOPHIA operates. With the predicted workload, SOPHIA queries a static configuration tuner that provides the optimal configuration for a single point in time in the predicted workload. The static configuration tuner is initially trained on the same traces from the system as the workload predictor. Similarly, the static configuration tuner is also trained incrementally like the workload predictor.
The **Dynamic Configuration Optimizer** (box 2) generates a time-varying reconfiguration plan for a given look-ahead window using cost-benefit analysis (CBA). This plan gives both the time points when reconfiguration should be initiated *and* the new configuration parameters at each such time point. The CBA considers both the static, point solution information and the estimated, time-varying workload information. It is run every look-ahead time window apart or when the workload characteristics have changed significantly enough. The **Controller** (box 3) initiates a distributed protocol to gracefully switch the cluster to new configurations in the reconfiguration plan (Sec.4.5). This controller is conceptually centralized but replicated and distributed in implementation using off-the-shelf tools like ZooKeeper. SOPHIA decides how many instances to switch concurrently such that the cluster always satisfies the user's availability and consistency requirements. The Workload Predictor is located at a point where it can observe the aggregate workload such as at a gateway to the database cluster or by querying each database instance for its near past workload profile. The Dynamic Configuration Optimizer runs at a dedicated node close to the workload monitor. A distributed component runs on each node to apply the new reconfiguration plan.
Cost-Benefit Analysis in the Reconfiguration Plan
Each reconfiguration has a cost, due to changing parameters that require restarting or otherwise degrading the database services, e.g., by flushing the cache. The CBA in SOPHIA calculates the costs of implementing a reconfiguration plan by determining the number, duration, and magnitude of degradations. If a reconfiguration plan is found globally beneficial, the controller initiates the plan, else it is rejected. This insight, and the resulting protocol design to decide *whether* and *when* to reconfigure, are the fundamental contributions of SOPHIA.
Now we give a specific example of this cost-benefit trade-off from real MG-RAST workload traces. Consider the example in Fig. 2 where we apply SOPHIA's reconfiguration plan to a cluster of 2 servers with an availability requirement that at least 1 of 2 be online (i.e. CL=1). The Cassandra cluster starts with a read-heavy workload but with a configuration C_1 (Cassandra's default), which favors a write-heavy workload and is therefore suboptimal. With this configuration, the cluster provides a throughput of ∼40,000 ops/s and a tail latency of 102 ms (P99), but a better read-optimized configuration C_2 exists, providing ∼50,000 ops/s and a tail latency of 83 ms. The Cassandra cluster is reconfigured to the new C_2 configuration setting, using SOPHIA's controller, resulting in a temporary throughput loss due to the transient unavailability of the server instances as they undergo the reconfiguration, one instance at a time given the specified availability requirement. Also during the reconfiguration period, the tail latency increases to 122.5 ms on average. The two dips in throughput at 200 and 270 seconds correspond to the two server instances being reconfigured serially, in which two spikes in tail latency of 180 ms are observed. We plot, using the dashed line, the gain (benefit minus cost) over time in terms of the total # operations served relative to the default configuration. We see that there is a crossover point (the red X point) with the duration of the new workload pattern. If the predicted workload pattern lasts longer than this threshold (190 seconds from the beginning of reconfiguration in our example), then there is a gain from this step and SOPHIA would include it in the plan. Otherwise, the cost will outweigh the benefit, and any solution implemented without the CBA risks degrading the overall system performance. *Thus, a naïve solution (a simple extension of all existing static configuration tuners) that always reconfigures to the best configuration for the current workload will actually degrade performance for any reasonably fast-changing workload.* Therefore, a workload predictor and a cost-benefit analysis model are needed to develop a reconfiguration plan that achieves globally optimal performance over time.

3 Background

Overview of Apache Cassandra: Cassandra is one of the most popular NoSQL databases that is being used by many companies such as Apple, eBay, Netflix and many others [8]. This is because of its durability, scalability, and fault-tolerance, which are essential features for production deployments with large volumes of data. To be able to support a wide range of applications and access patterns, Cassandra (like many other DBMS) exposes many configuration parameters that control its internal behavior and affect its performance. This is intended to customize the DBMS for widely different applications. According to the Cassandra architecture, it caches writes in an in-memory Log-structured merge tree [58] called *Memtable* for a certain period of time. Afterwards, all Memtables get flushed to their corresponding persistent representation on disk called *SSTables*. The same flushing process can be triggered if the size of the Memtable exceeds a specific threshold. A Memtable is always flushed to a new SSTable, which is never updated after construction. Consequently, a single key can be stored accross many SSTables with different timestamps, and therefore a read request to that key will require Cassandra to scan through all existing SSTables and retrieve the one with the latest timestamp. To keep the number of SSTables manageable, Cassandra applies a compaction strategy, which combines a number of old SSTables into one while removing obsolete records. This achieves better performance for reads, but is also a heavy operation that consumes CPU and memory and can negatively impact the performance for writes during compaction.

Dynamic Workloads in Cassandra: Optimal values of these performance-sensitive parameters are dependent on the workload. For example, we find empirically that size-tiered compaction strategy achieves 44% better performance for write-heavy workloads than leveled compaction strategy, while leveled compaction strategy achieves 90% better performance for read-heavy workloads (Figure 3). When the workload changes, the optimal parameters for the new workload will likely change as well. An incremental approach is desired, rather than restarting all servers concurrently, which renders all the data unavailable during reconfiguration.

Workloads in our pipeline have shifts in the number of requests/s and also the relative ratio of the different operations on the database (i.e., transaction mixture). Therefore, SOPHIA needs to react in an agile manner to such shifts. For example, MG-RAST traces show 443 sharp (more than 80% change) shift/day on average, mostly from read-heavy to write-heavy workloads and vice-versa. For the bus-tracking application, a smaller, but still significant, value of 63 shift/day is observed. The static tuners cannot handle such dynamism and cannot even pick a single parameter set that will on an average give the highest throughput aggregated over a window of time because of the lack of lookahead and also the lack of the Cost-Benefit analysis model.

4 Design of SOPHIA

SOPHIA seeks to answer the following two broad questions: When should the cluster be reconfigured? How should we apply the reconfiguration steps? The answer to the first question leads to what we call a *reconfiguration plan*. The answer to the second question is given by our distributed protocol that reconfigures the various server instances in rounds. Next, we describe SOPHIA's components.

4.1 Workload Modeling and Forecasting: In a generic sense, we can define the workload at a particular point in time as a vector of various time-varying features:

$$\boldsymbol{W}(t) = \{p_1(t), p_2(t), p_3(t), ..., p_n(t)\} \tag{1}$$

where the workload at time t is $\boldsymbol{W}(t)$ and $p_i(t)$ is the time-varying i-th feature. These features may be directly measured from the database, such as the load (i.e., requests/s) and the occupancy level of the database, or they may come from the computing environment, such as the number of users or jobs in a batch queue. These features are application dependent and are identified by analyzing the application's historical traces. Details for time-varying features of each application are described in Section 5. For workload forecasting, we discretized time into sliced T_d durations ($= 30s$ in our model) to bound the memory and compute cost. We then predicted future workloads as:

$$\boldsymbol{W}(t_{k+1}) = f_{\text{pred}}(\boldsymbol{W}(t_k), \boldsymbol{W}(t_{k-1}), ..., \boldsymbol{W}(t_0)) \tag{2}$$

where k is the current time index into T_d-wide steps. For ease of exposition for the rest of the paper, we drop the term T_d, assuming implicitly that this is one time unit. The function f_{pred} is any function that can make such a prediction, and in SOPHIA, we utilize a simple Markov-Chain model for MG-RAST and Bus-Tracking, while we use a deterministic, fully accurate output from a batch scheduler for the HPC data analytics workload, i.e., a perfect f_{pred}. However, more sophisticated estimators, such as neural networks [43, 31, 33], even with some degree of interpretability [32], have been used in other contexts and SOPHIA is modular enough to use any such predictor.

4.2 Adapting a Static Configuration Tuner for SOPHIA: SOPHIA uses a static configuration tuner (RAFIKI), designed to work with Cassandra, to output the best configuration for the workload at any given point in time. RAFIKI uses Analysis-of-variance (ANOVA) [55] in order to estimate the importance of each parameter. It selects the top-k parameters in its configuration optimization method, which is in turn determined by a significant drop-off in the importance score. The ability to adapt optimized "kernels" to build robust algorithms comes from our vision to accelerate the pipeline of creating efficient algorithms, conceptualized in Sarvavid [44]. The 7 highest performance-sensitive parameters for all three of our workloads are: (1) Compaction method, (2) # Memtable flush writers, (3) Memory-table clean-up threshold, (4) Trickle fsync, (5) Row cache size, (6) Number of concurrent writers, and (7) Memory heap space. These parameters vary with respect to the reconfiguration cost that they entail. The change to the compaction method incurs the highest cost as it causes every Cassandra instance to read all its SSTables and re-write them to the disk in the new format. However, due to interdependability between these parameters, the compaction frequency is still being controlled by reconfiguring the second and third parameters with the cost of a server restart. Similarly, parameters 4, 6, 7 need a server restart for their new values to take effect and these cause the next highest level of cost. Finally, some parameters (parameter 5 in our set) can be reconfigured without needing a server restart and therefore have the least level of cost.

In general, the database system has a set of performance-impactful configuration parameters $\boldsymbol{C} = \{c_1, c_2, \cdots, c_n\}$ and the optimal configuration $\boldsymbol{C}_{opt}$ depends on the particular workload $\boldsymbol{W}(t)$ executing at that point in time. In order to optimize performance across time, SOPHIA needs the static tuner to provide an estimate of throughput for both the optimal and the current configuration for any workload:

$$H_{\text{sys}} = f_{\text{ops}}(\boldsymbol{W}(t), \boldsymbol{C}_{\text{sys}}) \qquad (3)$$

where H_{sys} is the throughput of the cluster of servers with a configuration $\boldsymbol{C}_{\text{sys}}$ and $f_{\text{ops}}(\boldsymbol{W}(t), \boldsymbol{C}_{\text{sys}})$ provides the system-level throughput estimate. $\boldsymbol{C}_{\text{sys}}$ has $N_s \times |\boldsymbol{C}|$ dimensions for N_s servers and C different configurations. Cassandra by careful design achieves efficient load balancing across multiple instances whereby each contributes approximately equally to the overall system throughput [39, 20]. Thus, we define a single server average performance as $H_i = \frac{H_{\text{sys}}}{N_s}$.

From these models of throughput, optimal configurations can be selected for a given workload:

$$C_{\text{opt}}(\boldsymbol{W}(t)) = \arg\max_{\boldsymbol{C}_{\text{sys}}} H_{\text{sys}} = \arg\max_{\boldsymbol{C}_{\text{sys}}} f_{\text{ops}}(\boldsymbol{W}(t), \boldsymbol{C}_{\text{sys}}) \qquad (4)$$

In general, $\boldsymbol{C}_{\text{opt}}$ can be unique for each server, but in SOPHIA, it is the same across all servers and thus has a dimension of $|\boldsymbol{C}|$ making the problem computationally easier. This is due to the fact that SOPHIA makes a design simplification—it performs the reconfiguration of the cluster as an atomic operation. Thus, it does not abort a reconfiguration action mid-stream and all servers must be reconfigured in round i prior to beginning any reconfiguration of round $i+1$. We also speed up the prediction system f_{ops} by constructing a cached version with the optimal configuration $\boldsymbol{C}_{opt}$ for a subset of $\boldsymbol{W}$ and using nearest-neighbor lookups whenever a near enough neighbor is available.

4.3 Dynamic Configuration Optimization:

SOPHIA's core goal is to maximize the total throughput for a database system when faced with dynamic workloads. This introduces time-domain components into the optimal configuration strategy $\boldsymbol{C}^T_{\text{opt}} = \boldsymbol{C}_{\text{opt}}(\boldsymbol{W}(t))$, for all points in (discretized) time till a lookahead T_L. Here, we describe the mechanism that SOPHIA uses for CBA modeling to construct the best reconfiguration plan (defined formally in Eq. 5) for evolving workloads.

In general, finding solutions for $\boldsymbol{C}^T_{\text{opt}}$ can become impractical since the possible parameter space for $\boldsymbol{C}$ is large and the search space increases linearly with T_L. To estimate the size of the configuration space, consider that in our experiments we assumed a lookahead $T_L = 30$ minutes and used 7 different parameters, some of which are continuous, e.g., `Memory-table clean-up threshold`. If we take an underestimate of each parameter being binary, then the size of the search space becomes $2^{7\times30} = 1.6 \times 10^{63}$ points, an impossibly large number for exhaustive search. We define a compact representation of the reconfiguration points (Δ's) to easily represent the configuration changes. The maximum number of switches within T_L, say M, is bounded since each switch takes a finite amount of time. The search space for the dynamic configuration optimization is then $Combination(TL, M), M) \times |\boldsymbol{C}|$. This comes from the fact that we have to choose at most M points to switch out of all the T_L time points and at each point there are $|\boldsymbol{C}|$ possible configurations. We define the reconfiguration plan as:

$$\boldsymbol{C}^{\Delta}_{\text{sys}} = [\boldsymbol{T} = \{t_1, t_2, ..., t_M\}, \boldsymbol{C} = \{C_1, C_2, ..., C_M\}] \qquad (5)$$

where t_k is a point in time and C_k is the configuration to use at t_k. Thus, the reconfiguration plan gives *when* to perform a reconfiguration and at each such point, *what* configuration to choose.

The objective for SOPHIA is to select the best reconfiguration plan $(\boldsymbol{C}^{\Delta}_{\text{sys}})^{\text{opt}}$ for the period of optimization, lookahead time T_L:

$$(\boldsymbol{C}^{\Delta}_{\text{sys}})^{\text{opt}} = \arg\max_{\boldsymbol{C}^{\Delta}_{\text{sys}}} B(\boldsymbol{C}^{\Delta}_{\text{sys}}, \boldsymbol{W}) - L(\boldsymbol{C}^{\Delta}_{\text{sys}}, \boldsymbol{W}) \qquad (6)$$

where $\boldsymbol{C}^{\Delta}_{\text{sys}}$ is the reconfiguration plan, B is the benefit function, and L is the cost (or loss) function, and $\boldsymbol{W}$ is the time-varying workload description. Detailed derivation of functions B and L is shown in Supplemental Material (Section 9.1). When SOPHIA will extend to allow scale out, we will have to consider the data movement volume as another cost to minimize. The L function captures the opportunity cost of having each of N_s servers offline for T_r seconds for the new workload versus if the servers remained online with the old configuration. As the node downtime due to reconfiguration never exceeds Cassandra's threshold for declaring a node is dead (3 hours by default), data-placement tokens are not re-assigned due to reconfiguration. Therefore, we do not include cost of data movement in functions L. SOPHIA can work with any reconfiguration cost, including different costs for different parameters—these can be fed into the loss function L.

The objective is to maximize the time-integrated gain (benefit – cost) of the reconfiguration from Eq. (6). The three unknowns in the optimal plan are M, $\boldsymbol{T}$, and $\boldsymbol{C}$, from Eq. (5). If only R servers can be reconfigured at a time (explained in Sec. 4.5 how R is calculated), at least $T_r \times \frac{N_s}{R}$ time must elapse between two reconfigurations. This puts a limit on M, the maximum number of reconfigurations that can occur in the lookahead period T_L.

A greedy solution for Eq. (6) that picks the first configuration change with a net-increase in benefit may produce suboptimal $\boldsymbol{C}_{\text{sys}}^{\Delta}$ over the horizon T_L because it does not consider the coupling between multiple successive workloads. For example, considering a pairwise sequence of workloads, the best configuration may not be optimal for either $\boldsymbol{W}(t_1)$ or $\boldsymbol{W}(t_2)$ but *is* optimal for the paired sequence of the two workloads. This could happen if the same configuration gives reasonable performance for $\boldsymbol{W}(t_1)$ or $\boldsymbol{W}(t_2)$ and has the advantage that it does not have to switch during this sequence of workloads. This argument can be naturally extended to longer sequences of workloads.

A T_L value that is too long will cause SOPHIA to include decision points with high prediction errors, and a value that is too short will cause SOPHIA to make almost greedy decisions. The appropriate lookahead period is selected by benchmarking the non-monotonic but convex throughput while varying the lookahead duration and selecting the point with maximum end-to-end throughput. We give our choices for our three applications when describing the first experiment with each application (Section 6).

4.4 Finding Optimal Reconfiguration Plan with Genetic Algorithms: We use a heuristic search technique, Genetic Algorithms or GA, to find the optimal reconfiguration plan. Although meta-heuristics like GA do not guarantee finding global optima, they have two desirable properties for SOPHIA. Our space is non-convex because many of the parameters impact the same resources such as CPU, RAM, and disk, and settings of one parameter impact the others.Therefore, greedy or gradient descent-based searches are prone to converge to a local optima. Also the GA's tunable completion is needed in our case for speedy decisions, as the optimizer executes in the critical path.

The representation of the GA solution incorporates two parts. First, the chromosome orientation, which is simply the reconfiguration plan (Eq. 5). The second part is the fitness function definition used to assess the quality of different reconfiguration plans. For this, we use the cost-benefit analysis as shown in Eq. 6 where fitness is the total number of operations (normalized for bus-tracking traces to account for different operations' scales) for the T_L window for the tested reconfiguration plan and given workload. We build a simulator to apply the individual solutions and to collect the corresponding fitness values, which are used to select the best solutions and to generate new solutions in the next generation. We utilize a Python library, `pyeasyga`, with 0.8 crossover fraction and population size of 200. We run 10 concurrent searches and pick the best configuration from those. The runtime of this algorithm is dependent on the length of the lookahead period and the number of decision points. For MG-RAST, the GA has 30 decision points in the lookahead period and results in execution time of 30-40 sec. For the HPC workload, the number of decision points is 180 as it has a longer lookahead period, resulting in a runtime of 60-70 sec. For the bus-tracking workload, the GA has 48 decision points and a runtime of 20-25 sec. The GA is typically re-run toward the end of the lookahead period to generate the reconfiguration plan for the next lookahead time window. Also, if the actual workload is different from the predicted workload, the GA is re-invoked. This last case is rate limited to prevent too frequent invocations of the GA during (transient) periods of non-deterministic behavior of the workload.

4.5 Distributed Protocol for Online Reconfiguration: Cassandra and other distributed databases maintain high availability through configurable redundancy parameters, consistency level (CL) and replication factor (RF). CL controls how many confirmations are necessary for an operation to be considered successful. RF controls how many replicas of a record exist throughout the cluster. Thus, a natural constraint for each record is RF $\geq$ CL. SOPHIA queries token assignment information (where a token represents a range of hash values of the primary keys which the node is responsible for) from the cluster, using tools that ship with all popular NoSQL distributions (like `nodetool ring` for Cassandra), and hence constructs what we call a *minimum availability subset* (N_{minCL} for short). We define this subset as the minimum subset of nodes that covers at least CL replicas of *all* keys. To meet CL requirements, SOPHIA insures that N_{minCL} nodes are operational at any point of time. Therefore, SOPHIA makes the design decision to configure up to $R = N_s - N_{minCL}$ servers at a time, where N_{minCL} depends on RF, CL, and token assignment. If we assume a single token per node (Cassandra's default with vnodes disabled), then a subset of $\lceil \frac{N_s}{RF} \rceil$ nodes covers all keys at least once. Consequently, N_{minCL} becomes CL $\times \lceil \frac{N_s}{RF} \rceil$ to cover all keys at least CL times. Thus, the number of reconfiguration steps $= \frac{N_s}{R} = \frac{RF}{RF-CL}$ becomes independent of the cluster size N_s.

In the case where RF = CL, N_{minCL} becomes equivalent to N_s and hence SOPHIA cannot reconfigure the system, without harming data availability, hence we use the SOPHIA-AGGRESSIVE variant in that case. However, we expect most systems with high consistency requirements to follow a read/write quorum with CL = $\lceil \frac{\text{RF}}{2} \rceil$ [23]. Note that SOPHIA reduces the number of available data replicas during the transient reconfiguration periods, and hence reduces the system's resilience to additional failures. However, one optional parameter in SOPHIA is how many failures during reconfiguration the user will want to tolerate (our experiments were run with zero). This is a high-level parameter that is intuitive to set by the database admin. Also notice that data that existed on the offline servers prior to reconfiguration is not lost due to the drain step, but data written during the transient phase has lower redundancy until the reconfigured servers get back online. In order to reconfigure a Cassandra cluster, SOPHIA performs the following steps, *R* server instances at a time:

51. Drain: Before shutting down a Cassandra instance, we flush the entire Memtable to disk by using Cassandra's tool `nodetool drain` and this ensures that there are no pending commit logs to replay upon a restart.

1. **Drain**: Before shutting down a Cassandra instance, we flush the entire Memtable to disk by using Cassandra's tool `nodetool drain` and this ensures that there are no pending commit logs to replay upon a restart.
2. **Shutdown**: The Cassandra process is killed on the node.
3. **Configuration file**: Replace the configuration file with new values for all parameters that need changing.
4. **Restart**: Restart the Cassandra process on the same node.
5. **Sync**: SOPHIA waits for Cassandra's instance to completely rejoin the cluster by letting a coordinator know of where to locate the node and then synchronizing the missed updates during the node's downtime.

In Cassandra, writes for down nodes are cached by available nodes for some period and re-sent to the nodes when they rejoin the cluster. The time that it takes to complete all these steps for one server is denoted by T_r, and T_R for the whole cluster, where $T_R = T_r \times \frac{RF}{RF-CL}$. During all steps 1-5, additional load is placed on the non-reconfiguring servers as they must handle the additional write and read traffic. Step 5 is the most expensive and typically takes 60-70% of the total reconfiguration time, depending on the amount of cached writes. We minimize step 4 practically by installing binaries from the RAM and relying on draining rather than commit-log replaying in step 1, reducing pressure on the disk.

5 Datasets

MG-RAST Workload: We use real workload traces from MG-RAST, the leading metagenomics portal operated by the US Department of Energy. As the amount of data stored by MG-RAST has increased beyond the limits of traditional SQL stores (23 PB as of August 2018), it relies on a distributed NoSQL Cassandra database cluster. Users of MG-RAST are allowed to upload "jobs" to its pipeline, with metadata to annotate job descriptions. All jobs are submitted to a computational queue of the US Department of Energy private Magellan cloud. We analyzed 80 days of query trace from the MG-RAST system from April 19, 2017 till July 9, 2017. From this data, we make several observations: (i) Workloads' read ratio (RR) switches rapidly with over 26,000 switches in the analyzed period. (ii) A negative correlation of -0.72 is observed between the Workloads' read ratio and number of requests/s (i.e., load). That is due to the fact that most of the write operations are batched to improve network utilization. (iii) Majority (i.e., more than 80%) of the switches are abrupt, from RR=0 to RR=1 or vice versa. (iv) KRD (key reuse distance) is very large. (v) No daily or weekly workload pattern is discernible, as expected for a globally used cyberinfrastructure.

Bus Tracking Application Workload: Secondly, we use real workload traces from a bus-tracking mobile application called **Tiramisu** [43]. The system provides live tracking of the public transit bus system. It updates bus locations periodically and allows users to search for nearby bus stops. There are four types of queries—read, update, insert, and scan (retrieving all the records in the database that satisfy a given predicate, which is much heavier than the other operations). A sample of the traces is publicly available [42]. We trained our model using 40 days of query traces, while 18 days were used as testing data. We draw several observations from this data: (i) The traces show a daily pattern of workload switches. For example, the workload switches to scan-heavy in the night and switches to update-heavy in the early morning. (ii) The Workload is a mixture of Update, Scan, Insert, and Read operations in the ratio of 42.2%, 54.8%, 2.82%, and 0.18% respectively. (iii) KRD is very small. From these observations, we notice that the workload is very distinct from MG-RAST and thus provides a suitable point for comparison.

Simulated Analytics Workload: For long-horizon reconfiguration plans, we simulate synthetic workloads representative of batch data analytics jobs, submitted to a shared HPC queue. We integrate SOPHIA with a job scheduler (like PBS [27]), that examines jobs while they wait in a queue prior to execution. Thus, the scheduler can profile the jobs waiting in the queue, and hence forecast the aggregate workload over a lookahead horizon, which is equal to the length of the queue. We model the jobs on data analytics jobs submitted to a Microsoft Cosmos cluster [21] and as in that paper, we achieve high accuracy in predicting when a job will start executing. Thus, SOPHIA is able to drive long-horizon reconfiguration plans. Each job is divided into phases: a write-heavy phase resembling an upload phase of new data, a read-heavy phase resembling executing analytical queries to the cluster, and a third, write-heavy phase akin to committing the analysis results. However, some jobs can be recurring (as shown in [1, 21]) and running against already uploaded data. These jobs will execute the analysis phase directly, skipping the first phase. The size of each phase is a random variable with U(200,100K) operations, and whenever a job finishes, a new job is selected from the queue and executed. We vary the level of concurrency and have an equal mix of the two types of jobs and monitor the aggregate workload. Figure 11 in Supplemental Material shows the synthetic traces for three job sizes. With increase in concurrency, the aggregate pattern becomes smoother and the latency of individual jobs increases.

6 Experimental Results

Here we evaluate the performance of SOPHIA under different experimental conditions for the 3 applications. We use a simple query model typical for NoSQL databases and is in contrast to complex analytics queries supported by more complex database engines. Hence, our throughput is defined as the number of queries per second. In all experiments, we collect both throughput and tail latency (p99) performance metrics. However, since the two parameters have an almost perfect inverse relationship in all experiments, we omit tail latency (except in Figures 4 and 6). We evaluate SOPHIA on Amazon EC2 using instances of size M4.xlarge with 4 vCPU's, 16 GB of RAM, provisioned IOPS (SSD) EBS for storage and network bandwidth of 0.74 Gbits/s for all Cassandra servers and workload drivers. Each node is loaded with 6 GB of data initially (SOPHIA's performance is evaluated with greater data volumes in Experiment 4). We use multiple concurrent clients to saturate the database servers and aggregate the throughput and tail latency observed by every client.

Baseline Comparisons

We compare the performance of SOPHIA to baseline configurations (1-5). We consider 3 variants of SOPHIA (6-8).

(1) Default: The default configuration that ships with Cassandra. This configuration favors write-heavy workloads by design [48].

(2) Static Optimized: This baseline resembles the static tuner (RAFIKI) when queried to provide the one *constant* configuration that optimizes for the entire future workload. This is an impractically ideal solution since it is assumed here that the future workload is known perfectly. However, non-ideally no configuration changes are allowed dynamically.

(3) Naïve Reconfiguration: Here, when the workload changes, RAFIKI's provided reconfiguration is always applied, instantiated by concurrently shutting down all server instances, changing their configuration parameters, and restarting all of them. Practically, this makes data unavailable and may not be tolerable in many deployments such as user-facing applications. The static configuration tuners are silent about when the optimal configurations determined by them must be applied and this baseline is a logical instantiation of all of the prior work.

(4) ScyllaDB: The performance of NoSQL database ScyllaDB [57] in its vanilla form. ScyllaDB is touted to be a much faster (10X or higher) drop-in replacement to Cassandra [56]. This stands in for other self-tuning databases [30].

(5) Theoretical Best: This baseline resembles the theoretically best achievable performance over the predicted workload period. This is simulated by assuming Cassandra is running with the optimal configuration at any point of time and not penalizing it for the cost of reconfiguration. This serves as an upper bound for the performance.

(6) SOPHIA with Oracle: This is SOPHIA with a fully accurate workload predictor.

(7) SOPHIA-AGGRESSIVE: A variant from SOPHIA that prefers faster reconfiguration over data availability and is used only when RF=CL. SOPHIA-AGGRESSIVE violates the availability requirement by reconfiguring all servers at the same time. Unlike Naïve, it uses the CBA model to decide when to reconfigure, and therefore it does not execute reconfiguration every time the workload changes.

(8) SOPHIA: This is our complete system.

Major Insights

We draw some key insights from the experimental results. First, globally shared infrastructures with black-box jobs only allow for short-horizon workload predictions. This causes SOPHIA to take single-step reconfiguration plans and limits its benefit over a static optimized approach (Figure 3). In contrast, when job characteristics can be predicted well (bus tracking and data analytics applications), SOPHIA achieves significant benefit over both default and static optimized cases (Figures 4 and 5). This benefit stays even when there is significant uncertainty in predicting the exact job characteristics as shown in Figure 9. Second, Cassandra can be used in preference to the recent popular drop-in ScyllaDB, an auto-tuning database, with higher throughput across the entire range of workload types, as long as we overlay a dynamic tuner, such as SOPHIA, atop Cassandra (Figures 3 and 5). Third, as the replication factor increases while the number of server are fixed, the reconfiguration time of SOPHIA decreases, thus improving its benefit (Figure 7). Contrarily, as CL increases, the benefit of SOPHIA shrinks (Figure 7).Finally, SOPHIA is applied to a different NoSQL database, Redis, and solves a long-standing configuration problem with it, one which has caused Redis to narrow its scope to being an in-memory database only (Figure 10).

Experiment 1: MG-RAST Workload

We present our experimental evaluation with 20 test days of MG-RAST data. To zoom into the effect of SOPHIA with different levels of dynamism in the workload, we segment the workload into 4 scenarios and present those results in addition to the aggregated ones.

Workload Prediction Model: We created 16,512 training samples composed of $T_d = 5min$ steps across the 60 days MG-RAST workloads. We compare the performance of a first-order and a second-order Markov Chain model. We represent the states as the proportion of read operations during the T_d interval. We use a quantization level of 10% in the read ratio between different states. We categorize the test days into 4: "Slow", "Medium", and "Fast", by the frequency of switching from the read- to the write-intensive workloads and this maps to the average read ratios (RR) shown in Table 1. "Write" represents days with long write-heavy periods. Table 1 shows the prediction RMSE for the four representative workload scenarios. Because of the lack of application-level knowledge, in addition to the well-known uncertainty in job execution times in genomics pipelines [40], the Markov Chain model only provides accurate predictions for short time intervals. Moreover, increasing the order of the model has very little impact on the prediction performance and also increases the number of states (11 states in the First-order model vs. 120 states in the Second-order model). We also tried to train a more complex model (RNN) but its prediction quality was similar. We notice that the best accuracy is for the "Slow" scenario, whereas it drops below 50% for "Medium", and it is always below 50% for the "Fast" and "Write" scenarios. Because the "Slow" scenario is the most common (observed 74% of time in the training data), we use a value of $T_L = 5min$.

Table 1: RMSE for predicting MG-RAST and Bus-Tracking workloads.

MG-RAST					
MC-Order	First		Second		RR
Frequency	5m	10m	5m	10m	-
Slow	34.4%	56%	34%	55%	70%
Medium	59%	90%	59%	89%	50%
Fast	66%	93%	63%	89%	45%
Write	52.8%	76.1%	51.5%	75.5%	35%
Aggregate	43.7%	68.7%	43.4%	68.2%	-

Bus-Tracking		
Lookahead	First	Second
15m	6.9%	7.12%
1h	7.4%	7.4%
2h	7.9%	7.4%
5h	10%	7.5%
10h	13.7%	8%
#States	117	647

Performance Comparison:

Now we show the performance of SOPHIA with respect to the four workload categories. We first present the result with the smallest possible number of server instances, 4, run with operational MG-RAST's parameters RF=3 and CL=1 [35]. We show the result in terms of total operations for each test workload as well as a weighted average "combined" representation that models behavior for the entire MG-RAST workload. Figure 3 shows the performance improvements for our test cases.

From Figure 3, we see that SOPHIA always outperforms naïve in total ops/s (average of 31.4%) and individually in read (31.1%) and write (33.5%) ops/s. SOPHIA also outperforms the default for the slow and the mid frequency cases, while it slightly under performs in the fast frequency case. The average improvement due to SOPHIA across the 4 categories is 20.4%. The underperformance for the fast case is due to increased prediction error. Naïve baseline has a significant loss compared to default: 21.6%. The static optimized configuration (which for this workload favors read-heavy pattern) has a slightly higher throughput over SOPHIA by 6.3%. This is because the majority of the selected samples are read periods (RR=1), which hides the gain that SOPHIA achieves for write periods. However, we see that with respect to write operations, SOPHIA achieves 17.6% higher throughput than the static optimized configuration. Increased write throughput is critical for MG-RAST to support the bursts of intense writes. This avoids unacceptable queuing of writes, which can create bottlenecks for subsequent jobs that rely on the written shared dataset. Moreover, we observe that SOPHIA performs within 2-3% to SOPHIA w/ Oracle in all scenarios, which shows the minor impact of the workload predictor accuracy. For instance, SOPHIA w/ Oracle shows a 2% reduction in performance compared to SOPHIA in the slow trace. This is because Oracle has perfectly accurate predictions for $T_L = 5min$ only. With this very short lookahead, SOPHIA makes greedy reconfiguration decisions, and hence does not achieve globally optimal performance over other baselines.

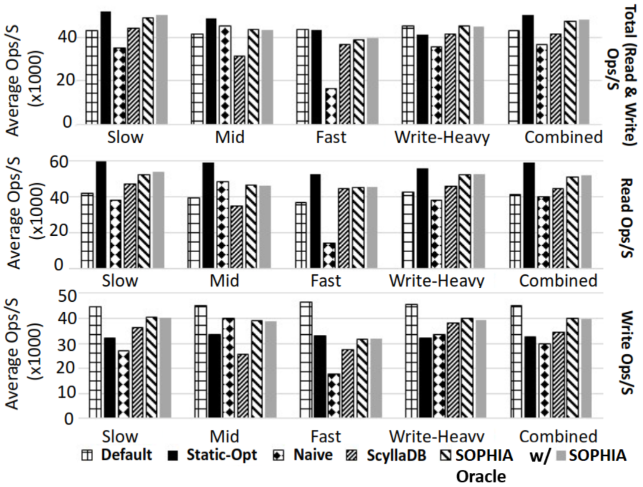


Figure 3: Improvement for four different 30-minute test windows from MG-RAST real traces over the baseline solutions.

ScyllaDB has an auto-tuning feature that is supposed to continuously react to changes in workload characteristics and the current state (such as, the amount of dirty memory state). ScyllaDB is claimed by its developers to outperform Cassandra in all workload mixes by an impressive 10X [56]. However, this claim is not borne out here and only in the read-heavy case (the "Slow" scenario) does ScyllaDB outperform. Even in this case, SOPHIA is able to reconfigure Cassandra at runtime and turn out a performance benefit over ScyllaDB. We conclude that based on this workload and setup, a system owner can afford to use Cassandra with SOPHIA for the *entire range* of workload mixes and not have to transition to ScyllaDB.

Experiment 2: Tiramisu Workload

We evaluate the performance of SOPHIA using the bus-tracking application traces. Figure 4 shows the gain of using SOPHIA over the various baselines. In this experiment, we report the normalized average Ops/s instead of the absolute average Ops/s metric. This means we normalize each of the 4 operation's throughputs by dividing by the maximum Ops/s seen under a wide variety of configurations and then average the 4 normalized throughputs. The reason for

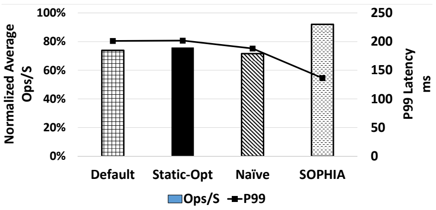


Figure 4: Gain of applying SOPHIA *to the bus-tracking application. We use 8 Cassandra servers with RF=3, CL=1. A 100% on the Y-axis represents the theoretical best performance.* SOPHIA *achieves improvements of 24.5% over default, 21.5% over Static-Opt, and 28.5% over naïve.*

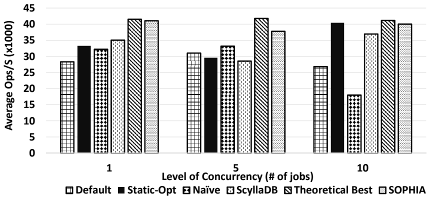


Figure 5: Improvement for HPC data analytics workload with different levels of concurrency. We notice that SOPHIA *achieves higher average throughput over all baselines*

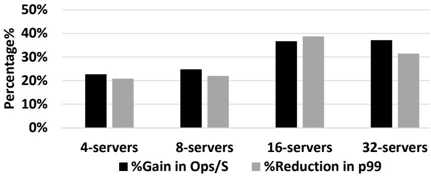


Figure 6: Improvement with scale using HPC workload with 5 jobs with RF=3 and CL=1. SOPHIA *provides consistent gains across scale because the cost of reconfiguration does not change with scale (for the same RF and CL). The higher gains for 16 and 32 servers is due to the use of M5 instances, which can be exploited by* SOPHIA *better than Static-Opt.*

this is that for this workload, different operations have vastly different throughput scales. For example, when the workload switches to a Scan-heavy phase, the performance of the cluster varies from 34 Ops/s to 219 Ops/s depending on the configuration of the cluster. For an Update-heavy phase, the performance varies from 1,739 Ops/s to 5,131 Ops/s. This is because Scan is a much heavier operation for the DBMS compared to Update.

SOPHIA outperforms default configuration by 24.5%, Static-Opt by 21.5%, and Naïve by 28.5%. The gains are higher because SOPHIA can afford longer lookahead times with accurate workload prediction. We notice that Naïve is achieving a comparable performance to both Default and Static-Opt configurations, unlike MG-RAST. This is because the frequency of workload changes is lower here. However, Naïve still renders the data unavailable during the reconfiguration period.

Workload Prediction Model: Unlike MG-RAST, the bus-tracking application traces show a daily pattern which allows our prediction model to provide longer lookahead periods with high accuracy (Table 1). We use a Markov Chain prediction model to capture the workload switching behavior. We start by defining the state of the workload as the proportion of each operation type in an aggregation interval (15 minutes in our experiments). For example, Update=40%, Read=20%, Scan=40%, Insert=0% represents a state of the workload. We use a quantization level of 10% in any of the 4 dimensions to define the state. We use the second-order Markov Chain with a lookahead period of 5 hours as this is when our prediction error is $\leq$ 8%. As expected theoretically, the second order model is more accurate at all lookahead times, since there is enough training data available for training the models. Seeing the seeming regular diurnal and weekly pattern in the workload, we create two simple predictor straw-men that uses only the current time-stamp or the current time-stamp and day of the week as input features to perform prediction. The predicted workload is the average of the mixtures at the previous 10 points. These predictors have unacceptably high RMSE of 31.4% and 24.0%. Therefore, although the workload is showing a pattern, we cannot generate the optimal plan once and use it for all subsequent days. Therefore, online workload monitoring and prediction is needed to achieve the best performance

Experiment 3: HPC Data Analytics

We evaluate the performance of SOPHIA using HPC data analytics workload patterns described in Section 5. Here our lookahead is the size of the job queue, which is conservatively taken as 1 hour. Figure 5 shows the result for the three levels of concurrency (1, 5, and 10 jobs). We see that SOPHIA outperforms the default for all the three cases, with average improvement of 30%. In comparison with Static-Opt (which is a different configuration in each of the three cases), we note that SOPHIA outperforms for the 1 job and 5 jobs cases by 18.9% and 25.7%, while it is identical for the 10 jobs case. This is because in the 10 jobs case, the majority of the workload lies between RR=0.55 and RR=0.85, and in this case, SOPHIA switches only once: from the default configuration to the same configuration as Static-Opt. We notice that SOPHIA achieves within 9.5% of the theoretical best performance for all three cases. We notice that SOPHIA achieves significantly better performance over Naïve by 27%, 13%, and 122% for the three cases. Naïve, in fact, degrades the performance by 32.9% (10 concurrent jobs). In comparison with ScyllaDB, SOPHIA achieves a performance benefit of 17.4% on average, which leads to a similar conclusion as in MG-RAST about the continued use of Cassandra.

Experiment 4: Scale-Out & Greater Volume

Figure 6 shows the behavior of SOPHIA with increasing scale using the data analytics workload. We show the comparison between SOPHIA and Static-Opt (all other baselines performed worse than Static-Opt). We use a weak scaling pattern, i.e., keeping the amount of data per server fixed while still operating close to saturation. We increase the number of shooters as well to keep the request rate per server fixed. By our design (Sec. 4.5), the number of reconfiguration steps stays constant with scale. We notice that the network bandwidth needed by Cassandra's gossip protocol increases with the scale of the cluster, causing the

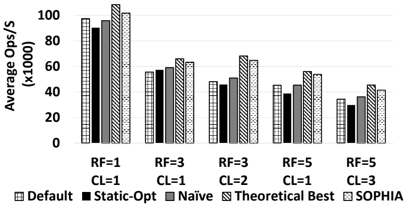


Figure 7: Effect of varying RF and CL on system throughput. We use a cluster of 8 nodes and compare the performance of SOPHIA *to Default, Static-Opt, and naïve.* SOPHIA *outperforms the static baselines and approaches the theoretical best as RF–CL increases.*

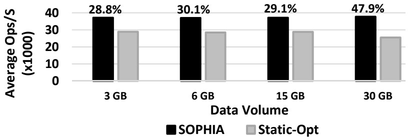


Figure 8: Effect on increasing data volume per node. We use a cluster of 4 servers and compare the performance to the static optimized. The results show that SOPHIA*'s gain is consistent with increasing data volumes per node.*

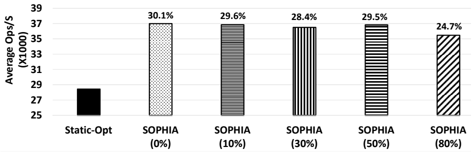


Figure 9: Effect of noise in workload prediction on the performance of SOPHIA *on the data analytics workload with level of concurrency = 5. The percentage represents the amount of noise added to the predicted workload pattern.*

network to become the bottleneck in the case of 16 and 32 servers when M4.xlarge instances are used. Therefore, we change the instance type to M5.xlarge in these cases (network bandwidth of 10 Gbit/s compared to 0.74 Gbit/s). The results show that SOPHIA's optimal reconfiguration policy has a higher performance over Static-Opt across all scales. Moreover, we see a higher gain in the cases of 16 and 32 servers since M5 instances have higher CPU power than M4 ones. This extra CPU capacity allows for faster leveled compaction, which is used by SOPHIA's plan (while Static-Opt uses size-tiered compaction), and hence leads to greater performance difference for reads.

We also evaluate SOPHIA with the same workload when the data volume per node increases. We vary the amount of data loaded initially into each node (in a cluster of 4 nodes) and measure the gain over Static-Opt in Figure 8. For the 30GB case, the data volume grows beyond the RAM size of the used instances (M4.xlrage with 16 GB RAM). We notice that the gain from applying SOPHIA's reconfiguration plan is consistent with increasing the data volume from 3 GB to 30 GB. We also notice that the gain increases for the case of 30 GB. This is also due to the different compaction methods used by Static-Opt (size-tiered) and SOPHIA (Leveled compaction), the later can provide better read performance with increasing data volumes. However, this benefit of Leveled compaction was not captured by RAFIKI predictions, which was trained on a single node with 6 GB of data. This can be addressed by either replacing RAFIKI by a data volume-aware static tuner, or re-training RAFIKI when a significant change in data volume per node occurs.

Experiment 5: Varying RF and CL

We evaluate the impact of applying SOPHIA to clusters with different RF and CL values. We use the HPC workload with 5 concurrent jobs. We fix the number of nodes to 8 and vary RF and CL as shown in Figure 7 (CL quorum implies CL = $\lceil$RF/2$\rceil$). We notice that SOPHIA continues to achieve better performance than all 3 static baselines for all RF, CL values. For RF=1, CL=1, we use SOPHIA-AGGRESSIVE because when RF=CL, we cannot reconfigure the cluster without degrading availability. The key observation is that SOPHIA's performance gets closer to the *Theoretical best* as RF-CL becomes higher (compare the RF=3,CL=1 to the RF=5,CL=1 case). This is because the number of steps SOPHIA needs to perform the reconfiguration is inversely proportional to RF-CL as discussed in Sec. 4.5). This allows SOPHIA to react faster to changes in the applied workload and thus achieve better performance. Moreover, we notice that the performance of the cluster degrades with increasing RF or CL. Increasing RF increases the data volume stored by each node, which increases the number of SSTables and hence reduces the read performance. Also increase in CL requires more nodes to respond to each request before acknowledgment to the client, which also reduces the performance.

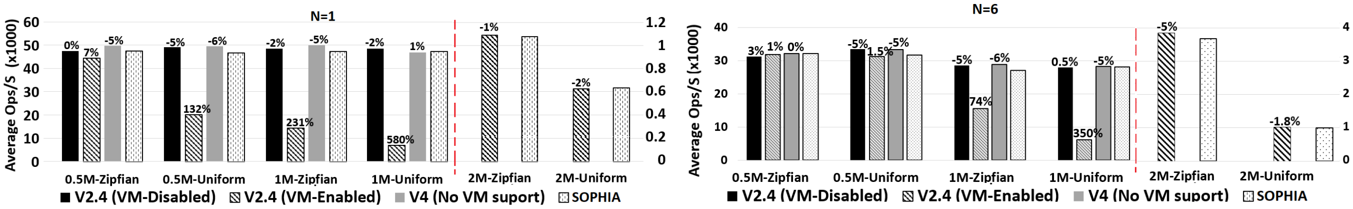
Experiment 6: Noisy Workload Predictions

We show how sensitive SOPHIA is to the level of noise in the predicted workload pattern. We use the HPC workload with 5 concurrent jobs. In HPC queues, there are two typical sources of such noise—an impatient user removing a job from the queue and the arrival of hitherto unseen jobs. We add noise to the predicted workload pattern $\sim$ U(-R,R), where R gives the level of noise. The resulting value is bounded between 0 and 1.

From Figure 9, we see that adding noise to SOPHIA slightly reduces its performance. However, such noise will not cause significant changes to SOPHIA's optimal reconfiguration plan. This is because SOPHIA treats each entry in the reconfiguration plan as a binary decision, i.e., reconfigure if Benefit $\geq$ Cost. So even if the values of both Benefit and Cost terms change, the same plan takes effect as long as the inequality still holds. This allows SOPHIA to achieve significant improvements for long-term predictions even with high noise levels.

Experiment 7: Redis Case Study

We now show a case study with the popular NoSQL database Redis, which has a long-standing pain point in setting a performance-critical parameter against changing workloads. Large-scale processing frameworks such as Spark can de-



(a)

(b)

Figure 10: Impact of tuning Redis' VM configuration parameters with SOPHIA ***with the data analytics workload. The percentage improvement of*** SOPHIA ***is shown on each bar and the right Y-axis is for the 2M jobs. A missing bar represents a failed job. We notice that the current Redis fails for large workloads (2M), while*** SOPHIA ***achieves the best of both worlds***

liver much higher performance when combined with Redis due to its shared distributed memory infrastructure [67, 38]. Redis is an in-memory data store (stores all keys and values in memory) while writing to persistent storage is only supported for durability. However, in its earlier versions (till V2.4), Redis used to offer a feature called *Virtual Memory* [36]. This feature allowed Redis to work on datasets larger than the available memory by swapping rarely used values to disk, while keeping all keys and hot values in memory. Since V2.4, this feature was removed as it caused serious performance degradation in many Redis deployments due to non-optimal setting as reflected in many posts in discussion forums [59, 25, 60]. We use SOPHIA to tune this feature and compare the performance to three baselines: (1) Redis V2.4 with VM-disabled (Default configuration), (2) Redis V2.4 with VM-enabled, (3) Redis V4 with default configuration (no VM support, most production-proven).

To tune Redis' *VM*, we investigate the impact of two configuration parameters: (1) *vm-enable*: a Boolean parameter that enables or disables the feature. (2) *vm-max-memory*: the memory limit after which Redis starts swapping least-recently-used values to disk. These features cannot be reconfigured without a server restart.

We tune the performance of Redis for simulated data analytics workloads that vary with respect to job sizes and access patterns. We use the popular YCSB (Yahoo! Cloud Serving Benchmark) tool [12] to simulate HPC workloads as in [28, 24]. We collect 128 data points for jobs that vary with respect to their sizes (0.5M, 1M, 2M), their access patterns (i.e., read-heavy vs write-heavy) and also their request distribution (i.e., Uniform vs Zipfian). We use 75% of the data points (selected uniformly) to train a linear regression model and 25% for testing. The model provides accurate prediction of throughput for any job and configuration (avg. R^2=0.92). Therefore, we use this simpler model in place of Rafiki.

Redis can operate in `Stand-alone` mode as well as a `Cluster` mode [37]. In `Cluster` mode, data is automatically sharded across multiple Redis nodes. We show the gain of using our system with Redis for both modes of operation. No replication is used for `Stand-alone` mode. Whereas for `Cluster` mode, we use a replication factor of 1 (i.e., a single slave per master). We use AWS servers of type C3.Large with 2 vCPUs and 3.75GB RAM each. Selecting such a small RAM server demonstrates the advantage of using *VM* with jobs that cannot fit in memory—1.8M records fit in the memory. We evaluate the performance SOPHIA on a single server (Figure 10a) as well as a cluster of 6 servers with 3 masters and 3 slaves (Figure 10b) and report the average throughput per server. From Figure 10 we see that for all record sizes and request distributions, SOPHIA performs the best or close to the best. If records fit in memory, then the no VM configuration is better. For Uniform distribution, VM performs worst, because records often have to be fetched from disk. If records do not fit in memory, the no VM options (including the latest Redis) will simply fail (hence the lack of a bar for 2.0M records). Thus, SOPHIA, by automatically selecting the right parameters for changing workloads, can achieve the best of both worlds: fast in-memory database, and leverage disk in case of spillover.

7 Related Work

Reconfiguration for dynamic workloads. A few systems such as Rafiki [45], Outtertune [64], and SmartConf [65] have been proposed to automatically find the optimal software configurations for a given workload. All these systems assume that the workload change is a long-term, and for which reconfiguring the system is always beneficial. However, we show that in many real-world workloads, both long-term and short-term changes are observed and therefore SOPHIA decides when and how to apply the new configurations to achieve globally optimal performance, while respecting the user's availability requirements.

Reconfiguration in databases. Several works proposed online reconfiguration for databases where the goal is not to update the configuration settings, but to control how the data is distributed among multiple server instances [14, 6, 22, 18, 66]. Among these, Morphus [22] targets MongoDB but cannot handle Cassandra due to its peer-to-peer topology and sharding. Tuba [5] reconfigures geo-replicated key-value stores by changing locations of primary and secondary replicas to improve overall utility of the storage system. Rocksteady [34] is a data migration protocol for in-memory databases to keep tail latency low with respect to workload changes. However, no parameter tuning or cost-benefit analysis is involved. A large body of work focused on choosing the best logical or physical design for static workloads in DBMS [13, 10, 70, 26, 11, 63, 2, 53, 54]. Another body of work improves performance for static workloads by finding correct settings for DBMS performance knobs [17, 16, 45, 69, 64]. SOPHIA performs *online* reconfiguration of the performance tuning parameters of distributed databases for *dynamic* workloads.

Reconfiguration in distributed systems and clouds. Several works have addressed the problem in the context of traditional distributed systems [29, 3] and cloud platforms [41, 68, 47, 46]. Some solutions present a theoretical approach, reasoning about correctness for example [3], while some present a systems-driven approach such as performance tuning for MapReduce clusters [41, 4]. BerkeleyDB [52] models probabilistic dependencies between configuration parameters. A recent work, *SmartConf* [65] provides a rigorous control-theoretic approach to continuously tune a distributed application in an application-agnostic manner. However, it cannot consider dependencies among the performance-critical parameters and cannot handle categorical parameters.

8 Conclusion

Current static tuners can provide close to optimal configuration for a static workload. However, they cannot determine whether and when to perform a configuration switch to maximize benefit over a future time horizon with changing workloads. We design SOPHIA to perform such reconfiguration while maintaining data availability and respecting the consistency level requirement. Our fundamental technical contribution is a cost-benefit analysis that analyzes the relative cost and the benefit of each reconfiguration action and determines a reconfiguration plan for a future time window. It then develops a distributed protocol to gracefully switch over the cluster from the old to the new configuration. We find benefits of SOPHIA applied to three distinct workloads (a metagenomics portal, a bus-tracking application, and a data analytics workload) over the state-of-the-art static tuners, for two NoSQL databases, Cassandra and Redis. Our work uncovers two big open challenges. How to do anticipatory configuration changes for future workload patterns? How to handle heterogeneity in the cluster, i.e., one where each server instance may have its own configuration and may contribute differently to the overall performance?

Acknowledgement

We thank our shepherd Asaf Cidon and all the reviewers for their insightful comments. This work is supported in part by NSF grant 1527262, NIH Grant 1R01AI123037, Lilly Endowment (Wabash Heartland Innovation Network - WHIN), and Adobe Research. This material was in part based upon research supported by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research, under contract DE-AC02-06CH11357. The funders had no role in the design or execution of the work. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

9 Supplemental Material

9.1 Cost-Benefit Analysis Derivation

Qualitatively, the benefit summed up over the time window is the increase in throughput due to the new optimal configuration option relative to the current configuration option.

$$B = \sum_{k \in [0,T_L]} H_{\text{sys}}(\boldsymbol{W}(k), \boldsymbol{C}^{T}_{\text{sys}}(k)) \tag{7}$$

where $\boldsymbol{W}(k)$ is the k-th element in the time-varying workload vector $\boldsymbol{W}$ and $\boldsymbol{C}^{T}_{\text{sys}}$ is the time-varying system configuration derived from $\boldsymbol{C}^{\Delta}_{\text{sys}}$. Likewise, the cost summed up over the time window is the loss in throughput incurred during the transient period of reconfiguration.

$$\begin{aligned} L &= \sum_{k \in [1,M]} \frac{\cancel{R}}{\cancel{N_s}} \cdot H_{\text{sys}}(\boldsymbol{W}(t_k), \boldsymbol{C}_k) \cdot \frac{\cancel{N_s}}{\cancel{R}} \cdot T_r \\ &= \sum_{k \in [1,M]} H_{\text{sys}}(\boldsymbol{W}(t_k), \boldsymbol{C}_k) \cdot T_r \end{aligned} \tag{8}$$

where $\boldsymbol{C}_k$ the configuration specified by the k-th entry of the reconfiguration plan $\boldsymbol{C}^{\Delta}_{\text{sys}}$, and T_r is the number of seconds a single server is offline during reconfiguration.

9.2 Synthetic HPC Workloads

For long-horizon reconfiguration plans, we simulate synthetic workloads representative of batch data analytics jobs, submitted to a shared HPC queue. We integrate SOPHIA with a job scheduler (like PBS [27]), that examines jobs while they wait in a queue prior to execution. Thus, the scheduler can profile the jobs waiting in the queue, and hence forecast the aggregate workload over a lookahead horizon, which is equal to the length of the queue. We model the jobs on data analytics jobs submitted to a Microsoft Cosmos cluster [21]

Figure 11 shows the simulated workload pattern for HPC Analytics case. We vary the level of concurrency and collect the aggregate workload observed by the NoSQL datastore.

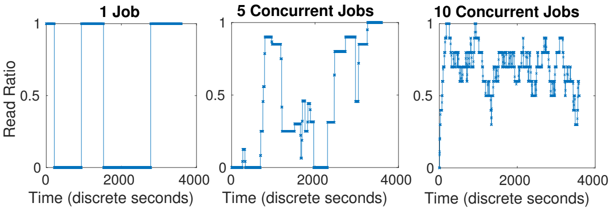


Figure 11: Simulated Workload patterns for 1, 5, and 10 concurrent jobs

References

[1] AGARWAL, S., KANDULA, S., BRUNO, N., WU, M.-C., STOICA, I., AND ZHOU, J. Re-optimizing data-parallel computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012), USENIX Association, pp. 21–21.

[2] 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).

[3] AJMANI, S., LISKOV, B., AND SHRIRA, L. Modular software upgrades for distributed systems. *ECOOP 2006–Object-Oriented Programming* (2006), 452–476.

[4] ANANTHANARAYANAN, G., AGARWAL, S., KANDULA, S., GREENBERG, A., STOICA, I., HARLAN, D., AND HARRIS, E. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Proceedings of the sixth conference on Computer systems* (2011), ACM, pp. 287–300.

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

[6] BARKER, S. K., CHI, Y., HACIGÜMÜS, H., SHENOY, P. J., AND CECCHET, E. Shuttledb: Database-aware elasticity in the cloud. In *ICAC* (2014), pp. 33–43.

[7] CARLSON, J. L. *Redis in action*. Manning Publications Co., 2013.

[8] CASSANDRA. Cassandra. `http://cassandra.apache.org/`, September 2018.

[9] 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 (2017), 235–244.

[10] 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.

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

[12] 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.

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

[14] DAS, S., NISHIMURA, S., AGRAWAL, D., AND EL ABBADI, A. Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. *VLDB Endowment* (2011).

[15] DCSL. Rafiki configuration tuner middleawre. `https://engineering.purdue.edu/dcsl/software/`, February 2018.

[16] DEBNATH, B. K., LILJA, D. J., AND MOKBEL, M. F. Sard: A statistical approach for ranking database tuning parameters. In *IEEE International Conference on Data Engineering Workshop (ICDEW)* (2008).

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

[18] ELMORE, A. J., DAS, S., AGRAWAL, D., AND EL ABBADI, A. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In *ACM SIGMOD International Conference on Management of data* (2011).

[19] FALOUTSOS, C., GASTHAUS, J., JANUSCHOWSKI, T., AND WANG, Y. Forecasting big time series: old and new. *Proceedings of the VLDB Endowment 11*, 12 (2018), 2102–2105.

[20] FEATHERSTON, D. Cassandra: Principles and application. *Department of Computer Science University of Illinois at Urbana-Champaign* (2010).

[21] 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 (Eurosys)* (2012), ACM, pp. 99–112.

[22] 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).

[23] GIFFORD, D. K. Weighted voting for replicated data. In *SOSP* (1979).

[24] GREENBERG, H., BENT, J., AND GRIDER, G. {MDHIM}: A parallel key/value framework for {HPC}. In *7th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 15)* (2015).

[25] GROUPS.GOOGLE. Problem with restart redis with vm feature on. https://groups.google.com/forum/#!topic/redis-db/EQA0WdvwghI, March 2011.

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

[27] HENDERSON, R. L. Job scheduling under the portable batch system. In *Workshop on Job Scheduling Strategies for Parallel Processing* (1995), Springer, pp. 279–294.

[28] JIA, Z., WANG, L., ZHAN, J., ZHANG, L., AND LUO, C. Characterizing data analysis workloads in data centers. In *2013 IEEE International Symposium on Workload Characterization (IISWC)* (2013), IEEE, pp. 66–76.

[29] 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.

[30] KHANDELWAL, A., AGARWAL, R., AND STOICA, I. Blowfish: Dynamic storage-performance tradeoff in data stores. In *NSDI* (2016), pp. 485–500.

[31] KIM, S. G., HARWANI, M., GRAMA, A., AND CHATERJI, S. EP-DNN: a deep neural network-based global enhancer prediction algorithm. *Scientific reports 6* (2016), 38433.

[32] KIM, S. G., THEERA-AMPORNPUNT, N., FANG, C.-H., HARWANI, M., GRAMA, A., AND CHATERJI, S. Opening up the blackbox: an interpretable deep neural network-based classifier for cell-type specific enhancer predictions. *BMC systems biology 10*, 2 (2016), 54.

[33] 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.

[34] KULKARNI, C., KESAVAN, A., ZHANG, T., RICCI, R., AND STUTSMAN, R. Rocksteady: Fast migration for low-latency in-memory storage. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 390–405.

[35] LAB, A. N., AND OF CHICAGO, U. MG-RAST's m5nr-table schema, 2019. [Online; accessed 1 29-August-2018].

[36] LABS, R. Redis Virtual Memory. https://redis.io/topics/virtual-memory, 2018. [Online; accessed 1-September-2018].

[37] LABS, R. Redis Cluster. https://redis.io/topics/cluster-tutorial, 2019. [Online; accessed 1-May-2019].

[38] LABS, R. Spark-Redis: Analytics Made Lightning Fast. https://redislabs.com/solutions/use-cases/spark-and-redis/, 2019. [Online; accessed 1-May-2019].

[39] LAKSHMAN, A., AND MALIK, P. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review 44*, 2 (2010), 35–40.

[40] LEIPZIG, J. A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics 18*, 3 (2017), 530–536.

[41] LI, M., ZENG, L., MENG, S., TAN, J., ZHANG, L., BUTT, A. R., AND FULLER, N. Mronline: Mapreduce online performance tuning. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing* (2014), ACM, pp. 165–176.

[42] MA, L. Tiramisu: Dataset for bus tracking applications. http://www.cs.cmu.edu/~malin199/data/tiramisu-sample/, June 2018. [Online; accessed 1-September-2018].

[43] 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 (SIGMOD)* (2018), ACM, pp. 631–645.

[44] MAHADIK, K., WRIGHT, C., ZHANG, J., KULKARNI, M., BAGCHI, S., AND CHATERJI, S. SARVAVID: A domain specific language for developing scalable computational genomics applications. In *International Conference on Supercomputing* (2016), ACM, p. 34.

[45] 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 International ACM/IFIP/USENIX Middleware Conference* (2017), pp. 1–13.

[46] MAJI, A. K., MITRA, S., AND BAGCHI, S. Ice: An integrated configuration engine for interference mitigation in cloud services. In *EEE International Conference on Autonomic Computing (ICAC)* (2015).

[47] MAJI, A. K., MITRA, S., ZHOU, B., BAGCHI, S., AND VERMA, A. Mitigating interference in cloud services by middleware reconfiguration. In *ACM International Middleware Conference* (2014).

[48] MEDIUM. Cassandra: Distributed key-value store optimized for write-heavy workloads. https://medium.com/coinmonks/cassandra-distributed-key-value-store-optimized-for-write-heavy-workloads-77f69c01388c. [Online; accessed 1-May-2019].

[49] MEYER, F., BAGCHI, S., CHATERJI, S., GERLACH, W., GRAMA, A., HARRISON, T., PACZIAN, T., TRIMBLE, W. L., AND WILKE, A. MG-RAST version 4lessons learned from a decade of low-budget ultra-high-throughput metagenome analysis. *Briefings in bioinformatics* (2017).

[50] MEYER, F., BAGCHI, S., CHATERJI, S., GERLACH, W., GRAMA, A., HARRISON, T., TRIMBLE, W., AND WILKE, A. Mg-rast version 4–lessons learned from a decade of low-budget ultra-high-throughput metagenome analysis. *Briefings in Bioinformatics 105* (2017).

[51] MOZAFARI, B., CURINO, C., JINDAL, A., AND MADDEN, S. Performance and resource modeling in highly-concurrent oltp workloads. In *Proceedings of the 2013 acm sigmod international conference on management of data* (2013), ACM, pp. 301–312.

[52] OLSON, M. A., BOSTIC, K., AND SELTZER, M. I. Berkeley db. In *USENIX Annual Technical Conference* (1999), pp. 183–191.

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

[54] 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).

[55] SCHEFFE, H. *The analysis of variance*, vol. 72. John Wiley & Sons, 1999.

[56] SCYLLADB. Scylla vs. Cassandra benchmark. http://www.scylladb.com/technology/cassandra-vs-scylla-benchmark-2/, October 2015.

[57] SCYLLADB. ScyllaDB. http://www.scylladb.com/, September 2017.

[58] SEARS, R., AND RAMAKRISHNAN, R. blsm: a general purpose log structured merge tree. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), ACM, pp. 217–228.

[59] SERVERFAULT. Should I use vm or set the maxmemory. https://serverfault.com/questions/432810/should-i-use-vm-or-set-the-maxmemory-with-redis-2-4, September 2012.

[60] STACKOVERFLOW. Redis Virtual Memory in 2.6. https://stackoverflow.com/questions/9205597/redis-virtual-memory-in-2-6, February 2012.

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

[62] TRAN, D. N., HUYNH, P. C., TAY, Y. C., AND TUNG, A. K. A new approach to dynamic self-tuning of database buffers. *ACM Transactions on Storage (TOS)* (2008).

[63] 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).

[64] 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.

[65] WANG, S., LI, C., HOFFMANN, H., LU, S., SENTOSA, W., AND KISTIJANTORO, A. I. Understanding and auto-adjusting performance-sensitive configurations. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2018).

[66] WEI, X., SHEN, S., CHEN, R., AND CHEN, H. Replication-driven live reconfiguration for fast distributed transaction processing. In *USENIX Annual Technical Conference* (2017).

[67] ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 10–10.

[68] ZHANG, R., LI, M., AND HILDEBRAND, D. Finding the big data sweet spot: Towards automatically recommending configurations for hadoop clusters on docker

containers. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on* (2015), IEEE, pp. 365–368.

[69] ZHU, Y., LIU, J., GUO, M., BAO, Y., MA, W., LIU, Z., SONG, K., AND YANG, Y. Bestconfig: Tapping the performance potential of systems via automatic configuration tuning. In *Symposium on Cloud Computing (SoCC)* (2017).

[70] ZILIO, D. C., AND SEVCIK, K. C. *Physical database design decision algorithms and concurrent reorganization for parallel database systems.* PhD Thesis Citeseer, 1999.