EUNOMIA: Online reconfiguration of Clustered NoSQL Databases for Time-Varying Workloads

Ashraf Mahgoub[α], Paul Wood[β], Alexander Medoff[α], Subrata Mitra[γ],
Folker Meyer[δ], Somali Chaterji[α], Saurabh Bagchi[α]

α: Purdue University, β: Johns Hopkins University, γ: Adobe Research, δ: Argonne National Lab

Abstract

Reconfiguring NoSQL databases in the face of changing workload patterns is crucial for maximizing database throughput. However, 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. This happens due to the three fundamental limitations. First, they do not account for performance degradation during the reconfiguration (say due to database restart, which is often needed to apply the new configuration). Second, they do not account for how transient the new workload pattern will be. Third, they overlook the application's availability requirements during reconfiguration. Our solution, EUNOMIA, addresses all these shortcomings. The fundamental technical contribution is a cost-benefit analyzer that computes the relative cost and the benefit of each reconfiguration action and determines a reconfiguration plan for a future time window. This specifies when to change and to what configurations. We demonstrate its effectiveness for three different workload traces: a multi-tenant, global-scale metagenomics repository (*MG-RAST*), a bus-tracking workload (*Tiramisu*), and an HPC data-analytics job queue, all with varying levels of workload complexity and demonstrating dynamic workload changes. We compare the benefit of EUNOMIA in throughput over the default, a static configuration, and a theoretically ideal solution for two widely popular NoSQL databases—Cassandra and Redis.

1 Introduction

Automatically tuning database management systems is challenging due to their plethora of performance-related parameters and the complex interdependencies among subsets of these parameters. For example, Cassandra (V3.0) has 56 performance tuning parameters and Redis (V4.0) has 46 such parameters, and different parameter combinations can affect performance in different ways. Several prior works like Rafiki [35], OtterTune [50], BestConfig [54], and others [15, 48, 47], 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 the availability of the data.

The drawback of static configuration tuners arises from the observation that 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. In the meanwhile, there is degraded throughput as a newly resurrected server instance is updated with current data and updates missed 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.

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 4). For example, during periods of high dynamism in the read-write switches in a metagenomics workload in the largest metagenomics portal called MG-RAST [38], naïve reconfiguration degrades performance by a substantial 61.8%.

Our Solution: EUNOMIA

We develop an online reconfiguration system—EUNOMIA—for a NoSQL cluster comprising of multiple server instances,

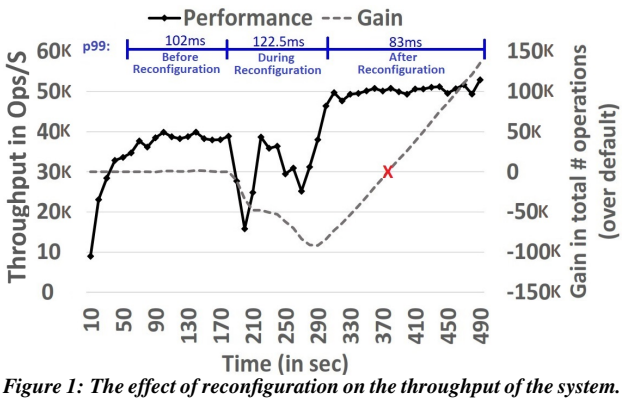

Figure 1: The effect of reconfiguration on the throughput of the system. If the new workload persists for greater than a certain duration (where the gain curve becomes positive), then reconfiguration should be done, else it is better to stay with the earlier configuration.

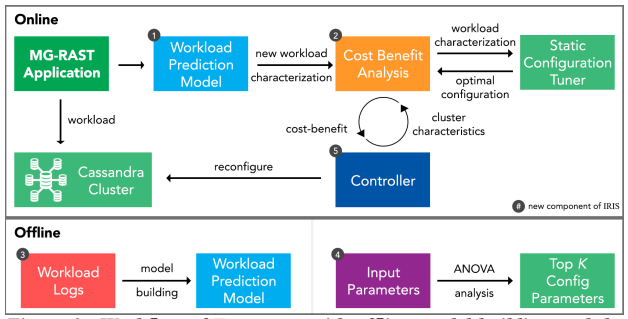

Figure 2: Workflow of EUNOMIA *with offline model building and the online operation, plus the new components of our system. It also shows the interactions with the Cassandra cluster and a static configuration tuner, which comes from prior work.*

which is applicable to dynamic workloads with various rates of change. EUNOMIA extracts information about current workload from a job scheduler and has a workload predictor to predict future workloads. However, the workload predictor itself is not a contribution of EUNOMIA, and it can operate with any workload predictor with sufficient accuracy. It then determines a long-horizon optimal reconfiguration plan through a novel Cost Benefit Analysis (CBA) scheme. When the workload changes, EUNOMIA 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 [13]), to quickly provide the optimal *point configurations* for the new workload and the estimated benefit from this new configuration. EUNOMIA performs the CBA analysis, taking into account the predicted duration of the new workload and the estimated benefit from a reconfiguration. If the CBA indicates that the benefit outweighs the cost, EUNOMIA initiates a distributed protocol to reconfigure the Cassandra cluster. During its reconfiguration, EUNOMIA 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 Cassandra server instances that are concurrently reconfigured.

Evaluation Cases

For evaluation of our solution, we apply EUNOMIA to two NoSQL databases, Cassandra (most of the evaluation) and Redis (a single experiment). EUNOMIA is applicable to any NoSQL database that exposes externally controllable performance-sensitive parameters. The first use case is based on real workload traces from the metagenomics analysis pipeline, MG-RAST [8, 38]. MG-RAST is a global-scale metagenomics portal, the largest of its kind, which allows many users to simultaneously upload their metagenomic data (nucleotide or protein sequences) to the repository, apply a pipeline of computationally intensive processes, such as similarity search, 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. 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 evaluation cases span the range of patterns and corresponding predictability of the workloads.

Running multiple Cassandra server instances on the AWS platform, we compare our approach to existing baseline solutions and show that EUNOMIA increases throughput under all dynamic workload patterns and for all types of queries, with no downtime. For example, EUNOMIA achieves 24.5% higher throughput over the default configuration and 21.5% higher throughput over a statically determined idealized optimal configuration in the bus-tracking workload. Similarly, EUNOMIA achieves 38% and 30% higher aggregate throughput over these two baselines in the HPC cluster workload. We also show the performance of the Redis NoSQL database applied to the data analytics workload. With EUNOMIA'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.

To summarize, the main contributions of EUNOMIA 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. EUNOMIA performs cost-benefit analysis to achieve long-horizon optimized throughput for clustered Cas-

sandra instances. It achieves this in the face of dynamic workload changes, including with unpredictable and fast changes to the workload patterns.

3. EUNOMIA 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.

The rest of the paper is organized as follows. Section 2 provides an overview of our solution EUNOMIA. 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 EUNOMIA

Here we give an overview of the workflow and the main components of EUNOMIA. A schematic of the system is shown in Figure 2. We provide details of each step and each component in Section 4.

Periodically, EUNOMIA 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 EUNOMIA 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 EUNOMIA operates. With the predicted workload, EUNOMIA 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 to adapt with any changes to the system such as changes in the application schema.

The **Dynamic Configuration Optimizer** (box 2) generates a time-varying reconfiguration plan for a given lookahead window using a cost-benefit analysis (CBA). The reconfiguration 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 lookahead 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 (Section 4.5). This controller is conceptually centralized but replicated and distributed in implementation using off-the-shelf tools like ZooKeeper. EUNOMIA decides how many instances to switch concurrently such that the cluster always satisfies the user's availability and consistency requirements.

Physically, EUNOMIA is implemented as separate components. The Workload Predictor is located at a point where it can observe the aggregate incident workload such as at a gateway to the database cluster or by asking 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

EUNOMIA accrues benefits in throughput by implementing optimal reconfigurations when the database workload changes, boosting performance over the previous configuration. However, 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 EUNOMIA calculates the costs of implementing a reconfiguration plan by determining the number, duration, and magnitude of degradations. If the overall throughput gain is greater than the cost to reconfigure, then EUNOMIA controller initiates the reconfiguration plan, else it rejects the plan. This insight, and the resulting protocol design to decide *whether* and *when* to reconfigure, are the fundamental contributions of EUNOMIA.

Now we give a specific example of this cost-benefit tradeoff from real MG-RAST workload traces to illustrate this argument. Consider the example shown in Figure 1 where we apply EUNOMIA's reconfiguration plan to a cluster of 2 servers with an availability requirement that at least 1 of 2 be online. The Cassandra cluster starts with a read-heavy workload but with a configuration C_1 (the Cassandra default), that favors a write-heavy workload and is therefore suboptimal. With this configuration, the cluster provides a throughput of $\sim$40,000 ops/s and a tail latency of 102 ms (P99), but a better read-optimized configuration C_2 exists, providing $\sim$50,000 ops/s and a tail latency of 83 ms. The Cassandra cluster is reconfigured to the new C_2 configuration setting, using EUNOMIA'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 EUNOMIA would include it in the plan. Otherwise, the costs will outweigh the benefit, and any solution implemented without the CBA risks degrading 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.*

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 [7]. 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, such as, mining tweets or storing financial data. According to the Cassandra architecture, it caches writes in an in-memory structure called *Memtable* for a certain period of time. Afterward, all Memtables get flushed to their corresponding persistent representation on disk called *SSTables*. Also 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 written to again 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 is essential for Cassandra to achieve better performance for reads, but is also a heavy operation that consumes CPU and memory and can negatively impact the performance for writes if they conflict in time when the compaction is occurring.

The optimal values of these performance-sensitive configuration parameters are dependent on the workload. For example, we find empirically that the size-tiered compaction strategy achieves 44% better performance for write-heavy workloads than the leveled compaction strategy, while the leveled compaction strategy achieves 90% better performance for read-heavy workloads (Figure 4). When the workload changes, the optimal parameters for the new workload will likely be different as well. However, the reconfiguration also incurs two broad types of cost. The first is the cost of data movement across nodes, say one node is being taken out of a cluster. The second is the cost of incrementally switching over the cluster of Cassandra instances to the new configuration. An incremental approach is desired, rather than the naïve one of switching all concurrently, because the latter renders all the data unavailable during reconfiguration. This is exacerbated by the fact that a server-restart is required for reconfiguring most impactful parameters.

Dynamic Workloads in Cassandra

Dynamic workloads in our pipeline have swings in the relative ratio of the different operations on the database. Therefore, EUNOMIA needs to react in an agile manner to such swings. For example, MG-RAST traces show 443 sharp swings (more than 80% change) per 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 swings per day is observed. The static tuners cannot handle such dynamism and cannot even pick a single parameter set that will on 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 EUNOMIA

EUNOMIA seeks to answer the following two broad questions:

1. When should the cluster be reconfigured? This is dependent on the workload in a lookahead period and the decision has to be made based on predicted workload patterns as well as accounting for the costs of the reconfiguration.
2. How should we apply the reconfiguration steps? The goal here is to minimize the transient impact on the performance of the system (i.e. Throughput and Tail latency) and maintain the required data availability.

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 the various components of EUNOMIA that we introduced earlier.

4.1 Workload Description 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:

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

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 rate of operations 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. For example, for MG-RAST, two features captured the workload characteristics: (1) the proportion of reads versus writes, i.e., the read ratio (RR) and (2) the key reuse distance (KRD) triggered by the queries. From the computing environment, we also obtained the number of jobs in the cluster. For the bus-tracking application, we use 5 features: the key reuse distance (KRD), and the proportions of the four operations: reads, writes (inserts), updates, and

scans. Notice that scans (unlike reads) have a condition (i.e. predicate) which requires the DBMS to evaluate that condition against all existing rows. Therefore, scan is a heavier operation than reading individual rows by their keys.

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:

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

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 EUNOMIA, we utilize two types of prediction models. The first type is an ARIMA model, which is more suitable for short lookahead predictions such as MG-RAST. The second type is a Markov Chain model, which is suitable for long lookahead periods such as bus-tracking. Moreover, we also use a deterministic, fully accurate output from a batch scheduler for the HPC data analytics workload, i.e., a perfect f_{pred}. However, more complex estimators, such as neural networks [26, 34], have been used in other contexts and EUNOMIA is modular enough to use any such predictor. After the workload predictor provides EUNOMIA with future workload patterns, EUNOMIA will use a static configuration tuner, RAFIKI in our current design, plus the CBA model to apply its long-horizon optimization technique.

4.2 Adapting a Static Configuration Tuner for EUNOMIA

EUNOMIA uses a static configuration tuner (RAFIKI), designed to work with Cassandra, to output the best configuration for any given *static* workload. RAFIKI uses ANOVA in order to estimate the importance of each parameter. It replays the given workload and changes one parameter at a time while all other parameters are set to the default value. In each replay, it measures the change in performance (both increase and decrease) and 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 7 highest performance-impactful 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. This is because a change to the compaction method causes every Cassandra instance to launch a full compaction operation in the background, reading all its SSTables and re-writing them to the disk in a new format. However, due to inter-dependability 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 ((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, EUNOMIA needs the static tuner to provide an estimate of throughput for both the optimal and the current configuration for any workload:

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

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 can achieve efficient load balancing across multiple instances whereby each contributes approximately equally to the overall system throughput [30, 17]. 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:

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

In general, $\boldsymbol{C}_{\text{opt}}$ can be unique for each server, but in EUNOMIA, 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 EUNOMIA 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 $i+1$. This simplification, together with the load balancing feature of a Cassandra cluster, allows for the pragmatic simplification. 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.

4.3 Dynamic Configuration Optimization

EUNOMIA'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 EUNOMIA 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., row cache size. If we take an underestimate of each parameter being binary, then the size of the search space becomes $2^{7\times30} = 1.6\times10^{+63}$ 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 $C(T_L,M)\times|\boldsymbol{C}_{\text{sys}}|$. 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:

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

where t_k is a point in time, $k\in[0,T_L]$ 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 EUNOMIA is to select the best configurations for some period of optimization T_L:

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

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. 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. Likewise, the cost summed up over the time window is the loss in throughput incurred during the transient period of reconfiguration.

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

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 $\boldsymbol{C}^{\Delta}_{\text{sys}}$.

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

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. 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 and unadjusted at the old configuration. EUNOMIA is general enough to work with any reconfiguration cost, including different costs for different parameters, and these can be fed into the loss function (Eq. 8).

The objective is to maximize the time-integrated gain (benefit – cost) of the reconfiguration from Eq. (6) and EUNOMIA is responsible for determining the optimal reconfiguration plan. 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 Section 4.5 how R is calculated), at least $\frac{T_r\cdot N_s}{R}$ 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 $\boldsymbol{C}^{\Delta}_{\text{sys}}$ 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.

The value for T_L should be based on the confidence of the workload predictor. A value that is too large will cause EUNOMIA to include decision points with high errors, and a value that is too small will cause EUNOMIA to take almost greedy decisions. 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 EUNOMIA. 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. In our use case, speedy decisions are needed as well since the optimizer executes in the critical path. The GA with its tunable completion criterion satisfies this purpose too.

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. EUNOMIA queries token assignment information 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, EUNOMIA insures that N_{minCL} nodes are operational at any point of time. Therefore, EUNOMIA 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. A token represents a range of hash values of the primary keys which the node is responsible for. 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 EUNOMIA cannot reconfigure the system, without harming data availability and we use the EUNOMIA-AGGRESSIVE variant in that cae. However, we expect most systems with high consistency requirements to follow a read/write quorum with CL $= \lceil \frac{RF}{2} \rceil$ [20]. Note that EUNOMIA reduces the number of available data replicas (from RF to CL) during the transient reconfiguration periods by taking R servers offline at a time. 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, EUNOMIA performs the following steps, R server instances at a time:

1. Drain: Before shutting down a Cassandra instance, we flush the entire Memtable to disk by using Cassandra's drain 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 of all the configuration parameters that need changing.
4. Restart: Restart the Cassandra process on the same node.
5. Sync: EUNOMIA 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 (denoted as *max_hint_window_in_ms*). These cached writes are resent to the nodes when they rejoin the cluster. EUNOMIA achieves $T_r \ll$ *max_hint_window_in_ms*, which is critical because if the timeout kicks in, no more writes are cached and a manual repair is needed to bring back the node's data consistency.

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 the draining option rather than the commit log replay in step 1, reducing pressure on the disk.

5 Dataset and Implementation

5.1 MG-RAST Workload

For short-horizon reconfiguration plans, we use real workload traces from MG-RAST. As the amount of data stored by MG-RAST has increased beyond the limits of traditional SQL stores (150 Tera base pairs or roughly 23 PB as of August 2018), it relies on a distributed NoSQL Cassandra database. 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) Majority (i.e., more than 80%) of the switches are abrupt, from RR=0 to RR=1 or vice versa. (iii) KRD (key reuse distance) is very large. (iv) No daily or weekly workload pattern is discernible, as expected for a globally used cyberinfrastructure.

5.2 Bus Tracking Application Workload

We use real workload traces from a bus-tracking mobile application called **Tiramisu** [34]. 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, scan, update, and insert. The scan operation means reading of all the records in the database, which is much heavier than the other operations. The meanings of the other operations are self evident. A sample of the traces is publicly available [33]. 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. (ii) 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. Moreover, as the workload has a daily pattern, our prediction model can predict much longer lookahead periods (e.g., 5 hours) with high accuracy compared to the MG-RAST case.

5.3 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 EUNOMIA with a job scheduler (like PBS), 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 [18] and as in that paper, we achieve high accuracy in predicting the execution times of the jobs. Thus, EUNOMIA 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, 18]) 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 added to the set of active jobs. We vary the level of concurrency and have an equal mix of the two types of jobs and monitor the aggregate workload. Figure 3 shows the synthetic traces for three job sizes. With increase in concurrency, the aggregate pattern becomes smoother and the latency of individual jobs increases.

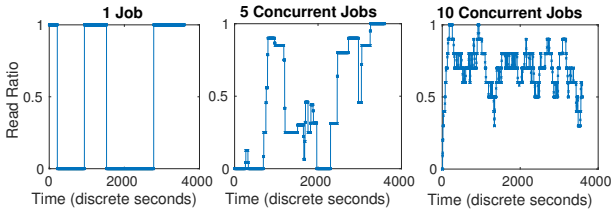

Figure 3: Simulated Workload patterns for 1, 5, and 10 concurrent jobs

6 Experimental Results

Here we evaluate the performance of EUNOMIA under different experimental conditions for the 3 applications. In EUNOMIA, 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 simple queries per second. We evaluate EUNOMIA on Amazon EC2 using instances of size M4.xlarge with 4 vCPU's and 16 GB of RAM for all Cassandra servers and workload drivers and provisioned IOPS (SSD) EBS for storage and high network bandwidth ($\sim$ 0.74 Gbits/s). Each node is loaded with 6 GB of data initially. We use multiple concurrent clients to saturate the database servers and aggregate the throughput observed by every client for the output metric, the system-level throughput.

6.1 Baseline Comparisons

We compare the performance of EUNOMIA to baseline configurations (1-5). We also consider 2 variants of EUNOMIA (6-7).

(1) Default: The default configuration that ships with Cassandra. This favors write-heavy workloads.

(2) Static Optimized: Here, the static tuner (RAFIKI) is 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. 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 ScyllaDB in its vanilla

form. ScyllaDB is touted to be a much faster (10X or higher) drop-in replacement to Cassandra [43]. This stands in for other self-tuning databases [25].

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

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

(7) EUNOMIA**:** This is our complete system.

(8) EUNOMIA-AGGRESSIVE**:** This is a variant from our solution which prefers faster reconfiguration over data availability. EUNOMIA-AGGRESSIVE violates the availability requirement by reconfiguring all servers at the same time. However, it does not execute reconfiguration every time the workload changes such as Naïve. We use EUNOMIA-AGGRESSIVE whenever RF=CL as in this case, each node has data that are not replicated any where else and hence reconfiguration cannot happen without availability degradation.

6.2 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 EUNOMIA to take single-step reconfiguration plans and limits its benefit over a static optimized approach (Figure 4). In contrast, when job characteristics can be predicted well (bus tracking and data analytics applications), EUNOMIA achieves significant benefit over both default and static optimized cases (Figures 5 and 6). This benefit stays even when there is significant uncertainty in predicting the exact job characteristics (Figure 10). 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 EUNOMIA, atop Cassandra (Figures 4 and 6). Third, as the replication factor increases while the number of server are fixed, the reconfiguration time of EUNOMIA decreases, thus improving its benefit (Figure 8). Contrarily, as CL increases, the benefit of EUNOMIA shrinks (Figure 8).Finally, EUNOMIA 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 11).

6.3 Experiment 1: MG-RAST Workload

We present our experimental evaluation with the workload traces (queries and data records) from 20 test days of MG-RAST data. To zoom into the effect of EUNOMIA 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.

Table 1: RMSE of the MG-RAST workload prediction with Markov Chain. We use the three representative workloads corresponding to different frequencies of workload switches.

MC-Order	First-Order		Second-Order		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%	-

Prediction Model Training: 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 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 [31], 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 notice that the accuracy is high 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 $T_L = 5$ minutes in EUNOMIA.

Performance Comparison:

Now we show the performance of EUNOMIA with respect to the four workload categories. We first present the result with the smallest possible number of server instances, 4, run with MG-RAST's parameters RF=3 and CL=1 [28]. 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 4 shows the performance improvements for our test cases.

From Figure 4, we see that EUNOMIA always outperforms naïve in total ops/s (average of 31.4%) and individually in read (31.1%) and write (33.5%) ops/s. EUNOMIA also outperforms the default for the slow and the mid frequency cases, while it slightly under performs in the fast frequency case with average improvement across the 4 categories of 20.4%. The underperformance for the fast case is due to increased prediction error. The static optimized configuration (which for this workload favors read-heavy pattern) has a

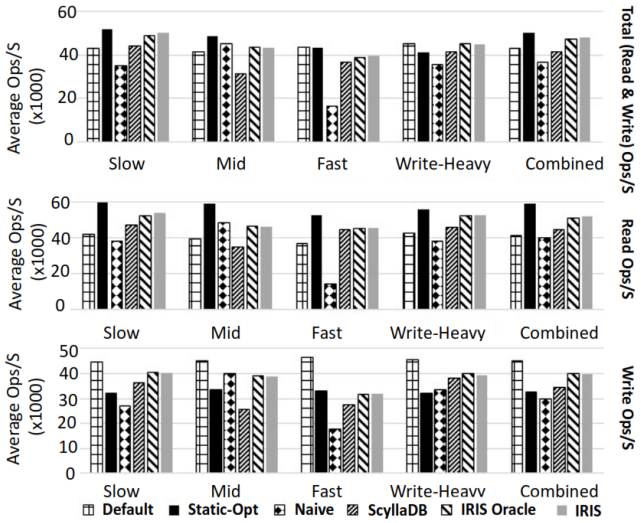

Figure 4: Improvement for four different 30-minute test windows from MG-RAST real traces over the baseline solutions.

slightly higher throughput over EUNOMIA by 6.3%. This is because the majority of the selected samples are read periods (RR=1), which hides the gain that EUNOMIA achieves for write periods. However, we see that with respect to write operations, EUNOMIA achieves 17.6% higher throughput than the static optimized configuration. Increased write throughput is critical for MG-RAST to support the bursts of intense and voluminous writes. This avoids unacceptable queuing of writes, which can create bottlenecks for subsequent jobs that rely on the written shared dataset.

We observe that EUNOMIA performs similar to EUNOMIA w/ Oracle in the slow and mid scenarios, which shows the minor impact of the workload predictor. However, in the fast scenario, EUNOMIA shows a loss of 8% in comparison to both the default and static optimized configurations due to inefficient reconfigurations. Naïve reconfiguration has an even higher loss compared to default: 61.8%.

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). Since the throughputs achieved by Cassandra-default and ScyllaDB are different under different workload mixes, the reader should first calibrate herself by looking at the "Default" and "ScyllaDB" bars. ScyllaDB is claimed by its developers to outperform Cassandra in all workload mixes by an impressive 10X [43]. 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, EUNOMIA 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 EUNOMIA for the *entire range* of workload mixes and not have to transition to ScyllaDB.

6.4 Experiment 2: Bus Tracking Application Workload

We evaluate the performance of EUNOMIA using the bus-tracking application traces. Figure 5 shows the gain of using EUNOMIA over Default, Static-Opt, and Naïve 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 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. EUNOMIA outperforms default configuration by 24.5%, Static-Opt by 21.5%, and Naïve by 28.5%. The gains are higher because EUNOMIA 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 the case with MG-RAST. This is because the frequency of workload changes is lower here. However, the naïve solution still renders the data unavailable during the reconfiguration period.

Prediction Model Training: 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 2). 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 first-order and the second-order Markov Chain models to predict the workloads. We notice that expectedly, increasing the order of the Markov Chain increases the number of states of the model. We predict the future workload for different lookahead periods and calculate the RMSE between the predicted and the actual states. 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 to predict the workload or uses the time-stamp and day of the week. The predicted workload is the average of the mixtures observed at the pre-

Table 2: RMSE of the bus-tracking workload prediction with Markov Chain. We notice that second-order Markov Chain model achieves better performance with longer lookahead times. We use 5h in our experiments.

Lookahead	15m	1h	2h	5h	10h	# States
First-Order	6.9%	7.4%	7.9%	10%	13.67%	117
Second-Order	7.12%	7.4%	7.35%	7.5%	8%	647

vious 10 instances. These simple predictors have unacceptably high RMSE of 31.4% and 24.0%. Therefore, although the workload has a somewhat predictable pattern, we cannot generate the optimal plan once and use it for all subsequent days.

6.5 Experiment 3: HPC Data Analytics Workload

In this set of experiments we evaluate the performance of EUNOMIA using HPC data analytics workload patterns described in Section 5.3. Here our lookahead is the size of the job queue, which is conservatively taken as 1 hour. Figure 6 shows the result for the three levels of concurrency (1, 5, and 10 jobs). We see that EUNOMIA outperforms the default for all the three cases, with average improvement of 30%. In comparison with static optimized configuration (which is a different configuration in each of the three cases), we note that EUNOMIA 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, EUNOMIA switches only once: from the default configuration to the same configuration as the static optimized. We notice that EUNOMIA achieves within 9.5% of the theoretical best performance for all three cases. Finally, we notice that EUNOMIA achieves significantly better performance over Naïve by 27%, 13%, and 122% for the three cases, while the naïve approach can degrade the performance even lower than the default by 32.9% (10 concurrent jobs). In comparison with ScyllaDB, EUNOMIA is able to reconfigure Cassandra at runtime and turn out a performance benefit over ScyllaDB by 17.4% on average, which leads to a similar conclusion as in MG-RAST about the continued use of Cassandra.

6.6 Experiment 4: Scale-Out

Figure 7 shows the behavior of EUNOMIA with increasing scale using the data analytics workload. We show the comparison between EUNOMIA and Static-Opt (all other baselines performed worse the 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 (Section 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 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 1 Gbit/s in the case of M4.xlarge). The results show that EUNOMIA'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 EUNOMIA's plan (while Static-Opt uses size-tiered compaction), and hence leads to greater performance difference for reads.

6.7 Experiment 5: Varying RF and CL

We also evaluate the impact of applying EUNOMIA to clusters with different RF (Replication Factor) and CL (Consistency Level). 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 8 (CL quorum implies CL = $\lceil RF/2 \rceil$). We notice that EUNOMIA continues to achieve better performance than all 3 static baselines for all RF, CL values. For RF=1, CL=1, we use EUNOMIA-AGGRESSIVE because when RF=CL, we cannot reconfigure the cluster without degrading availability. The key observation is that EUNOMIA'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 EUNOMIA needs to perform the reconfiguration is inversely proportional to RF-CL as discussed in Sec. 4.5. This allows EUNOMIA 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. This is because 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 for both reads and writes.

6.8 Experiment 6: Greater Data Volume

We evaluate EUNOMIA when the data volume per node increases. We use the HPC workload with 5 concurrent jobs. We vary the amount of data loaded initially into each node (in a cluster of 4 nodes) and measure the gain over static optimized in Figure 9. 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 EUNOMIA'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 because the static optimized configuration for this workload uses the Size-Tiered compaction, whereas the configurations applied by EUNOMIA had the compaction method switched to Leveled compaction, which 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.

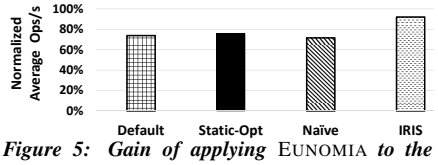

Figure 5: Gain of applying EUNOMIA *to the bus-tracking application. We use 8 Cassandra servers with RF=3, CL=1. A 100% on the Y-axis represents the theoretical best performance.* EUNOMIA *achieves improvements of 24.5% over default, 21.5% over Static-Opt, and 28.5% over naïve.*

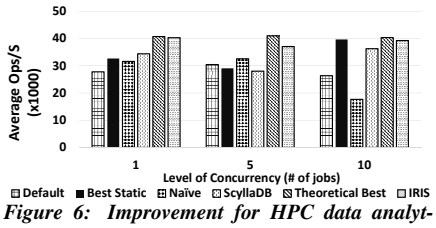

Figure 6: Improvement for HPC data analytics workload with different levels of concurrency. We notice that EUNOMIA *achieves higher average throughput over all baselines*

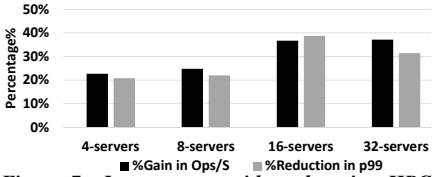

Figure 7: Improvement with scale using HPC workload with 5 jobs with RF=3 and CL=1. EUNOMIA *provides consistent gains across scale because the cost of reconfiguration does not change with scale (for the same RF and CL). The higher gains for 16 and 32 servers is due to the use of M5 instances, which can be exploited by* EUNOMIA *better than Static-Opt.*

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.

6.9 Experiment 7: Noisy Workload Predictions

We show how sensitive EUNOMIA 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 10, we see that adding noise to EUNOMIA slightly reduces its performance. However, such noise will not cause significant changes to EUNOMIA's optimal reconfiguration plan. This is because EUNOMIA 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 EUNOMIA to achieve significant improvements for long-term predictions even with high noise levels.

6.10 Experiment 8: Redis Case Study

We now show a case study with a different and popular NoSQL database called Redis. We wanted to evaluate the generality of our approach and Redis has a long-standing pain point in setting a performance-critical parameter against changing workloads. Redis is an in-memory data store which can be used either as a stand-alone database or a cache for another database. By default, Redis stores all keys and values in memory; 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* [29]. 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 [44, 21, 46]. We use EUNOMIA to tune this feature and compare the performance to three baselines: (1) Redis V2.4 with VM-disabled (Default configuration) (released Aug 2012), (2) Redis V2.4 with VM-enabled, (3) Redis V4 with default configuration (no VM support, latest version, released July 2017).

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 online and a server restart is required. We use the HPC analytics jobs workload with only one active job. Applied jobs vary with respect to their sizes (0.5M, 1M, 2M) and also their request distribution (i.e., Uniform vs Zipfian). We use an AWS server of type C3.Large with 2 vCPUs and 3.75GB RAM. 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. From Figure 11 we see that for all record sizes and request distributions, EUNOMIA 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, EUNOMIA, by automatically selecting the right parameters for changing workloads, can achieve the best of both worlds—fast in-memory database, and ability to leverage disk in case of spillover.

7 Related Work

We categorize the related work under three major heads.

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 [12, 6, 19, 16, 52]. Among these, Morphus [19] targets MongoDB, a noSQL DBMS but cannot handle Cassandra due to its peer-to-peer topology and sharding. In general, data partitions appear more suitable to online changes than updating configuration parameters. Tuba [5] reconfigures geo-replicated key-value stores by changing locations of primary and sec-

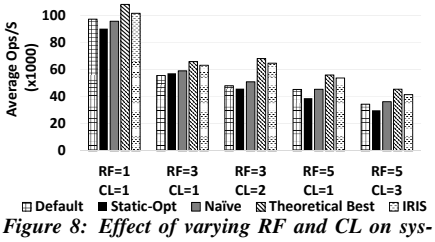

Figure 8: Effect of varying RF and CL on system throughput. We use a cluster of 8 nodes and compare the performance of EUNOMIA *to Default, Static-Opt, and naïve.* EUNOMIA *outperforms the static baselines and approaches the theoretical best as RF–CL increases.*

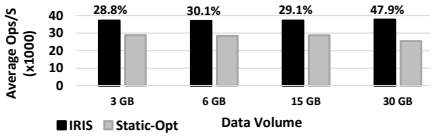

Figure 9: Effect on increasing data volume per node. We use a cluster of 4 servers and compare the performance to the static optimized. The results show that EUNOMIA*'s gain is consistent with increasing data volumes per node.*

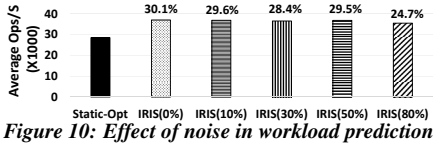

Figure 10: Effect of noise in workload prediction on the performance of EUNOMIA *on the data analytics workload with level of concurrency = 5. The percentage represents the amount of noise added to the predicted workload pattern.*

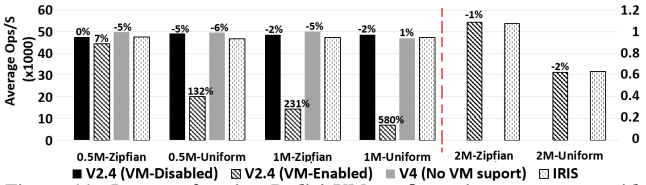

Figure 11: Impact of tuning Redis' VM configuration parameters with EUNOMIA *with the data analytics workload. The percentage improvement of* EUNOMIA *is shown on each bar and the right Y-axis is for the 2M jobs. A missing bar represents a failed job. We notice that the current Redis fails for large workloads (2M), while* EUNOMIA *achieves the best of both worlds*

ondary replicas to improve overall utility of the storage system. Rocksteady [27] 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 [11, 9, 55, 22, 10, 49, 2, 41, 42]. Another body of work improves performance for static workloads by finding correct settings for DBMS performance knobs [15, 14, 35, 54, 50] as discussed before. EUNOMIA 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 [24, 3] and cloud platforms [32, 53, 37, 36]. 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 [32, 4]. BerkeleyDB [40] models probabilistic dependencies between configuration parameters. A recent work, *SmartConf* [51] 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.

Reconfiguration in OS and other single-node stack. There has been long-past work on this topic [45, 23, 39] developing modularization techniques for OS for it to be live upgraded, without causing application downtime. The goals were to update code to adapt to changing workloads [45] or to perform maintenance tasks, e.g., performance debugging [45, 39].

8 Conclusion

When faced with dynamic and fast-changing workloads, NoSQL databases have to be tuned for achieving the highest throughput. According to current practice, clusters of Cassandra server instances are shut down for reconfiguration and then restarted with the new configuration, thus degrading data availability. Current static tuners can provide the optimal configuration for any given workload. However, they cannot determine whether and when to perform a configuration switch to maximize benefit over a future time horizon with changing workloads and they cannot perform online reconfiguration. We presented EUNOMIA 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 EUNOMIA 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 several open challenges. How to do anticipatory configuration changes for future workload patterns? How to handle heterogeneity in the database cluster, i.e., one where each server instance may have its own configuration and may contribute differently to the overall system throughput? Finally, how should EUNOMIA factor in configuration parameters whose changes take effect only after a time lag?

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 auto-

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

[8] CHATERJI, S., KOO, J., LI, N., MEYER, F., GRAMA, A., BAGCHI, S., AND CHATERJI, S. Federation in genomics pipelines: techniques and challenges. *Briefings in Bioinformatics 102* (2017).

[9] 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.

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

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

[12] 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).

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

[14] 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).

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

[16] 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).

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

[18] 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.

[19] 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).

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

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

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

[23] HICKS, M., AND NETTLES, S. Dynamic software updating. *ACM Transactions on Programming Languages and Systems (TOPLAS) 27*, 6 (2005), 1049–1096.

[24] 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.

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

[26] 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.

[27] 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.

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

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

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

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

[32] 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.

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

[34] 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.

[35] 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.

[36] 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).

[37] 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).

[38] 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).

[39] OBERTHÜR, S., BÖKE, C., AND GRIESE, B. Dynamic online reconfiguration for customizable and self-optimizing operating systems. In *Proceedings of the 5th ACM international conference on Embedded software* (2005), ACM, pp. 335–338.

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

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

[42] 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).

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

[44] SERVERFAULT. Should I use vm or set the maxmemory. `https://serverfault.com/questions/432810/should-i-use-vm-or-set-the-maxmemory-with-red` September 2012.

[45] SOULES, C. A., APPAVOO, J., HUI, K., WISNIEWSKI, R. W., DA SILVA, D., GANGER, G. R., KRIEGER, O., STUMM, M., AUSLANDER, M. A., OSTROWSKI, M., ET AL. System support for online reconfiguration. In *USENIX Annual Technical Conference* (2003).

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

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

[48] 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).

[49] 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).

[50] 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.

[51] 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).

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

[53] 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.

[54] 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).

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