IRIS: Online Reconfiguration of Distributed NoSQL Databases for Dynamic Workloads

Ashraf Mahgoub[(a)], Paul Wood[(a)], Subrata Mitra[(b)], Folker Meyer[(c)], Somali Chaterji[(a)], Saurabh Bagchi[(a)]

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 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 following two fundamental limitations. First, they do not account for performance degradation due to database restarting (often needed to apply the new configurations), and second, they overlook the application's availability requirements during reconfiguration. Our solution, IRIS, addresses both these shortcomings and we demonstrate its effectiveness for a multi-tenant, global-scale metagenomics pipeline called MG-RAST and an HPC data analytics job queue, both of which have dynamically changing workloads.We compare the benefit of IRIS in throughput over the default, a static configuration, and a theoretically ideal solution.

1 Introduction

Automatically tuning database management systems (DBMSs) is challenging due to their underlying plethora of performance-related parameters and that too with complex interdependencies among subsets of these parameters. For example, MySQL has 100+ and Cassandra has 50+ tuning parameters, and different parameter combinations can affect performance in different ways. Several prior works like Rafiki [32], OtterTune [45], BestConfig [50], and others [15, 43, 42], 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 do not perform a cost-benefit analysis (CBA) to assess *when* a configuration switch will be globally beneficial, over a time horizon, *and* cannot gracefully reconfigure a clustered set of database server instances while maintaining user visible availability.

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 and there is degraded throughput as a newly resurrected server instance is updated with missed data records. In the case of dynamic workloads, the new workload may not last long enough for the reconfiguration cost to be recouped over a reasonably long time horizon. *Fundamentally, this is where the drawback of prior approaches lies—they are either silent on when to reconfigure or perform a naïve reconfiguration whenever the workload changes.*

We posit that an accurate CBA is crucial for determining whether or not to reconfigure the database. We show that a naïve reconfiguration, which is oblivious to the reconfiguration cost, actually *degrades* the performance for dynamic workloads in general and specifically for metagenomics workloads from the MG-RAST production system [48], relative to the state-of-practice of using a static tuner (Figure 1). For example, during periods of high dynamism in the read-write switches in MG-RAST, naïve reconfiguration degrades performance by a substantial 61.8%.

Our System: IRIS

We develop an online reconfiguration system—IRIS—for a NoSQL Cassandra cluster comprising of multiple server instances. IRIS actively extracts information about current and future workloads from a job scheduler or predictor and determines a long-horizon optimal reconfiguration plan through our proposed CBA scheme. IRIS is applicable to dynamic workloads and works across a range of workload changing patterns. When the workload changes, IRIS interacts with any existing static configuration tuner (we use RAFIKI in our work because it is

already engineered to work with Cassandra), to quickly provide the optimal *point configurations* for the new workload and the estimated benefit from this new configuration. IRIS takes into consideration the possibility that the new workload may be a short-term shift and the estimated benefit from a reconfiguration might not be larger than the cost incurred for such a change. Only if the CBA indicates that the benefit outweighs the cost, does IRIS initiate a distributed protocol to reconfigure the Cassandra cluster. IRIS deals with different replication factors (RF) and consistency level (CL) requirements from the cluster during the online reconfiguration and ensures that the data remains continuously available through the reconfiguration process, while obeying the required CL. It does this by controlling how many Cassandra server instances should be concurrently reconfigured.

Evaluation Cases: MG-RAST and Data Analytics Workloads

For evaluation of our solution, we use real workload traces from the metagenomics analysis pipeline, MG-RAST [48, 9, 35]. MG-RAST provides software-as-a-service, accessible through a RESTful API, 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 preferentially commit the results back to the repository for shared use. As the amount of data stored by MG-RAST has increased beyond the limits of traditional SQL stores (131 Tera base pairs or roughly 20 PB as of January 2018), it relies on a distributed NoSQL Cassandra database. 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 lookahead is possible. The second 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 (using say [19]) over long time horizons (order of an hour) by observing the jobs in the queue and leveraging the fact of recurring job patterns. In principle, IRIS can be used with other DBMS engines, with little (*e.g.*, ScyllaDB, a drop-in Cassandra replacement) to large (*e.g.*, in-memory databases) engineering effort, provided that the system has workload-sensitive parameters that can be updated externally.

To summarize, the main contributions of IRIS are:

1. We show that state-of-the-art static tuners when applied to dynamic workloads provide suboptimal configurations and degrade data availability.
2. IRIS performs cost-benefit analysis (CBA) to achieve long-horizon optimized throughput for clustered Cassandra instances.
3. IRIS executes a distributed protocol to gracefully switch over the cluster to the new configuration while respecting the data consistency guarantees and keeping data continuously available to users.
4. We evaluate IRIS with 80 days of real MG-RAST's workload traces, running multiple Cassandra server instances on the AWS Amazon cloud platform. We compare our approach to existing baseline solutions and show that IRIS optimizes performance with no downtime. For example, compared to the statically determined optimal configuration, IRIS achieves 17.6% higher throughput during the write bursts, typical in MG-RAST, and 25.7% higher aggregate throughput in the HPC cluster case.

The rest of the paper is organized as follows. In Section 2, we provide an overview of our solution approach IRIS. We provide a background on Cassandra and its sensitivity to configuration parameters and on static configuration tuners in Section 3. Then, 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 IRIS

Here we give an overview of the workflow and the main components of IRIS. We provide details of each step and each component in Section 4. A schematic of the system is shown in Figure 2. IRIS is used to improve the performance of Cassandra, by selecting the performance-sensitive parameters as the workload changes, and reconfiguring the cluster, only when deemed opportune using our CBA modeling.

Periodically, IRIS queries the **Workload Predictor** (box 1 in figure) to determine if any future workload changes exist that may merit a reconfiguration. The prediction model, provided by the database administrator (DBA), is initially trained from representative workload traces from prior runs of the application and incrementally updated with additional data as IRIS operates. With the predicted workload, IRIS queries a static configuration tuner that provides the optimal configuration for a single point in time for the predicted workload. This module may also require some initial training data from the system. The **Dynamic Configuration Optimizer** (box 2) generates a time-varying reconfiguration plan for a given lookahead time window by combining the static, point solution information with the estimated, time-varying workload information. The reconfiguration plan gives both the time points when reconfiguration should be initiated *and* the new configuration parameters at each such time point. The **Controller** (box 3) initiates a distributed protocol to "gracefully" switch the cluster to new configurations in the reconfiguration plan (Section 4.5). IRIS decides how many instances to switch at a time such that data items *always* satisfy the

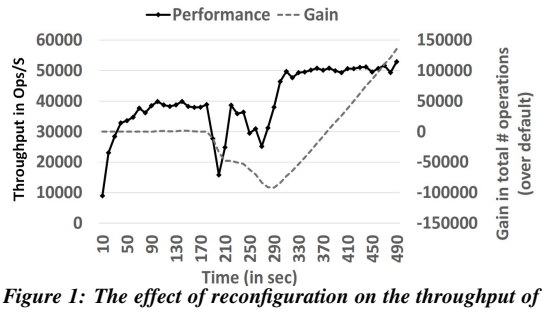

Figure 1: The effect of reconfiguration on the throughput of the system. If the new workload persists for greater than a certain duration (where the gain curve becomes positive), then reconfiguration should be done, else it is better to stay with the earlier configuration.

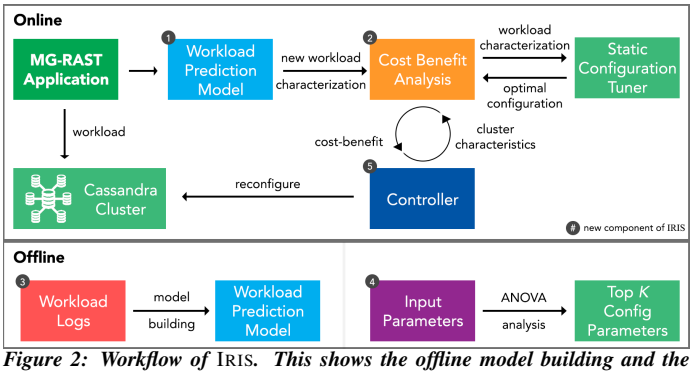

Figure 2: Workflow of IRIS. *This shows the offline model building and the online operation, as well as the new components that are introduced in our system. It also shows the interactions with the Cassandra cluster and a static configuration tuner, which comes from prior work.*

user's availability and consistency requirements.

Physically, IRIS is implemented as separate components. The Workload Predictor is located at a point where it can observe the aggregate incident workload, measured either through a gateway or by querying each instance for loadings. 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. The reconfiguration plan is re-calculated whenever the workload predictor predicts a change in the future workload, but this is practically limited by the computational capacity of the workload predictor and the time constants of the dynamic configuration optimizer and the controller.

Cost-Benefit Analysis in the Reconfiguration Plan

IRIS accrues benefits in throughput by implementing optimal reconfigurations when the database workload changes, boosting performance over static configurations. Each reconfiguration has a cost, however, due to changing parameters that require restarting or otherwise degrading the database services, *e.g.*, by flushing the cache. The CBA in IRIS calculates the costs of implementing a reconfiguration plan by determining the number, duration, and magnitude of degradations. If the overall throughput gains are greater than the cost to reconfigure, then IRIS accepts the reconfiguration plan (which is executed by the controller), else it rejects the plan. This insight, and the resulting protocol design to decide *whether and when* to reconfigure, is a fundamental contribution of IRIS. Prior works in static configuration tuning [32, 45, 50, 15, 43, 42] answer the question of *what* configuration parameter set to use.

Now we give a specific example of this cost-benefit trade-off from real MG-RAST workload traces to illustrate this argument. Consider the example shown in Figure 1 where we apply IRIS'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 ∼40,000 ops/s, but a better read-heavy configuration C_2 exists, providing ∼50,000 ops/s. The Cassandra cluster is reconfigured to the new C_2 configuration setting, using IRIS'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. The two dips at 200 and 270 seconds correspond to the two server instances being reconfigured serially. We plot, using the dashed line, the gain (benefit minus cost) over time and see that there is a crossover point for the duration of the new workload pattern. If the predicted workload pattern lasts longer than this threshold (190 seconds in our example), there is a gain from the reconfiguration and the CBA would approve 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

Data Processing Model: In IRIS, we use a simple query model sufficient to support our data processing applications. The computation is separate logically from the database and the latter supports the standard operations of select, insert, delete, and update. This is typical for NoSQL databases and is in contrast to complex analytics queries that are supported by more complex database engines. From this perspective, our throughput is defined as the number of simple queries per second. Our target applications are scheduled on compute clusters using batch job scheduling systems, such as PBS (Portable Batch System) or Torque [24], and the computation gives

rise to the database operations.

Dynamic Workloads in Cassandra; Dynamic workloads in our pipeline have large swings in the relative ratio of reads to writes, such as a phase of data upload followed by its read-based processing and then subsequent result upload. The internals of Cassandra respond to these types of workloads with algorithms that are themselves tunable and replaceable. For example, reads and writes may be cached in RAM with different cache sizes, and the method by which data is compacted on the disk may be altered ("Leveled" versus "Size-Tiered" versus "Date-Tiered" [25, 8]). When the workload is static, the performance-sensitive parameters and their impact have been studied and described in several prior works in [32, 45, 15, 43, 42], etc. For example, it has been found that the best compaction strategy for a read-heavy workload is leveled compaction, which performs poorly for write-heavy workloads. The size-tiered compaction strategy is recommended for the write-heavy workloads [17]. In the dynamic case, a transition from read-heavy to write-heavy may occur after some time, and thus the optimal compaction strategy would also change. Therefore, it would be impossible to find a static configuration that would be optimal for every point in time for a dynamic workload. IRIS is designed to consider the benefit of switching parameters while accounting for the costs associated with each update.

4 Design of IRIS

IRIS seeks to answer the following two questions:

1. Which configurations to be used in the cluster and when to switch? This is dependent on the workload in a lookahead period and the decision has to be made based on predicted workload patterns.
2. How should we apply the reconfiguration steps? The goal here is to minimize the transient impact on the throughput of the system and maintain the required data RF and CL.

The answer to the first question leads to what we call a *reconfiguration plan* (Section 4.3). The answer to the second question is given by our distributed protocol that reconfigures the various server instances in rounds (Section 4.5). Next, we describe the various components of IRIS 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 expressed by:

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

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.

To forecast the workload, we discretized time into sliced T_d durations ($= 30s$ in our experiments) to bound the memory and the computational 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 IRIS, we utilize an ARIMA model in one experiment and a deterministic output from a batch scheduler in another, *i.e.*, a perfect f_{pred}. However, more complex estimators, such as neural networks [27], have been used in other contexts and IRIS is modular enough to use any such predictor. After the workload predictor provides IRIS with future workload patterns, IRIS 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 IRIS

IRIS uses a static configuration tuner (RAFIKI), designed to work with Cassandra, to output the best configuration for any given workload. RAFIKI selects the top-k parameters in its configuration optimization method, which is in turn determined by a significant drop-off in the importance score ([32] Section 3.4). In MG-RAST, the cut-off is at 7 parameters, namely: (1) Compaction method (2) Memory-table flush writes (3) Memory-table clean-up threshold (4) Trickle fsync (5) Row cache size (6) Concurrent writers (7) Memory heap space. These parameters vary with respect to how fast they can impact the performance. For example, the compaction method will cause 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. This is a compute-intensive process and will degrade performance for a long time horizon. The impact of other 6 parameters is observed immediately upon server restart (changing row cache size does not even need a server restart). We modify RAFIKI to find the best values of these 6 parameters, while fixing the compaction method to the best static value across all future workloads.

The database system has a set of critical performance-determining configuration parameters $\boldsymbol{C} = \{c_1, c_2, ..., c_n\}$

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, IRIS needs the static tuner to provide an estimate of throughput for both the optimal and the current configuration for any workload that is seen in the system:

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

where H_{sys} is the throughput of the cluster of servers (as observed by the client) 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. Cassandra can achieve efficient load balancing across multiple instances whereby each contributes approximately equally to the overall system throughput [28, 18]. Thus, we define a single server average performance as:

$$H_i = \frac{H_{\text{sys}}}{N_s} \tag{4}$$

We argue that parameters such as RF and CL are selected by the application owner based on her organization's availability and consistency requirements. These should *not* be changed by any configuration engine and IRIS does not vary these either but can work under different RF/CL constraints, *e.g.*, to decide how many servers to reconfigure at a time (Section 4.5). However, if the application owner changes these parameters mid-stream, the static tuner in IRIS must itself adjust to such a change.

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

$$C_{\text{opt}}(\boldsymbol{W}(t)) = \arg\max_{\boldsymbol{C}_{\text{sys}}} H_{\text{sys}} = \arg\max_{\boldsymbol{C}_{\text{sys}}} f_{\text{ops}}(\boldsymbol{W}(t), \boldsymbol{C}_{\text{sys}}) \tag{5}$$

In general, $\boldsymbol{C}_{\text{opt}}$ can be unique for each server, but in IRIS, it is same across all servers, given our atomic reconfiguration policy in which all servers are run with the same configuration and all servers must be reconfigured in round i prior to reconfiguration of round $i+1$.The optimization in Eq. (5) can be solved using genetic algorithms (*e.g.*, RAFIKI) or nearest-neighbor interpolation for an unseen workload (*e.g.*, OtterTune or iTuned).

4.3 Dynamic Configuration Optimization

IRIS'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 the lookahead T_L. In this section, we describe the mechanism that IRIS uses for CBA modeling to construct the best reconfiguration plan for evolving workloads. Such analysis prevents naïve reconfiguration based solely on improvement due to a new configuration suggested by a static tuner.

In general, finding solutions for $\boldsymbol{C}^T_{\text{sys}}$ can become impractical since the possible parameter space for $\boldsymbol{C}$ is large and the search space increases linearly with T_L. For example, if $T_d = 30$ s and $T_L = 30$ m, then there are $\frac{T_L}{T_d} = 60$ decision points. If there are only two possible configurations, we have 2^{60} choices about what configuration to choose and when. In reality, the configuration space is larger, *e.g.*, for MG-RAST, we used 7 parameters, some of which are continuous, *e.g.*, row cache size. If we take an underestimate of each parameter being binary, then the search space will have $2^{7\times 60}$ 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 M points to switch out of all the T_L time points and at each point there are $|\boldsymbol{C}_{\text{sys}}|$ possible configuration options. We define the reconfiguration plan as:

$$\boldsymbol{C}^{\Delta}_{\text{sys}} = [\boldsymbol{T} = \{t_1, t_2, ..., t_M\}, \boldsymbol{C} = \{C_1, C_2, ..., C_M\}] \tag{6}$$

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

$$(\boldsymbol{C}^{\Delta}_{\text{sys}})^{\text{opt}} = \arg\max_{\boldsymbol{C}^{\Delta}_{\text{sys}}} B(\boldsymbol{C}^{\Delta}_{\text{sys}}, \boldsymbol{W}) - L(\boldsymbol{C}^{\Delta}_{\text{sys}}, \boldsymbol{W}) \tag{7}$$

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.

$$B = \sum_{k \in [0, T_L]} H_{\text{sys}}(\boldsymbol{W}(k), \boldsymbol{C}^T_{\text{sys}}(k)) \tag{8}$$

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

$$L = \sum_{k \in [1, M]} H_{\text{sys}}(\boldsymbol{W}(t_k), \boldsymbol{C}_k) \cdot T_r \tag{9}$$

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 for having each of N_s servers offline for T_r seconds for the new workload but with the old configuration, *i.e.*, the operations

that would have been completed if the servers remained online and unadjusted. IRIS is general enough to work with any reconfiguration cost, potentially even different costs for different parameters, and these can be fed into its optimization function (Eq. 5). In total $N_s \cdot T_r$ seconds of downtime occurs per server, thus from Eq. (4), we use H_{sys} instead of $N_s \times H_i$ in our summation.

The objective is to maximize the time-integrated gain (benefit – cost) of the reconfiguration from Eq. (7) and IRIS is responsible for determining the optimal reconfiguration plan. The three unknowns in the optimal plan are M, $\boldsymbol{T}$, and $\boldsymbol{C}$, from Eq. (6). 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 look-ahead period T_L.

A greedy solution for Eq. (7) 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 trivially extended to longer sequences of workloads.

The value for T_L should be bounded by the confidence of the workload predictor. A value that is too large will cause IRIS to include decision points with high errors, and a value that is too small will cause IRIS to take almost greedy decisions. In our use case, we simply set T_L based on the historical accuracy of the ARIMA model through a simulation process, where we set the lower threshold of 70% for the accuracy.

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 genetic algorithms, or meta-heuristics in general, do not guarantee finding global optima, they are known to converge to reasonable solutions quickly, especially in non-convex search spaces. In such cases, there are few alternatives and speedy decisions are critical. Our space is non-convex because of the interdependence between time and reconfiguration where the penalty for switching may be amortized over the gap between reconfigurations, and that the placement of each reconfiguration on the timeline has a feedback effect on that amortization. So local decisions about where to reconfigure are impacted by other reconfiguration decisions in the same horizon period. Therefore, we cannot apply greedy searches or gradient descent-based searches on our space because we are likely to get stuck in local optima.

The representation of the GA solution incorporates two parts. First, the chromosome orientation, which is simply the reconfiguration plan (Eq. 6). The second part is the fitness function definition used to assess the quality of different reconfiguration plans (chromosomes). For this, we use the cost-benefit analysis as shown in Eq. 7 where fitness is the total operations 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 MATLAB function `GA`, with 0.8 crossover fraction and population size of 50. We terminate the search process either after investigating all possible values of M, or after 3 consecutive generations with similar best solutions.

4.5 Distributed Protocol for Online Reconfiguration

Cassandra and other distributed databases maintain high availability through configurable redundancy parameters, CL and 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. Therefore, at any time at most RF $-$ CL servers may be offline (due to reconfiguration in our case) and beyond that, database requests will start to fail. As a result, IRIS makes the design decision to configure up to $R = \text{RF} - \text{CL}$ servers at a time. In the case where RF $=$ CL, IRIS cannot reconfigure the system, without harming data availability. However, we expect most systems with high consistency requirements to follow a read/write quorum [21] with the minimum CL that satisfies CL $> \frac{\text{RF}}{2}$ rather than RF=CL. Note that IRIS reduces the number of available data replicas during the transient reconfiguration periods by taking R servers offline. 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, IRIS 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: 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. Start: Restart Cassandra process on the same node.

5. Sync: IRIS 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 data through a node-level repair process [1]. In Cassandra, writes for down nodes are cached by available nodes for some period (denoted as *max_hint_window_in_ms*). These cached writes (known as hinted handoffs) are resent to the nodes when they rejoin the cluster. IRIS 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 $T_R = \frac{N_s \times T_r}{R}$. During all steps 1-5, additional load is placed on the non-reconfiguring servers as they must handle the additional write and read traffic during the outages. 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.

Availability vs. Throughput: IRIS-AGGRESSIVE

In IRIS, we maintain availability during the entire reconfiguration period, while respecting the consistency guarantee. As the number of servers in the cluster grows, IRIS suffers from longer reconfiguration periods because of the sequence of steps where $R =$ (RF – CL) servers are reconfigured at each time step. At the limit, with very large clusters, IRIS performs identically to the static optimized configuration because the time for which any workload persists is smaller than the time to reconfigure the cluster. Therefore, we come up with a variant of IRIS called IRIS-AGGRESSIVE, which reconfigures $R >$ (RF – CL) servers at each time step. In the limit, it can reconfigure all server instances at the same time. This does make the data unavailable for a transient time during the reconfiguration, but completes the entire reconfiguration operation faster. This may therefore be well suited to large clusters, fast changing workloads, or non-interactive throughput-sensitive workloads. Nevertheless, IRIS-AGGRESSIVE is different from a naïve solution that reconfigures upon each workload change and this shows up as a performance advantage (Figure 5).

5 Dataset and Implementation

5.1 MG-RAST Workload

For short-horizon reconfiguration plans, we use real workload traces from MG-RAST. Users of MG-RAST are allowed to upload "jobs" to its pipeline, with metadata to annotate job descriptions. In the upload phase, data is uploaded in FASTA, FASTQ, and SFF formats, followed by a data hygiene phase in which MG-RAST performs tasks such as duplicate removal and sequence similarity checking. All jobs are submitted to a computational queue, pending sufficient computational resources. 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.

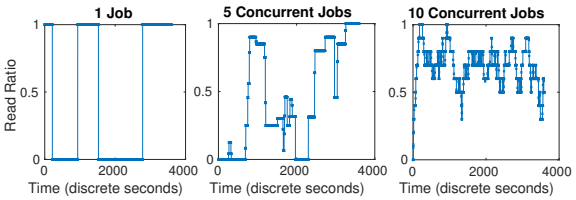

Figure 3: Simulated Workload patterns for 1, 5, and 10 concurrent jobs

5.2 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 IRIS 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. One example of such a scheduler comes from [19], which achieves very high accuracy in predicting execution times of jobs submitted to a Microsoft Cosmos cluster. Thus, in this application IRIS is able to drive long-horizon reconfiguration plans. Modeled on the jobs in [19], 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 [3, 19]) 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 $\sim$ 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.

5.3 Workload Representation

IRIS's simple yet sufficient workload model is $\boldsymbol{W}(t) = \{\text{RR}(t), \text{KRD}\}$. The definition of similarity is a difference of 0.1 in the RR, with the KRD being kept very large to capture the relatively constant KRD seen here. We also simplify 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.

IRIS uses DNN-based RAFIKI [32] as its static configuration tuner. It trains RAFIKI based on the specs of the AWS instances and training workload worth 60 days.

5.4 Testbed Setup

IRIS is designed for a clustered Cassandra setup. Both our workloads use RF=3 and CL=1 for both reads and writes. We evaluate IRIS on Amazon EC2 using instances of size M4.xlarge with 4 vCPU's and 16 GB of RAM for both 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 (this is varied in Experiment 5). We used multiple concurrent clients to saturate the database servers and added the throughput of every client for the output metric, the system-level throughput.

6 Experimental Results

Here we evaluate the performance of IRIS under the short-horizon lookahead (MG-RAST) and long-horizon lookahead (data analytics HPC jobs).

6.1 Baseline Comparisons

We compare the performance of IRIS to baseline configurations (1-5). We also consider 2 variants of IRIS (6-7).
(1) Default: The database administrator (DBA) simply starts Cassandra with the default configuration parameters. We find that by default Cassandra is configured to favor 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, with a new workload, 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 simplistic extension of all of this prior work.
(4) ScyllaDB: We compare IRIS's performance (with Cassandra) to the *self-tuning* ScyllaDB in its vanilla form. ScyllaDB is touted to be a much faster drop-in replacement to Cassandra [40].
(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) IRIS **with Oracle:** Here we apply IRIS's reconfiguration plan for the new workload, assuming fully accurate workload predictions.
(7) IRIS**:** This is our complete system. It uses ARIMA modeling for the short-horizon workload prediction (MG-RAST) and perfect prediction for the long-horizon lookahead (HPC data analytics).

6.2 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 IRIS 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.
ARIMA-based Prediction Model: We created 122,018 training samples composed of $T_d = 30$ second steps across the 60 days MG-RAST workloads. The model was constructed using an ARIMA (20,1,20) model. The complexity of the model (as $p+q$) was selected by minimizing the Bayesian information criterion (BIC), The predictor is initialized with 500 samples of history and the predictions are for 15 minutes into the future, providing a real-valued output. 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 accuracies (the ratio of predictions within 10% of the actual) 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 [29], the ARIMA-based model only provides accurate predictions for short time intervals. We notice that the accuracy of the ARIMA-based model 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 IRIS.
Performance Comparison:

Now we show the performance of IRIS with respect to the four workload categories. We first present the result with the smallest possible number of server instances,

Table 1: Accuracy of the Workload Prediction. We use the three representative workloads corresponding to different frequencies of workload switches.

Lookahead	1m	2m	5m	15m	Num Switch	RR
Slow	90.2%	83.6%	70.5%	70.5%	2	70%
Medium	77.0%	60.7%	14.8%	46.0%	4	59%
Fast	44.3%	39.3%	47.5%	39.3%	14	45%
Write	45.9%	59.0%	36.1%	37.7%	8	35%
Aggregate	64.4%	60.7%	42.2%	48.4%		

4, run with MG-RAST's parameters RF=3 and CL=1. 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 key result of our paper with performance improvements for our test cases.

From Figure 4, we see that IRIS always outperforms naïve for total ops/s (average of 31.4%) and individually for read (31.1%) and write (33.5%) ops/s. IRIS 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 slightly higher throughput over IRIS by 6.3%. This is because the majority of the selected samples are read periods (RR=1), which hides the gain that IRIS achieves for write periods. However, we see that with respect to write operations, IRIS 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 IRIS performs similar IRIS w/ Oracle case in the slow and mid scenarios, which shows the impact of the workload predictor. However, we notice that in the fast scenario, IRIS shows a loss of 8% in comparison to the 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 [40]. However, this claim is not borne out here and only in the read-heavy case (the "Slow" scenario) does ScyllaDB outperform. In this case, IRIS 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 IRIS for the entire range of workload mixes and not have to transition to ScyllaDB.

6.3 Experiment 2: HPC Data Analytics Workload

In this set of experiments we evaluate the performance of IRIS using HPC data analytics workload patterns described in Section 5.2. We show the result in terms of total operations for each test workload. Figure 6 shows the result for the three levels of concurrency (1, 5, and 10 jobs). We see that IRIS 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 IRIS 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, IRIS switches only once: from the default configuration to the same configuration as the static optimized. In comparison to the theoretical best performance, we notice that IRIS achieves within 9.5% of the performance for all three cases. Finally, we notice that IRIS achieves significantly better performance over the naïve approach 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, IRIS 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.

6.4 Experiment 3: Scale-Out

Figure 5 shows the behavior of IRIS using the same workload as in Experiment 2. We show a comparison between IRIS and the static optimized configuration for different cluster sizes under a weak scaling pattern, *i.e.*, keeping the data per server fixed while still operating at saturation. As we scale out, the benefit of IRIS's reconfiguration plan over the static optimized configuration decreases because the total time to reconfigure, T_R, grows linearly with N_s. For the quorum case (CL=2), the total reconfiguration time is even longer as R is halved. However, the loss at each reconfiguration point is lower compared to CL=1 because only 1 server is taken offline. Therefore, IRIS with Quorum is not significantly worse than IRIS. One solution to reduce the total reconfiguration time is to use the aggressive version of IRIS with $R = N_s$. The aggressive setup performs much better with scale because the T_R becomes constant, independent of N_s. Our results show that the aggressive case performs

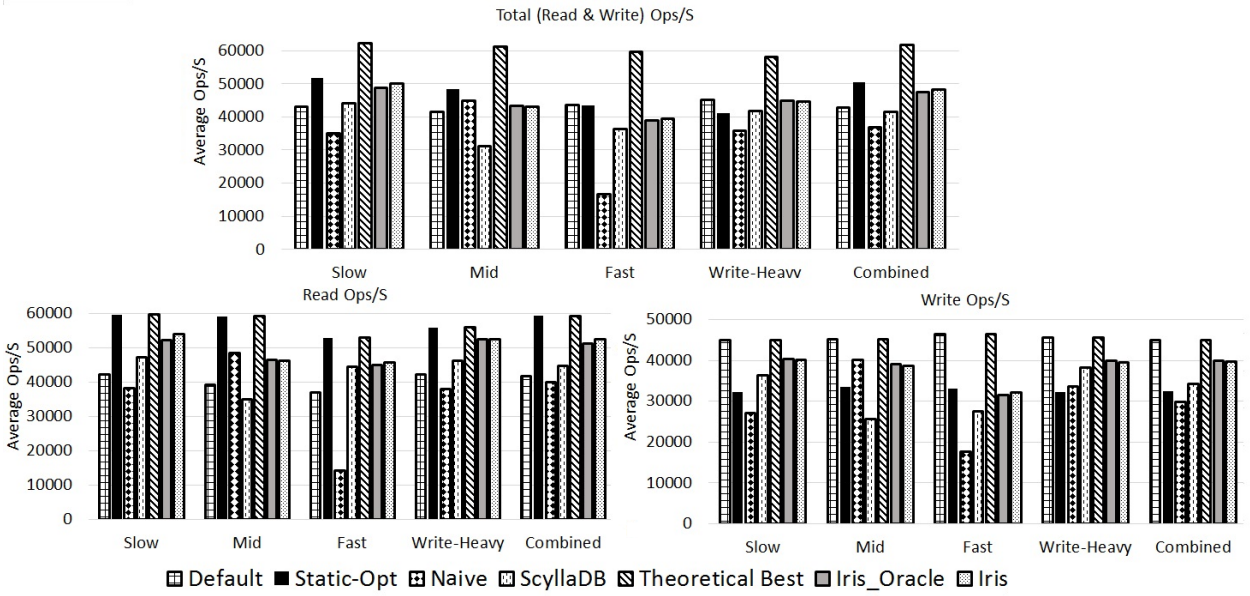

Figure 4: Improvement for four different 30-minute test windows from MG-RAST real traces over the baseline solutions.

close to the ideal, paying only a one-time cost of shutting down servers (concurrently) and restarting them.

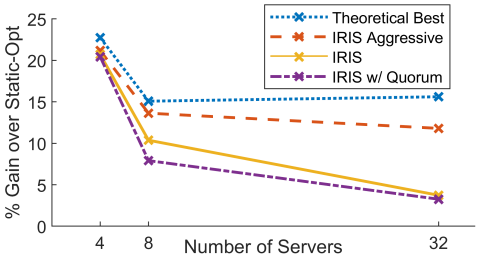

Figure 5: Improvement with scale using 4, 8, and 32 servers. The theoretical best shows the upper limit of a perfect IRIS *as it has no reconfiguration cost.* IRIS-AGGRESSIVE *provides consistent gains across scale because the reconfiguration plan does not change with scale–* $R \leftarrow N_s$. *Full* IRIS *has reduced gains since the number of serial steps increases with the number of servers for a fixed RF and CL. The impact of increased consistency requirements are shown with* IRIS *w/ Quorum.*

6.5 Experiment 4: Noisy Workload Predictions

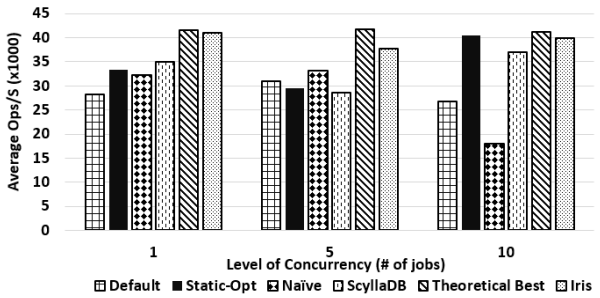

Figure 6: Improvement for HPC data analytics workload with different levels of concurrency.

We show how sensitive IRIS is to the level of noise in the predicted workload pattern. 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 (and therefore unprofiled) jobs. We use the HPC workload with 5 concurrent jobs shown in Figure 3. 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 7, we see that adding noise to IRIS slightly reduces its performance. However, such noise will not cause significant changes to IRIS's optimal reconfiguration plan. This is because IRIS 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 IRIS to achieve significant improvements for long-term predictions even with high noise levels.

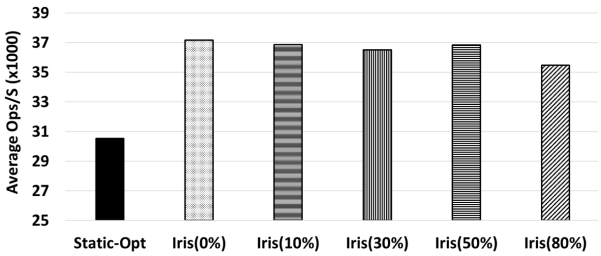

Figure 7: Effect of noise to improvements with IRIS *on simulated workload with level of concurrency = 5. The percentage represents the amount of noise added to the predicted workload pattern.*

Now we show how sensitive IRIS is to noisy predictions of the duration of a changed workload (Figure 8) where we plot the gain of IRIS over a static optimized configuration. We use a synthetic workload so as to

zoom in on the effect that we are trying to isolate. The workload starts with a read heavy workload for 3 min, switches to a write heavy workload for 3 min, and then back to read heavy of 3 min. We generate the reconfiguration plan for the synthetic workload, but then inject an error in the prediction of the write heavy workload duration using a random variable $U(0, 6\text{ min})$. Expectedly, the peak gain is achieved when the difference between the actual workload and the predicted workload is close to zero, while the gain becomes negative on both sides if the difference is $<$ -60 sec or $>$ 80 sec. One interesting observation is that the curve is *not* symmetric. This is because when the predicted duration of the new workload (the write heavy workload) is too small (the right side on the X-axis), then Cassandra runs a write heavy workload with a read optimized configuration. On the other hand, when the predicted duration is too large (the left side on the X-axis), then Cassandra runs a read heavy workload with a write optimized configuration. The latter has a greater performance penalty than the former (47% compared to 26%) and hence the IRIS performance is worse on the left hand side on the X-axis.

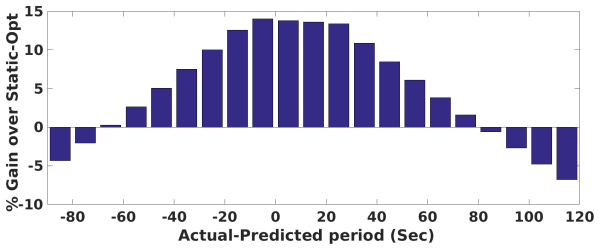

Figure 8: Effect of noise in the workload predictor on improvements due to IRIS. The gain degrades as the predictor becomes more error prone but the curve is not symmetric around the zero point because the mis-prediction costs in read-heavy vs. write-heavy are different.

6.6 Experiment 5: Greater Data Volume

We evaluate IRIS 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 optimized in Figure 9. We notice that the gain from applying IRIS'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 IRIS had the compaction method parameter set 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. This can be addressed using either of two strategies: replacing RAFIKI by a data volume-aware static tuner, or retraining RAFIKI when a significant change in data volume per node occurs.

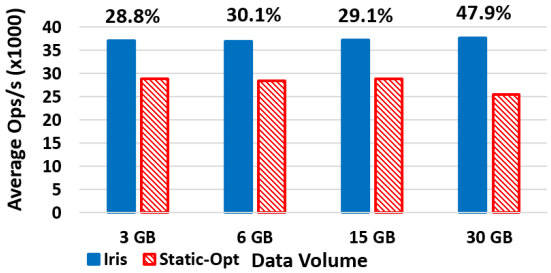

Figure 9: Effect on increasing data volume per node. We use a cluster of 4 servers and compare the performance to the static optimized. The results show that IRIS's gain is consistent with increasing data volumes per node.

6.7 Major Insights

We draw three key insights from these diverse results. First, globally shared infrastructures with black-box jobs only allow for short-horizon workload predictions. This causes IRIS to take single-step reconfiguration plans and limits its benefit over a static optimized approach (Figure 4). In contrast, when analytics jobs queue up and their characteristics can be modeled well, IRIS achieves significant benefit over both default and static optimized cases (Figure 6). This benefit stays even when there is significant uncertainty in predicting the exact job characteristics (Figures 7 and 8). 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 IRIS, atop Cassandra (Figures 4 and 6). Third, as the number of server instances increases, the reconfiguration time of IRIS increases, thereby limiting its benefit. However, the IRIS-AGGRESSIVE variant recovers most of the gains as its reconfiguration time stays constant with scale (Figure 5).

7 Discussion and Future Work

Some tuning parameters are data size dependent, such as the row and key caches and the compaction method, and thus some of IRIS's performance is dependent on the database size. IRIS does not address such concerns, say by predicting the data-size dependent response of a configuration, instead relying on RAFIKI to predict Eq. (3)–a function that lacks data size as an input. We can address this shortcoming by including data size as a parameter in the modeling that IRIS does.

It is possible that a change in RF and CL will affect the operation of IRIS. A change in either of these parameters will doubtless change the net throughput of the Cassandra cluster. However, we posit that the optimal configuration parameter values will not change with the change in these two parameters, up to a certain degree. This is because within a bounded range of change, the work-

load as seen by each server instance will stay relatively constant. However, if these parameters do change significantly, then the above condition will no longer hold and we may need to use a RAFIKI that has been specifically trained with these different (RF, CL) combinations. In practice however, a Cassandra cluster generally uses a fixed value of RF and of CL set to either 1 or RF/2 + 1 (defined as `LOCAL_QUORUM`); the former provides the loosest form of consistency while the latter provides strong consistency [2].

Currently, we enforce an atomic reconfiguration policy, which has the advantage that all server instances are homogeneous in terms of their contribution to the system throughput. The downside of this policy is that for frequent workload changes, there may not be time to reconfigure all the server instances. So in these workload scenarios, the server instances will continue to keep the current configuration. This may be sub-optimal in that it may be better to have the first fraction of the servers run with the new configuration and the second fraction with the earlier configuration. We leave it for future work where reconfiguration decisions are made on a per-server basis rather than atomically for the entire cluster.

8 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 [13, 7, 20, 16, 47]. Among these, Morphus [20] targets MongoDB, a noSQL DBMS but cannot handle Cassandra due to its peer-to-peer topology and hash-based sharding. In general, data partitions appear more suitable to online changes than updating configuration parameters. [31] compares the performance of Casandra to ScyllaDB, and Kemme *et al.* [26] proposed an online reconfiguration approach for replicated databases and focuses on efficient data transfer and fault-tolerance and does not optimize for dynamic workload changes. Tuba [6] reconfigures geo-replicated key-value stores by changing locations of primary and secondary replicas to improve overall utility of the storage system. A large body of work also focused on choosing the best logical or physical design for static workloads of relational databases [12, 10, 51, 22, 11, 44, 4, 38, 39]. Another body of work attempts to improve performance for static workloads by finding correct settings for DBMS performance knobs [15, 14, 32, 50, 45] as discussed before. In contrast, IRIS performs online reconfigurations of the performance tuning parameters of distributed NoSQL DBMS for dynamically changing workloads.

Reconfiguration in distributed systems and clouds. A wealth of work has addressed the problem in the context of traditional distributed systems [26, 5] and cloud platforms [30, 49, 34, 33]. Some solutions present a theoretical approach, reasoning about correctness for example [5], while some present a systems-driven approach such as performance tuning for MapReduce clusters [30]. BerkeleyDB [37] models probabilistic dependencies between configuration parameters. A recent work, *SmartConf* [46] provides a rigorous control-theoretic approach to continuously tune a distributed application in an application-agnostic manner, even as workloads change. However, it cannot consider dependencies among the performance-critical parameters and cannot handle categorical parameters. In contrast, IRIS considers the trade-off of an online reconfiguration through CBA and can handle dependencies among the performance-critical parameters as well as categorical parameters.

Reconfiguration in OS and other single-node stack. There has been long-past work on this topic [41, 23, 36] 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 [41] or to perform maintenance tasks, *e.g.*, pinpoint performance problems [41, 36]. In contrast, IRIS provides an online reconfiguration mechanism for distributed NoSQL DBMS to optimize performance.

9 Conclusion

When faced with dynamic and fast-changing workloads, NoSQL databases have to be tuned for achieving the highest throughput. Current practice dictates that clusters of Cassandra server instances be shut down for reconfiguration and while some prior works can provide optimal configuration settings for any given workload, they cannot perform online reconfiguration. Here we presented IRIS to perform such online reconfiguration while maintaining availability of the data records and respecting the consistency level requirements. We achieve this through three techniques: a simple workload predictor, which can predict the duration of the new workload pattern, a CBA, and a distributed protocol to gracefully switch over the cluster from the old to the new configuration. We apply IRIS to MG-RAST's metagenomics workload traces and see that it outperforms all prior techniques either during the entire operation or during intense write bursts, the latter typical in this application domain. The gains are greater for a more predictable HPC data analytics workload. 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 IRIS factor in configuration parameters whose changes take effect only after a time lag?

References

[1] Cassndra Repair. http://cassandra.apache.org/doc/latest/operating/repair.html.

[2] DATASTAX: How is the consistency level configured? . https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html. [Online; accessed 27-April-2018].

[3] 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.

[4] 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).

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

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

[7] 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.

[8] CASSANDRA, A. Compaction. http://cassandra.apache.org/doc/latest/operating/compaction.html.

[9] 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).

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

[13] 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).

[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] ENTERPRISE, D. Apache Cassandra 3.0: How is Data Maintained? http://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlHowDataMaintain.html, 2017.

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

[19] 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.

[20] 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).

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

[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] INC., A. E. PBS Professional Open Source Project. http://www.pbspro.org/, 2018. [Online; accessed 1-May-2018].

[25] INC., D. Configuring compaction. https://docs.datastax.com/en/cassandra/2.1/cassandra/operations/ops_configure_compaction_t.html.

[26] 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.

[27] 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.

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

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

[30] 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.

[31] MAHGOUB, A., GANESH, S., MEYER, F., GRAMA, A., AND CHATERJI, S. Suitability of nosql systemscassandra and scylladbfor iot workloads. In *Communication Systems and Networks (COMSNETS), 2017 9th International Conference on* (2017), IEEE, pp. 476–479.

[32] 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.

[33] 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).

[34] 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).

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

[36] 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.

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

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

[39] 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).

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

[41] 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).

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

[43] 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).

[44] 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).

[45] 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.

[46] 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).

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

[48] WILKE, A., BISCHOF, J., GERLACH, W., GLASS, E., HARRISON, T., KEEGAN, K. P., PACZIAN, T., TRIMBLE, W. L., BAGCHI, S., GRAMA, A., ET AL. The mg-rast metagenomics database and portal in 2015. *Nucleic Acids Research 44*, D1 (2015), D590–D594.

[49] 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.

[50] 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).

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