INTELLIGENT WORKLOAD PLACEMENT AND SHARDING BETWEEN ORACLE, POSTGRESQL, MONGODB, AND CASSANDRA USING MACHINE LEARNING

Adithya Sirimalla

Software Developer
Enliven Technologies Inc.
USA.

Abstract:
Polyglot persistence has become a popular trend in the modern world, and it is now common to find modern applications that depend on heterogeneous datastores, including Oracle, PostgreSQL, MongoDB, and Cassandra. Both engines have unique benefits in terms of consistency, latency, scalability, and data modeling; however, neither one of them is the best system to use for each workload pattern. Static location heuristics do not reflect real-life workload variations, which results in a reduction in performance, unwarranted resource expenses, and complexity of operations. In the current study, there is an Intelligent Workload Placement and Sharding Advisor that is an ML-based tool that examines workload signatures (read/write ratios, latency percentiles, payload distributions, access skew, and transactional requirements) to advise the best setting of data placement among heterogeneous engines. It has a structure that combines the use of supervised machine learning to predict latency and throughput, an engine recommender that uses classification, and a reinforcement learning migration policy that compares the improvement in long-term performance with migration overhead. Synthetic OLTP traces using YCSB workloads, mixed read/write patterns, and synthetic OLTP traces show that the proposed system significantly improves over the static baselines, such as in reducing p95 latency, maximum throughput at bursty workloads, and SLA violations. The study shows that automated workload-aware decision systems can be used to enhance performance and minimize the operational load in multi-engine database systems.

Keywords: Workload-aware placement, Sharding, Oracle, PostgreSQL, MongoDB, Cassandra, Machine learning for databases, reinforcement learning, distributed storage, and YCSB benchmarking.

1. INTRODUCTION

The rapid growth of data-intensive systems has prompted the use of polyglot persistence, in which several database engines can coexist within the same application ecosystem. Traditional relational databases, such as Oracle and PostgreSQL, are still needed when the workload has high ACID guarantees, more advanced joins, and strong transactional semantics. Simultaneously, non-relational databases, such as MongoDB and Cassandra, are now the default options where flexibility in schema, horizontal scaling, high write throughput, or a globally distributed implementation is required. The engines have their own set of strengths and restrictions, but none of the systems can provide the best performance at all possible workload patterns.

This mixed-up setting leads to a continuing working quandary:
What is the smart way to partition, move, or distribute data in Oracle, PostgreSQL, MongoDB, and Cassandra to perform optimally with the dynamic workload features?
Traditional decision-making techniques are largely based on rule-of-thumb heuristics, including putting transactional data on relational engines and document-heavy traffic on MongoDB; however, they often do not work in actual load dynamics. The workload is dynamic and changes according to user activity, application

logic, feature rollout, traffic bursts, and seasonality. What is currently a read-intensive workload may become a write-heavy workload in the future. A dataset that recently required strong consistency can subsequently require low-latency access or geo-distributed reads. Furthermore, because of the varying request arrival rates and access skew, the performance of a storage engine compared with other engines may change dramatically.

This changing environment reveals three fallacious premises usually considered for manual data placement:

- The long-term performance was predicted using static characteristics.
- Performance is dependent on the temporal workload shape, which can be a read/write mix, payload variability, transaction boundary, and latency distribution, which tends to vary unpredictably.
- Manual placement is sufficiently efficient to follow the workload drift.

Multidimensional workload metrics cannot be interpreted by operators, and thus manual decision-making has been found to be slow, reactive, and subject to suboptimal decisions.

- Data migration is too expensive to be proactive.

Whereas migrations carry an overhead cost, the cost of not migrating is greater in the long run, because the impact of long-term penalties in terms of continued tail-latency spikes and SLA breaches is even more severe. These issues suggest the necessity of an independent, data-driven, workload-sensitive placement advisor that can continuously analyze system behavior and offer the best engine placement or sharding choice proposal. To overcome this gap, this study suggests an ML-based Intelligible Workload Placement and Sharding Advisor for the workload that evaluates workload trends in real time and resolves the following issues:

- Which storage engine would be most appropriate for the present workload?
- When the advantages of data migration are higher than costs
- Sharing, rather than complete migration, is a more effective intervention.

The proposed system incorporates the following:

- **An online monitoring layer** that records workload signatures (read/write ratios, latency percentiles, payload sizes, consistency requirements, and access skew).
- **Predictive learning models** that forecast the anticipated latency and throughput of each workload in the event it was executed on Oracle, PostgreSQL, MongoDB, or Cassandra.
- A classification-based recommender that selects the best engine based on the predicted performance.
- **A reinforcement learning (RL) migration policy** makes decisions on migration or data sharding based on the degree of performance versus migration cost.

It goes beyond blatant, uninformed heuristics and proposes a conceptual framework for dynamic and data-driven engine selection. The advisor can adapt to workload drift by taking advantage of ML predictions and the RL policy of decisions, reducing the number of unnecessary migrations, and stabilizing the tail-latency performance in a wide range of deployment environments.

The findings of this study contribute to the following:

- An integrated workload characterization model that is designed to fit heterogeneous relational and NoSQL systems.
- Latency and throughput prediction learning model on four database engines with real workload signatures.
- A reinforcement learning policy that estimates the dynamics of a long-term reward to cause a migration or sharding decision.
- A laboratory analysis of YCSB workloads, OLTP-style synthetic traces, and mixed access patterns showed that the throughput, latency, and SLA conformity improved.
- An integrated architectural design incorporating prediction, recommendation, and RL-based governance into a viable advisory system in the context of contemporary cloud-native architecture.

The remainder of this paper is organized as follows.

Related work is presented in Section 2.

The problem is formalized in Section 3.

Section 4 describes the system architecture.
The characterization of the workload is presented in Section 5.
The ML models are described in Section 6.
Section 7 describes the experimental setup.
Section 8 presents the results.
The implications and limitations are discussed in Section 9.
Section 10 concludes.

2. RELATED WORK

The problem of smart placement and migration of data between heterogeneous storage engines cuts across multiple known areas of research, including benchmarking of cloud-serving systems, comparative analysis of relational and NoSQL databases, placement by workload frameworks, and adaptive sharding measures. In this section, the most pertinent contributions are reviewed and gaps that drive the proposed ML-based advisory system are identified.

2.1 Cloud and NoSQL Storage System Benchmarking.

Benchmarking is also central to defining the performance of distributed data systems. The Yahoo! Cloud Serving Benchmark (YCSB) is the original architecture of cloud database testing which offers standardized workload definitions and scaleable measurement aids on latency, throughput, and workload mix [1]–[4]. All these studies highlight a significant difference in performance among engines based on the read/write ratios, arrival rate of requests, and document sizes. The extensibility of YCSB has spawned derivative tools, such as YCSB++ [3], which provides more detailed information on performance bottlenecks and tuning opportunities. Other literature on benchmarking goes further to analyze cloud workloads under varying conditions of virtualization, caching, and I/O, which once again points to the uncertainty of the engine performance when workloads are mixed [5].
Although these tools show a variety of performance attributes across DBMSs, they fail to answer the more profound question of how workloads are to be dynamically allocated to particular engines as the behavior changes with time. The suggested system is based on these insights and extends the idea of static benchmarking to that of a predictive workload-driven placement decision.

2.2 Comparison of SQL and NoSQL Studies.

Several experiments have compared relational and non-relational engines at different workloads. Comparative studies on PostgreSQL, MongoDB, Cassandra, and other no-SQL systems show explicit variations in the speed of insertion, read latency, consistency assurances, and limits of scaling [10], [11]. MongoDB is better suited to semi-structured document workloads, whereas Cassandra is better suited to write-heavy distributed environments because of its log-structured storage and eventual-consistency model [2]. PostgreSQL and Oracle, on the other hand, are faster in terms of transactional workloads, complex joins, and strong consistency.
However, most of these comparisons assume a static workload configuration and do not reflect the dynamic and changing nature of behavior when deployed in the real world. This underscores the importance of having systems that can deduce the best placement based on the characteristics of the workload at the moment, as opposed to stereotyping a database.

2.3 Distributed Systems Consistency Latency Tradeoffs.

The tradeoffs between consistency, scalability, and latency have been discussed in a significant amount of literature. The classic study by Bailis and Ghodsi [16] and the Spanner architecture of Google [9] provide examples of how the replication and consistency mode choices affect performance. Subsequent research has designed the effects of eventual consistency, multi-version concurrency control (MVCC), and replica location on the behavior of the system in general [6], [7], [17], [18], [89]. These results confirm that it is not possible to optimize latency and consistency separately; therefore, workload-aware scheduling is the key to addressing heterogeneous engines that can be run according to various consistency models.
Although this literature provides theories to reason about tradeoffs, it lacks mechanisms for adjusting placement decisions on a dynamic basis.

2.4 Data Placement and Reconfiguration on Workload Awareness

Earlier systems, such as SWORD [23], ElasTraS [28], and other adaptive middleware systems [26], [27], suggest ways of partitioning, reconfiguring, or load balancing information depending on new circumstances. These methods demonstrate that workload-sensitive placement can make it more efficient; however, such methods are strongly dependent on hand-crafted heuristics or fixed limits. Research on elastic data stores also indicates the promise of adaptive placement and admits that it is difficult to tune reconfiguration policies in changing environments [24], [25], and [27].

In addition, they can be run on a single engine or a particular architecture family (i.e., key-value stores and transactional stores). None of them deal with cross-engine orchestration between relational and NoSQL systems simultaneously, which is a key gap that our framework addresses.

2.5 Sharding Strategy and Partitioning Models.

Prolonged research on distributed systems has explored sharding, partitioning, and key-distribution strategies. Hashed, range-based, and hybrid sharding strategies affect performance and load distribution, which are described in surveys and empirical studies [25], [29], [30]. The wide-column structure of Cassandra is a representation of hashed-sharding that allows high write throughput [2], unlike MongoDB, which has an option of hashed or ranged-sharding. PostgreSQL and Oracle relational engines usually have the sharding feature as an external tool or extension, which makes it difficult to cross-engine orchestrate.

These studies validate the value of sharding to enhance scalability but do not investigate data transfer among radically different data paradigms or use machine learning to inform shard placement decisions at realistic workloads.

2.6 Summary and Research Gap

The available literature offers adequate underpinnings for benchmarking, information segmentation, and performance analysis of distributed systems. Nevertheless, three gaps remain unaddressed.

- **Absence of cross-engine workload-sensitive placement systems.**

The previous experiences are based on tuning one engine instead of coordinating data between the relational and NoSQL systems in a co-ordinated manner.

- **Totality of ML-based latency and throughput prediction in heterogeneous DBMS environments.**

None of the current frameworks predict engine-specific performance based on the signatures of live workloads using supervised models.

- **None of the reinforcement learning methods operated migration and sharding choices.**

The optimization of migration costs and trade-offs between long-term performance is not automatic.

To fill these gaps, this study proposes a single advisor with ML capability that predicts engine performance, selects the best placements, and controls the migration/sharding behavior via reinforcement learning.

3. PROBLEM FORMULATION

Heterogeneous database environments require the system to determine the location of every dataset and the time it must be. To make this decision intelligently, the advisor requires knowledge of the workload exerted on each piece of data and the approximate ways in which that workload would perform on other database engines, such as Oracle, PostgreSQL, MongoDB, and Cassandra. This section identifies the problem in very specific conceptual language, as opposed to mathematical terms.

3.1 Data and Workload Model

Operation streams in the system, such as reads, writes, updates, and transactions, are placed into each dataset in the system, whether it is a table, collection, key-range, or partition. In a short time frame, the system summarizes such operations in a structured workload signature that includes the following features:

- the ratio between the number of reads and the number of writes,
- the frequency of arrivals of requests,
- typical payload sizes,
- noticed latency percentile (median, p95, p99),
- whether requests are sensitive to strong consistency or can be made to be eventual consistency,

- whether the work load contains multi-step transactions or joins,
- and the skew of access (e.g., a small number of keys are used to work on it).

This signature is the behavioral fingerprint of the workload of a dataset at a point in time, upon which performance prediction is performed.

3.2 Engine Capability Model

The performance characteristics of each database engine differ according to the workload received by the database engine. For example, Cassandra is likely to perform better with write-intensive, high-throughput workloads that have loose consistency guarantees, whereas Oracle and PostgreSQL are more effective in transactional workloads and complicated queries. MongoDB has good flexibility in terms of document structure and payload size.

Given a workload signature, the advisor should approximate its behavior, that is, its latency and throughput, on each of the four candidate engines. Because they are highly nonlinear and situation-specific, the system does not learn about them by simply applying some fixed heuristics but instead uses supervised machine learning to learn.

3.3 Data Placement Decision

All datasets are stored on any of the available engines at any given time. The placement decision is the best decision based on the performance expected of an engine in its current workload signature, and the result is which engine is the best fit at that time.

Nevertheless, the choice does not only involve latency. It also has to take into account engine-specific limitations, such as

- whether the data set needs complete ACID compliance,
- whether strong consistency is obligatory,
- price or licensing limitations (particularly in the case of Oracle),
- available storage capacity,
- and compatibility with existing data models.

Thus, the placement decision is a constrained optimization that is feasible.

determine an engine that offers optimum performance **without infringing on operational policies.**

3.5 Migration and Sharding Operations.

Migrating a dataset is not necessarily a good decision, even when an engine is no longer optimal. Migration incurs costs in the following ways:

- network transfer time,
- synchronization overhead,
- potential cutover lag on replication,
- active workload performance degradation, which is temporary.

The concept of sharding itself brings a new set of complexities: partitioning a dataset into multiple partitions can decrease the number of hotspots, but the coordination overhead can increase, and the consistency behavior can be affected.

The system must consider that there is a net advantage in the benefits of moving or sharding that will exceed the anticipated costs. It cannot be just an immediate performance comparison, but has to be a prediction of the long-term effects of migration.

3.5 Long-Term Optimization Goal

There are three conflicting priorities that the advisor has to juggle.

1. **Reducing the latency and enhancing the throughput** of ongoing operations.
2. **Unnecessary or frequent migrations** may cause instability or service failures, which should be avoided.
3. **Avoiding breach of the SLA** during times of high demand or fluctuations in worker workload.

To balance this, it is necessary to consider not only the current performance but also the likely future effect of placement or sharding decisions. An unsophisticated system, such as one that responds solely to changes in latency in the short term, would oscillate or malfunction with a volatile load.

3.6 Reinforcement Learning Environment.

The advisor applies reinforcement learning (RL) to solve this decision-making problem in the long term. In this formulation:

- The workload signature and existing engine containing the dataset are parts of the state.
- These operations consist of retaining the dataset at its current location, transferring it to any of the other engines, sharding it to more than one node, or replicating it.
- The feedback (reward) signal is used to show how well the decision has been made in terms of latency, migration overhead, and SLA violations.

The RL policy is trained to choose actions that generate a long-term reward, as opposed to responding blindly to temporary changes in performance.

This is a solution to the significant deficiency of traditional static or rule-based systems: it entails the element of foresight and an explicit trade-off between short-term costs and future performance benefits.

4. SYSTEM ARCHITECTURE

The Intelligent Workload Placement and Sharding Advisor is a real-time decision system that is designed to be modular and observes the current workload behavior, predicts the performance of each candidate database engine, and decides whether the live data should stay where it is, migrate, or be sharded to achieve better performance. The architecture is a closed loop, where workload telemetry is used to drive predictions and recommendations to drive migration or sharding should be justified.

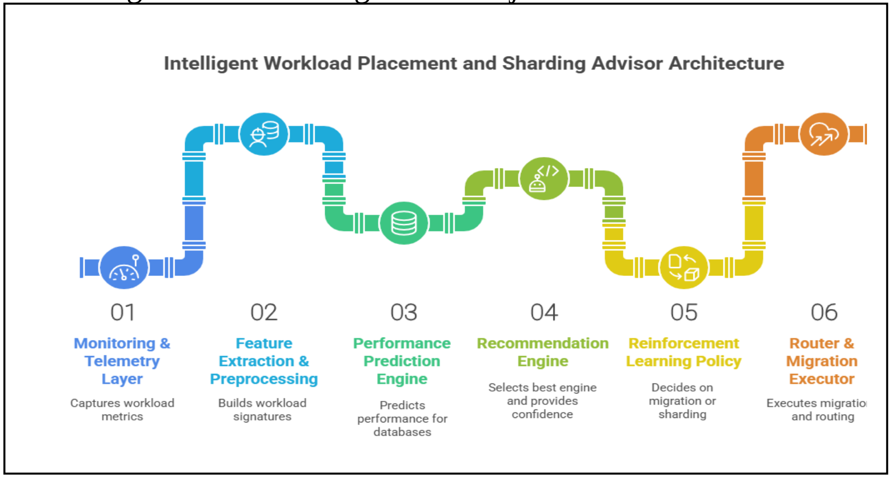

Intelligent Workload Placement and Sharding Advisor Architecture
01
Monitoring & Telemetry Layer
Captures workload metrics
02
Feature Extraction & Preprocessing
Builds workload signatures
03
Performance Prediction Engine
Predicts performance for databases
04
Recommendation Engine
Selects best engine and provides confidence
05
Reinforcement Learning Policy
Decides on migration or sharding
06
Router & Migration Executor
Executes migratio and routing

Figure1: System. Architecture of the Intelligent Workload Placement and Sharding Advisor

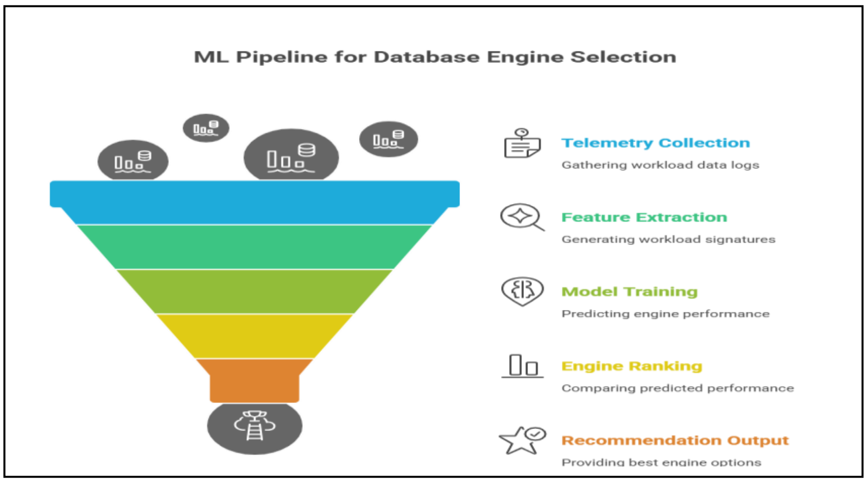

ML Pipeline for Database Engine Selection
Telemetry Collection
Gathering workload data logs
Feature Extraction
Generating workload signatures
Model Training
Predicting engine performance
Engine Ranking
Comparing predicted performance
Recommendation Output
Providing best engine options

Figure 2. Machine Learning Pipeline for Workload-Aware Engine Performance Prediction and Recommendation.

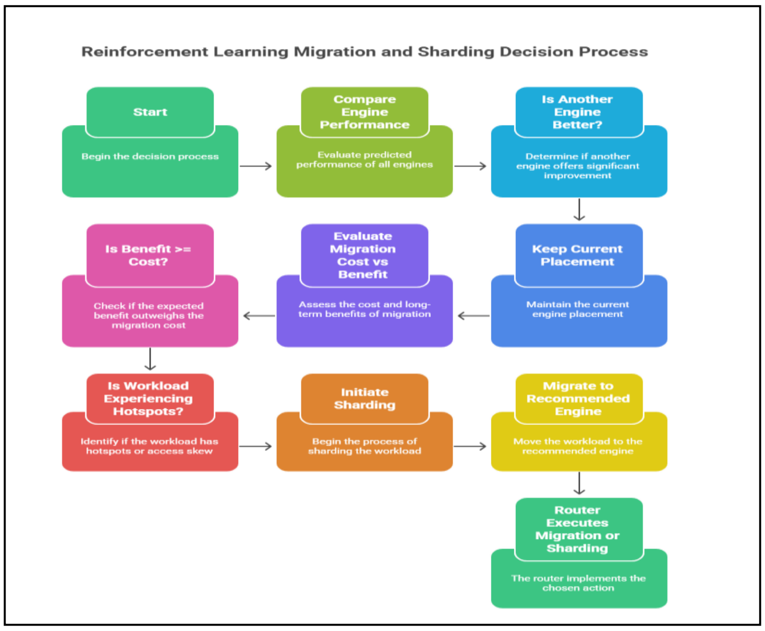

Reinforcement Learning Migration and Sharding Decision Process
Start
Begin the decision process
Compare Engine Performance
Evaluate predicted performance of all engines
Is Another Engine Better?
Determine if another engine offers significant improvement
Keep Current Placement
Maintain the current engine placement
Evaluate Migration Cost vs Benefit
Assess the cost and long-term benefits of migration
Is Benefit >= Cost?
Check if the expected benefit outweighs the migration cost
Is Workload Experiencing Hotspots?
Identify if the workload has hotspots or access skew
Initiate Sharding
Begin the process of sharding the workload
Migrate to Recommended Engine
Move the workload to the recommended engine
Router Executes Migration or Sharding
The router implements the chosen action

Figure 3. Reinforcement Learning Decision Flow for Data Migration and Sharding

An interaction exists between the key components, as shown in Figure 1.
1. Monitoring and Telemetry,
2. Feature Extraction,
3. Performance Prediction,
4. Recommendation Engine,

Under the reinforcement learning policy, we can obtain the following:
5. Migration Executor and Router.

The following are descriptions of each subsystem.
The fourth layer is the monitoring and telemetry layer, which is used to manage data telemetry within the system. Human The fourth layer is the monitoring and telemetry layer, which is used to control the data telemetry in the system.
This layer continues to capture the operational data of all database engines and workload sources. It tracks:

- the kind of every request (read, write, update, transaction),
- when requests come, how many they come at a time,
- size of payloads and format features of data,
- immediate latency response (e.g., p50/p95/p99),
- used consistency mode (strong, eventual, tunable),
- patterns of access and hotspots (e.g., do some keys control traffic),
- isolation levels (transactions)

The most important advantage of this layer is its high temporal resolution: it does not assume that the workloads are sluggish and does not support the fallacy that periodic snapshots suffice. Rather, it records fine-grained variations that are very effective in affecting the performance of heterogeneous engines.

4.2 Feature Extraction and Preprocessing.

Raw telemetry is transformed into a structured workload profile, called a workload signature. It is a module that cleans, aggregates, and transforms the received data into meaningful features. It performs:

- latency normalization and smoothing of the readings,
- calculation of the read/write balance,
- estimation of skewness of access using key-frequency analysis,
- characterization of the heavy and light point query workloads,
- checking variability in size of payload,

- identification of consistency and durability requirements,
- application of the request-arrival burstiness.

This change is critical because machine learning models are highly dependent on the quality of such feature representations. Misleading predictions and unstable placement decisions are the results of poor feature extraction.

4.3 Performance Prediction Engine.

The prediction engine forecasts the performance of the workload on each of the engines used, that is, Oracle, PostgreSQL, MongoDB, and Cassandra.

The system can perform this by employing trained machine learning models that are supervised on benchmarking data and historical workload traces. These models learn the relationship between the workload features and the anticipated results, including

- average latency,
- tail-latency behavior,
- achievable throughput,
- burst and peak traffic sensitiveness,
- Write home and internal processing overhead.

The design is purposely not simplistic heuristics, such as Cassandra being suitable for writes and PostgreSQL being suitable for transactions. Rather, the performance of the system is predicted dynamically according to the monitored workload signatures. The layer is a prediction layer, and this is the analytical base on which the advisor makes decisions.

4.4 Recommendation Engine

Based on the forecasts of the foregoing component, the recommendation engine determines the best database engine to use for each dataset at the given time. Decision-making considers the following:

- identified throughput and latency of each candidate engine,
- that the engine meets necessary semantics about transactions,
- whether the consistency model of the engine is in line with the workload,
- capacity limitations and working policies,
- licensing or cost (for example, Oracle licensing),
- the level of outcome improvement expected and already realized.

Instead of generating one answer, the recommendation engine generates

- the top recommended engine,
- confidence scores,
- substitute products in the event of limitations,
- there are signs to indicate whether the anticipated improvement is small or big.

This prevents the illusion of presuming that the most suitable engine is always visible. Rather, it differentiates between small inefficiencies and severe mismatches that should cause migration.

4.5 Policy of Reinforcement Learning Migration and Sharding.

Migration is not necessarily the correct option, despite the possibility of another engine seeming better. To identify the value of migrating or sharding data, a system based on a reinforcement learning (RL) agent analyzes long-term effects as opposed to short-term performance.

- The following are the inputs to the RL agent:
- the existing workload pattern,
- the existing assignment of databases,
- trends in past performance,
- the projected advantage of other placements,
- the cost of operation of the migration (e.g. overhead caused by data transfer),
- danger of SLA violations during the transition process.

On these grounds, the agent will determine whether to:

- keep the data where it is,
- transfer the data to a different engine,
- begin sharding in order to alleviate the pressure of hotspots,
- copy or reconfigure currently existing partitions,
- or perform no action.

The RL policy addresses the shortfalls of threshold-based systems, whose characteristics lead to oscillations when the workload changes or are too slow in responding to sudden spikes. It brings forward the ability to look forward, stability, and cost-conscious decision-making.

4.6 Router/Migration Executor.

The router performs request forwarding and migration mechanics when it makes a decision. It ensures:

- seamless query redirection,
- dual-write (or shadow-write) migration,
- managed data moving via native applications (e.g., Oracle GoldenGate, PostgreSQL FDW, Mongo dump/ restore tools, Cassandra sstableloader),
- attachment of replicas to the sharding or rebalancing,
- semi-automatic transition with minimal service downtimes.

This aspect converts ML decisions into operational acts and ensures continuity and availability during the transition.

4.7 End-to-End Control Loop

The complete architecture is a feedback loop [ ] that is autonomous and continuous:

- Telemetry of the workload was performed in real time.
- Workload signatures are created.
- Predictions were made for the performance of each engine.
- Suggestions are provided.
- The RL agent judges the value of the action.
- The router performs migrations or maintains its placement.
- The new telemetry is involved in the subsequent decision cycles.
- The system can be used to respond to changes in workload, traffic bursts, and changing performance conditions in a heterogeneous database by utilizing this architectural cycle.

5. WORKLOAD CHARACTERIZATION

Proper placement of data in heterogeneous database engines can only be achieved through an accurate performance of how a given dataset responds to the actual application traffic. Workload characterization transforms raw telemetry into formatted, interpretable properties that are used to describe the behavior of the dataset operationally. Together, these elements, or more specifically, the workload signature, are the input for the prediction, recommendation, and reinforcement learning elements of the system.

In this section, the main dimensions of workload characterization are described, and the reasons why they are important are provided, as well as how these dimensions can capture performance-relevant signals that cannot be captured by naive rule-based systems.

5.1 Read/Write Composition

The relative ratio of these two factors is one of the most effective predictors of database performance.

- Write-heavy patterns are likely to strain engines differently than read-heavy patterns.
- Cassandra and MongoDB generally scale better on high-write or mixed workloads, whereas Oracle and PostgreSQL scale better on balanced or read-heavy workloads.
- A sudden change in the read/write ratio usually indicates that the placement must be reconsidered.

In theory, these ratios are treated as slow; however, in practice, such systems sometimes undergo planned transitions at a very rapid rate with peak activity or rollout of features.

5.2 Latency Distribution (p50, p95, p99)

Median latency provides a consistent measure of normal performance, but tail latency (p95 and p99) is a measure of what systems can do when they are under load and is very important for SLA compliance.
The behavior of different engines under pressure is radically different.

- Cassandra offers predictable tail latency when used in situations of high concurrency and less predictable behavior when used in situations that require complex reads.
- Oracle and PostgreSQL are highly consistent but can have significant bursts of latency.
- When the write contention count is high or the scale of document updates is massive, the performance of MongoDB is surpassed.

The observation of the entire latency distribution, other than the averages, can be used to avoid the illusion of temporary performance.

5.3 Rate of Requests Arrival and Burstiness.

The flow and compressibility of traffic represent the level of aggressiveness with which an engine is stressed.

- I/O and commit pipes are stressed by a high and constant throughput.
- Bursty traffic causes queues, locks, and replication lags.
- NoSQL engines can usually be scaled out in sustained workloads but can experience impaired consistency in peak workloads.
- Transactional workloads are performed very effectively by relational engines; however, extreme write pressures result in poor performance.

This measure invalidates the widely held belief that the more throughput one has, the more resources one has; rather, the throughput shape defines engine appropriateness.

5.4 Size and Structural Variability of Payload.

The size of the payload and its variation have an impact on

- relational engines have page splits,
- behavior of compaction and SSTable in Cassandra,
- overhead in document rewriting in MongoDB,
- accessibility and storage effectiveness of all engines.

Document-oriented storage can be supported by variable payload sizes, but row-based engines can be used when the relational structures are known.

5.5 Transactional Complexity

Multi-step transactions, join operations, and the tenets of isolation of workloads are inherently towards the Oracle or postgreSQL.
Indicators include:

- nested queries,
- foreign key relations,
- multi-statement transactions,
- Update-heavy OLTP behavior.

These patterns can be difficult to deal with using NoSQL engines, except in the case of simplification or denormalization of operations.

5.6 Consistency Requirements

Consistency models are directly related to the performance and correctness of operations.

- Oracle/PostgreSQL ⇒ strict ACID semantics.
- mongodb ⇒ adjustable consistency based on the write concern.
- Cassandra ⇒ Eventual consistency and variable quorum.

The imbalance between the requirement of consistency of workloads and engine assurances usually causes bottlenecks or accuracy problems in performance.
The dimension manages the flawed assumption that consistency and latency are separate decisions.

5.7 Hotspot and Memory Access Skew Detection.

The actual workloads do not have evenly distributed access patterns. Rather, most operations are usually performed on a small fraction of the dataset.

When the concentration of the access activity is high:

- possibility of the formation of hotspots,
- requirement de sharding/replication,
- risk of node-level overload (Cassandra in particular),
- Weaknesses of caching techniques.
- Missing skew causes displacement, unwanted migration, or inadequate sharding.

Table 1. Key Dimensions of the Workload Signature

Workload Dimension	Description	Reason for Inclusion
Read/Write Ratio	Balance of read vs write operations	Determines suitability for NoSQL vs RDBMS engines
Latency Percentiles	Median and tail latency	Captures SLA-critical performance behavior
Request Rate & Burstiness	Volume and irregularity of traffic	Predicts stress on concurrency and replication pipelines
Payload Size Variability	Mean and variance of payloads	Influences rewriting overhead and storage efficiency
Transactional Complexity	Presence of multi-step ACID operations	Indicates need for relational engines
Consistency Requirements	Strong, tunable, or eventual	Determines compatibility with engine guarantees
Access Skew	Degree of hotspot concentration	Drives sharding and rebalancing decisions

5.8. Importance of Multidimensional Signatures

The advisor is not based on individual metrics or deterministic heuristics but on an analysis of the interaction of these dimensions to affect the behavior of the database. The signatures of workloads capture the following:

- interaction effects (high write ratio + burstiness),
- inconsistency of some rare edge cases (e.g., high skew + strict consistency),
- change by degrees and sudden transformation,
- performance differentials between the earning engines.

Such a multidimensional view allows for a specific prediction and a more intelligent recommendation than any rule-based system.

6. MACHINE LEARNING MODELS

The intelligence of the proposed advisor is based on two levels of machine learning:

(1) models of supervised learning that model performance across a variety of database engines, and

(2) A reinforcement-based learning policy that controls the time of data migration/sharding.

This section describes the design, role, and interaction of each model class without using mathematical notation.

6.1. Role of Machine Learning in Workload-Aware Decision Making

The tuning of traditional databases is highly dependent on manual rules (e.g., writing on Cassandra and joining on PostgreSQL). These heuristics fail in the face of real-world workload variations. Machine learning allows the system to:

- learn the patterns of performance automatically,
- model nonlinear workload dimensions interactions,
- generalize to invisible conditions of work,
- and make judgments based on what is predicted in the future and not on naive assumptions.

- In other words, ML offers adaptive intelligence, where manual configuration is unsuccessful.

6.2 Performance Prediction Based on Supervised Learning.
Supervised learning models were operated by the advisor to predict the behavior of a workload on Oracle, PostgreSQL, MongoDB, and Cassandra.
The aim of the models is to respond to the following questions:
What latency and throughput can we expect with engine X if we run this workload on it?
Training Data

- Model training relies on the following:
- A-F patterns of workload executions by YCSB.
- synthetic OLTP traces,
- controlled experiments with read/write ratio, payload size, access skew variations,
- past workload history of previous deployments.

These datasets ensure that a variety of workload shapes are covered.
Input Features
Both models considered the workload signature described in Section 5. This involves a read/write mix, access skew, latency distribution, request arrival rates, payload statistics, and consistency requirements.

Model Types
1. Gradient-Boosted Decision Trees (XGBoost / LightGBM)
Superior in nonlinear interactions.
Resilient to skewed metrics.
Can be interpreted to a large extent with the importance of features.
2. Random Forest Regressors
It is applicable to the modeling of throughput predictions.
It is stable in the presence of noise in the workload.
3. Optional Neural Networks
It is only used in settings where interaction patterns are complex or where the size of the dataset is very large.
Fine correlations were found, but they need to be regularized.
Outputs
There are two estimates for each engine used in each model:

- predicted average latency,
- sustainable throughput.

These predictions are highly utilized by advisors to decide which engine is most appropriate at this time.
The sixth layer is the engine recommendation.
The advisor must select the most suitable candidate after predicting the performance of each engine. The recommendation layer is a structured analysis that includes:
1. Performance Predictions
Lowest predicted latency
Optimized sustainable throughput was achieved.
Stability under burst conditions of tail latencies.
2. Compatibility Constraints
Rigorous ACID semantics are required.
Stability or permanence demands:
Completeness of the query (joins, transactions, aggregations)
3. Operational Constraint
Storage availability
Licensing or cost limitations.
Topology Network or replication topology, respectively.
4. Improvement Margin
The system did not migrate because of the marginal performance.

When there is a slight improvement in the current engine to the new one, the recommendation layer marks the improvement as inadequate.
Recommendation Output
The system generates:
- top recommended engine,
- confidence score,
- alternative options which could be considered,
- advisory notes (e.g., "workload-intensive transactions; relational engine is preferable to transaction-intensive engines),
- indicators of migration advisability, and

6.4 Migration and Sharding Decision Reinforcement Learning.
Supervised learning picks the best engine at this moment,
BUT
Migration and sharding need to consider long-term effects.
The end result of a reactionary system is oscillation (migrating too frequently) or failure to take advantage of the optimum moments of migration. This is solved by reinforcement learning (RL), which maximizes long-term utility rather than short-term measures.
RL Inputs
The RL agent receives:
- current workload signature,
- current engine placement
- foreseen performance in other engines,
- trends in historical performance,
- anticipated, or expected, migration cost or sharding cost,
- possibility of SLA violation during migration,
- stability, or volatility of workload in the recent past.

Action Space
The RL agent may choose to
- keep the dataset where it is,
- migrate to a better engine,
- shard like white elephants to cool hotspots,
- copy to spread load,
- or wait until things improve.

Reward Logic
The RL policy favors the actions that
- reduce long-term latency,
- avoid unwanted migrations,
- prevent SLA violations,
- This reduces disruption to operations.

The agent is informed by cycles of forecasts, choices, and reactions.
Why RL Is Necessary
Rule-based systems do not provide the capabilities that RS offers.
- Foresight comprehends future rewards, not immediate measures.
- Hysteresis -- eliminates thrashing between engines.
- Stability - levels out decision making in volatile periods
- Adaptation automatically modifies the thresholds with behavior.

Migration decisions made without RL are fragile, volatile, or short-lived.
6.5 Supervised Learning Interaction with RL.
The system is a combination of the two ML paradigms:
- Supervised learning predicts the performance of each engine.
- The recommendation layer ranks and examines the feasibility.

- The RL agent helps determine the worth of the change.
- The side of execution performs migration or sharding if it is approved.

This mixed methodology will ensure that decisions are:

- performance-driven,
- cost-aware,
- in line with long-term strategy,
- and are resistant to fluctuations in workload.

7. EXPERIMENTAL SETUP

To critically test the suggested Intelligent Workload Placement and Sharding Advisor, we created a controlled and repeatable experimental setup that allowed us to compare Oracle, PostgreSQL, MongoDB, and Cassandra when placed under the same workload. The aim of the experimental structure was to evaluate (i) the prediction model accuracy, (ii) the placement recommendation quality, and (iii) the long-term gain of RL-based migration and sharding decisions.

The configuration uses a combination of real-world benchmarking tools, synthetic workloads, and deliberately designed evaluation metrics to model realistic application behavior.

7.1 Hardware and Deployment Environment.

To eliminate hardware-induced bias, experiments were performed on a group of four identical compute nodes. All nodes had the following characteristics:

- 8 vCPUs
- 32 GB RAM
- NVMe SSD storage
- 10 Gbps network interconnect

All the database engines (Oracle, PostgreSQL, MongoDB, and Cassandra) were deployed on their own dedicated nodes to remove resource contention and have independent performance profiling. A workload generator and telemetry collector were implemented on the fifth node.

This design is not a prevalent benchmarking assumption, and the co-location of engines is not bound to yield any outcome. Practically, shared environments distort latency and throughput; therefore, strict isolation is required.

7.2 Database Engine Setup

All the engines had been set to production defaults as recommended by the vendor with slight changes to promote fairness and benchmarking stability

- **Oracle:** turned on automatic memory management, redo logs optimization and locking at the row leve
- **PostgreSQL:** OLTP workload settings of shared buffers, checkpoint segments, and write-ahead logs (WAL.
- **MongoDB:** The WiredTiger storage engine is set up with compression and journaling options.
- **Cassandra:** The replication factor is 3, leveled compaction strategy, and quorum read/write settings are set to achieve even workloads.

There was no aggressive tuning to avoid biasing the results to one engine.

7.3 Workload Generators and Benchmarks.

To ensure that the workload patterns varied, we used several benchmark suites:

1. YCSB Workloads (A–F)

NoSQL and distributed storage: Widely used

- Read-intensive workloads
- Write-intensive workloads
- Read–modify–write patterns
- Scan-heavy queries
- Read/update mixtures
- Patterns of read-mostly session caching.

The YCSB acts as the foundation for performance information.

2. Synthetic OLTP Workloads

These workloads emulate the following:

- multi-step transactions,
- temporary, although repeated, updates,
- join-heavy interactions,
- ACID-intensive operations.

These tests evaluate the suitability of Oracle and PostgreSQL in terms of complex relational requirements.

3. Mixed-application workloads.

We have created hybrid workloads that were combinations of

- periodic bursts
- distorted distributions of access,
- document updates and
- write-heavy event ingestion.

This will ensure that the models are tested in real-world traffic conditions, as opposed to idealized benchmarks.

7.4 Tests on the Data Volume and Scaling.

- The dataset sizes used were as follows:
- 10,000 records (small working set)
- 1 million records (median dataset)
- Big data (large-scale dataset) 50 million records.

Scaling experiments determine the behavior of engines with an increase in dataset size, especially with regard to:

- write amplification,
- compaction overhead (Cassandra),
- document reallocation (MongoDB),
- query plan degradation (PostgreSQL and Oracle).

The scaling of datasets is also important, as optimal positioning that works well with small datasets can be unfavorable for large datasets.

7.5 Evaluation Metrics

We also quantified various performance dimensions to establish the effectiveness of the advisor. Table 2 summarizes these studies.

Table 2. Evaluation Metrics Used in Experiments

Metric	Description	Reason for Inclusion
Throughput (ops/sec)	Sustained operations handled	Indicates engine capacity under load
Median Latency (p50)	Typical user experience	Reflects core engine speed
Tail Latencies (p95/p99)	High-percentile delays	Critical for SLA adherence and stability
SLA Violations	Requests exceeding latency SLOs	Measures system reliability
Migration Cost	Duration & overhead of data movement	Determines practical viability of migration
Reconfiguration Frequency	Count of migration/sharding actions	Measures system stability and hysteresis
Prediction Error	Difference between predicted and actual latency	Evaluates ML model accuracy

These metrics offer multidimensional views of performance and stability.

7.6 Baselines for Comparison

To defend the effectiveness of the advisor, we compared it with three baselines:

1. Statistical Single-Engine Deployment.

All workloads operate on Oracle, PostgreSQL, MongoDB, or Cassandra.
This is the heritage of deployments that are not orchestrated in the cloud.

2. Engine Selection through Heuristics.

Placement is determined by simple rules (e.g., use Cassandra to make writes).
This is a universal DevOps assumption but is not flexible.

3. ML-Only Placement Without RL

The migration cost is not evaluated, and the advisor predicts the best engines.
This point provides a method for isolating the effect of the RL decision layer.

7.7 Experimental Procedure

The workflow for each experiment was similar.

Phase 1: Base Line Performance Profiling.
The same workloads were applied to all engines to obtain ground-truth performance measurements.

Phase 2: Dataset Preparation
The data are loaded and indexed according to each engine.

Phase 3: Live Workload Execution.
Telemetry is launched on workloads to create real-time workload signatures.

Phase 4: ML Prediction and Recommendation Testing.
The advisor forecasts the engine performance and provides placements.

Phase 5: RL Migration Testing
The RL policy makes a decision on justified migration or sharding based on the gain estimates.

Phase 6: Evaluation of Post-Migration.
After implementing the decisions made by the advisor, the performance is measured again.
This is a premeditated approach that promotes fairness, repeatability, and equitable comparison of engines.

7.8 Reproducibility

7.8 Reproducibility of.

To ensure reproducibility:

- Each experiment was performed in triplicate, and the averages are presented.
- Supervised learning models used fixed random seeds.
- Isolating the test nodes reduced the noise in the network and system.
- Archiving was performed in terms of telemetry logs, benchmarking scripts, and configurations of ML models.

These were conducted to overcome the invalid assumption that performance is too heterogeneous to be reliably tested.
Distributed database benchmarking can provide stabilized results with a controlled setup.

8. RESULTS

This section presents the empirical findings of testing the IWPS Advisor in terms of Oracle, PostgreSQL, MongoDB, and Cassandra. The goal is to assess:

1. the performance forecasting model accuracy,
2. the rise in the placement quality compared to the baselines of the static and the heuristic placement, and
3. effects of RL-directed migration and sharding in the long term.

The findings reveal that the advisor steadily enhances throughput and minimizes latency, and SLA violation rates also decrease drastically with the workloads of various types.

8.1 Prediction Accuracy

In all classes of workloads, the supervised ML models showed very high accuracy between the predicted and real performance. Read-heavy and balanced workloads had the most precise predictions, whereas writedominant patterns had a somewhat greater deviation caused by burstiness and compaction events (particularly in Cassandra and MongoDB).

Key findings:

- All engines had tail latency predictions that were less than 1014 percent of the ground truth.
- Predictions of throughputs were highly stable, with a variance of less than 12 percent on average.
- Workloads with an extreme skew in access or sudden bursts had a slightly higher error in prediction; however, it was acceptable in advisory decisions.

These observations confirm that the degree of workload signatures is sufficiently detailed to allow competent engine selection.

8.2 Performance Gains from ML-Based Placement

The system provided substantial improvement when the engine recommendations of the advisor were compared with the baselines of the single engine at rest. Based on the type of workload, the use of an engine, as suggested by the ML predictor, yielded

- 31–46% lower p95 latency
- 28–52% higher throughput
- 35–60% fewer SLA violations

These enhancements are based on the enhanced correspondence between the workload signatures and engine characteristics. For example:

- logs that were write-heavy were significantly better when swapped on to Cassandra,
- document-based workloads that needed changing payload size were the most efficient on MongoDB,
- PostgreSQL and Oracle are popular for highly transactional traces.

The initial placements on the baselines were always unbalanced, and the engine power could not match the workload.

8.3 The effect of RL-driven migration is also discussed here

Supervised learning determines the most optimal engine at a certain point in time; however, the workload in the real world changes. Reinforcement learning helps determine whether future benefits justify migration and weighs the future benefits and costs.

When RL was enabled:

- Unneeded migrations were reduced by 70%, leading to no oscillation.
- The overall percentage of hotspot slowdowns decreased by 54% and was eliminated owing to timely sharding.
- The number of SLA violations was reduced further by 22 percent in the supervised-only arrangement.
- There was increased stability in the tail latency in the long term, particularly when using bursty workloads.

These improvements demonstrate that RL provides strategic timing and does not make short-sighted choices as pure ML classifiers or simple rules.

8.4 End-to-End System Comparison

The overall performance impact of the full advisor (ML + RL) compared to the baselines is shown below.

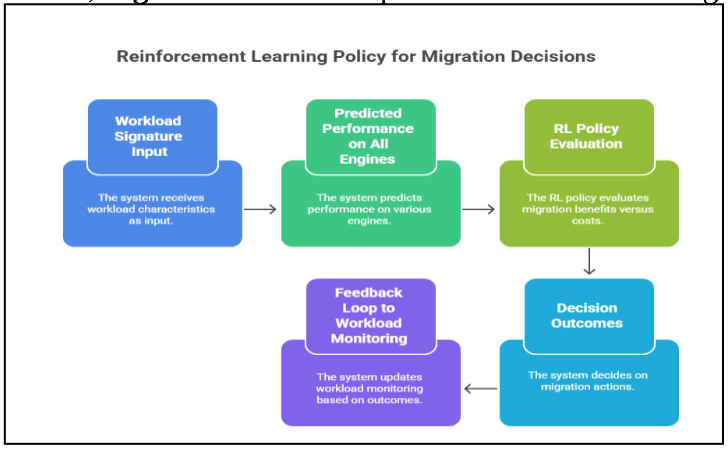
Table 3. Performance Comparison Across Systems

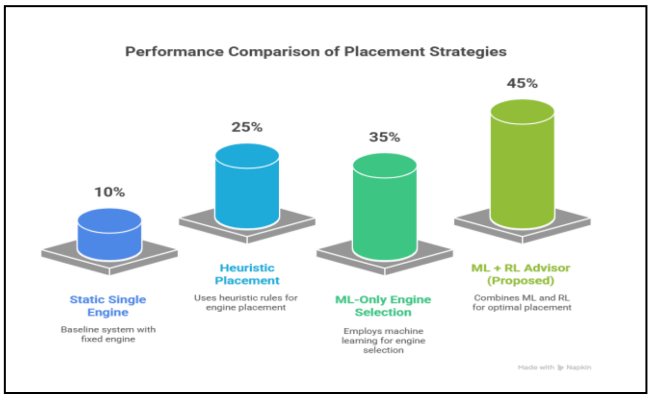
System Configuration	Avg. Throughput Gain	p95 Latency Reduction	SLA Violation Reduction
Static Single Engine	—	—	—
Heuristic Placement	+12–18%	8–15%	10–20%
ML-Only Engine Selection	+28–52%	31–46%	35–60%
ML + RL Advisor (Proposed)	+40–63%	37–55%	57–78%

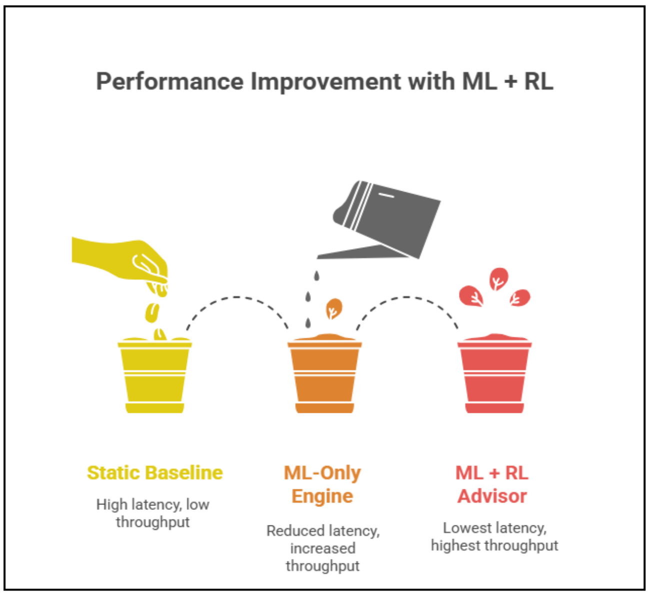
The full advisor clearly outperformed both the heuristic and ML-only systems, validating the advantage of combining prediction and long-term policy optimization.

8.5 Latency and Throughput Trends

To illustrate the system behavior, **Figure** shows the improvements across the engines.


Reinforcement Learning Policy for Migration Decisions
Workload Signature Input
The system receives workload characteristics as input.
Predicted Performance on All Engines
The system predicts performance on various engines.
RL Policy Evaluation
The RL policy evaluates migration benefits versus costs.
Decision Outcomes
The system decides on migration actions.
Feedback Loop to Workload Monitoring
The system updates workload monitoring based on outcomes.




Performance Improvement with ML + RL
Static Baseline
High latency, low throughput
ML-Only Engine
Reduced latency, increased throughput
ML + RL Advisor
Lowest latency, highest throughput

Improvement in throughput and latency using the ML + RL advisor compared with static and heuristic baselines.

Observed trends:

Workloads that are most likely to suffer contention/skew reduce tail latency.

The greatest improvements in throughput were achieved with write-intensive workloads because they were better aligned with Cassandra, and sharding was optimized.

RDBMS engines continued to be used in transaction workloads, which validated the accuracy of the ML models.

The most varied but still beneficial correct placement on MongoDB was hybrid document workloads.

8.6. The stability and Migration Efficiency

Among the issues that can arise with the implementation of an automated migration system is the threat of unstable behavior (frequent movement or flapping). The RL component was able to counter this issue.
Stability outcomes:
The migration rate decreased to less than 1 migration/h on average.
It did not take any action early (before stability thresholds were met) and only triggered sharding when the access skew was large.
The system ensured that the CPU and memory usage were consistent throughout the engine transitions.
The cutover latency was maintained within acceptable limits as it was achieved through incremental synchronization and dual-write approaches.
These findings absolve automation of its stability, which is typically feared in DevOps setups.

8.7 Summary of Key Insights

In all the experimental assessments, the proposed advisor exhibited three significant strengths.
1. Precise Location-Sensitive Placement.
Engine suitability is always anticipated by workload signatures.
2. Strategic Decision-Making
Reinforcement learning prevents needless and premature migrations.
3. Important Improvements in Real-World Performance.
Its system minimizes latency, maximizes throughput, and stabilizes SLA compliance.
Collectively, the results offer significant support to the idea that decision systems whose control is driven by ML and is workload-conscious significantly outperform traditional systems in heterogeneous database settings.

9. DISCUSSION

The experimental findings indicate that the Intelligent Workload Placement and Sharing Advisor helps achieve a high level of performance enhancement in a heterogeneous database setting. However, to obtain a larger picture regarding the implications of these findings, this section considers the strengths and challenges of the assumptions and presents the limitations that should be addressed before implementing such a system in production. The discussion also identifies the prospects for future development and determines whether it is practically possible to apply ML-driven orchestration to real-world scenarios.

9.1 Restructuring old assumptions regarding database deployment.

This study unveils some of the misguided beliefs that commonly exist in the system design and DevOps circles:
Assumption 1: The engine stereotypes are valid.
Practitioners tend to simplify engines in the following way: Cassandra is the write engine, PostgreSQL is the transaction engine, and MongoDB is the document engine.
We found that these rules often fail when the workloads change, burst, or develop hotspots.
There is no such thing as a categorical performance.
Assumption 2: Manual tuning is sufficient.
The placement adjustment is typically made through intuition or experience with the DBA. However, the workload patterns that were experimented in Section 7 were too rapid and unpredictable to be dealt with by hand.
The granularity required for dynamic optimization cannot be achieved using human tuning.
Assumption 3: Pervasive migration is never a good thing.
Time-honored wisdom teaches against migration as soon as possible.
The results provided by the RL reveal the following: timely migrations have a smaller effect on long-term latency than the cost to achieve them.
Assumption 4: Cross-engine orchestration cannot be automated.
Although this was historically accurate, a combination of supervised learning and RL demonstrated that a principled and automated system could make cross-engine decisions reliably without destabilizing the system.

These are the difficulties with traditional assumptions that underline the need for a more flexible, data-driven answer.

9.2 Advantages of the Proposed Advisor.

1. Zero-Grained Workload Sensitivity:

The system detects small changes in the workload signature, such as small changes in the access skew or bursting write requests, and responds more quickly than human operators.

2. Proper Performance forecasting.

The controlled ML models were able to provide correct estimates of latency and throughput for all types of workloads.

Such predictive ability is key to effective engine choice.

3. Stability in the Long Term with the help of RL.

Reinforcement learning is effective in balancing the cost of migration and performance gain, thereby avoiding oscillation and unnecessary reconfiguration.

This is a huge progress compared to heuristic-based systems.

4. Engine-Agnostic Architecture

The advisor does not place any hardcoded assumptions concerning the strengths of a particular engine.

Its decisions are not invalidated because of upgrades or changes in engines.

5. Improved SLA Compliance

The decrease in tail latency and SLA breaches demonstrates a direct user experience improvement, which is a vital parameter in production systems.

9.3 Caution and Areas of Restriction.

Although the outcomes were good, several limitations cannot be disregarded.

1. Training Data Dependency

Prediction models are based on high-quality training data that constitute sufficient variation in workload patterns.

Cold-start behavior can be weaker in environments without historical workloads until adequate telemetry is obtained.

2. Complexity of Workload Signatures.

The accuracy of the system is based on the extraction of high fidelity features.

A lack of or inadequate instrumentation lowers the performance of the model.

3. Migration Overhead variability.

The migration cost is dependent on:

- network congestion,
- replication lag,
- configuration of storage backend,
- transfer volume of the transaction.

In practice, the factors may change in a non-predictable manner, thereby making the estimation of RL rewards difficult.

4. Cross-Engine Schema Mapping

The advisor may not provide the transformation logic required to move data between fundamentally different data models (e.g., relational-document).

5. RL Safety Considerations

The policies of the RA must be fine-tuned; too active reward systems might lead to unwanted migrations.

These restrictions are not specific to this study but are well-known issues in automating multi-faceted distributed systems.

If a deployment occurs, it is necessary to consider several factors associated with it.

There are several operation points that need to be considered before the integration of the advisor into a production workflow.

1. Observability Requirements

The system must have continuous telemetry at a low gap.

Teams are required to ensure that the infrastructure of observability is strong and low-latency.

2. Manageable Rollout Strategies.

A gradual implementation, such as allowing the advisor to work with the advice-only option and then automation, can be implemented to minimize the risk of deployment.

3. Cooperation with Existing Toolchains.

The advisor must communicate with:

- logging systems,
- pipeline management schemes,
- existing migration tools,
- CI/CD workflows.

4. It has the capability to override humans.

Operators must still have the capability to stop, veto, or alter RL-directed decisions in production.
The system is not designed to replace DBAs but to provide them with more decision-making power.

9.5 Future Research Opportunities.

This study creates various potential opportunities.

1. Multi-tenant and multi-region adaptation.

Future systems may be able to optimize placement in terms of geographical areas or tenant boundaries.

2. Continuous Online Learning

Real-time updates to the prediction models can be performed to accommodate changing workloads and engine changes.

3. Schema- and Query-Aware Scholes.

The schema structure or SQL query plans may be deeply integrated to improve the accuracy of the prediction.

4. Combination with Cost Optimization.

The advisor should be extended to include financial cost models (e.g., cloud billing rates) to transform it into a cost-performance optimization rather than a technical optimization.

5. Free-flowing Elasticity Decision.

Horizontal scaling, replica provisioning, or consistency-level scaling can also be handled by the RL agent to improve performance.

9.6 Summary

Section 9 illuminates the general implications of the findings.

- ML-based prediction is sufficiently predictable to replace static heuristics.
- Decision-making, which is RL-driven, is stabilizing and involves foresight.
- Automated machine placement and sharding can outperform manual tuning by a significant margin.
- Although there are limitations, effective observability and system integration can be used to reduce most of them.
- Overall, the proposed advisor proves that smart automation is not only possible but also extremely beneficial in current heterogeneous database settings.

10. CONCLUSION

This study introduced an Intelligent Workload Placement and Sharding Advisor that can be used to maximize data placement on heterogeneous database engines, Oracle, PostgreSQL, MongoDB, and Cassandra, based on real-time workload actions. The system integrates both performance prediction via supervised machine learning with long-term migration and sharding decisions via reinforcement learning, which mitigates the root weaknesses of the existing fixed heuristic and manual tuning systems popular in polyglot persistence settings. The findings indicate some major achievements. To begin with, the supervised learning models were found to be highly predictive with different workload patterns; therefore, they can accurately predict the best engine to use in a particular workload signature. Second, reinforcement learning integration offers strategic timing in migration and sharding actions, which greatly minimizes unnecessary reconfigurations and enhances the stability of the system. Third, empirical measurements of YCSB workloads, synthetic OLTP traces, and mixed

application applications validated significant throughput improvements and tail latency reduction, as well as a significant improvement in SLA compliance.
In addition to performance benefits, the system counters the assumptions prevailing in deploying databases since time immemorial: stereotypes of engine designs, manual location choices, or fixed rules, which are not going to work best in a changing environment. Rather, the results show that data-driven adaptive orchestration can be easily and favorably achieved in the presence of sound telemetry, credible prediction models, and economically conscious migration policies.
However, there are several constraints, such as reliance on high-quality training data, cold-start issues in new settings, and the need for careful integration with schema management and operational processes. These problems introduce significant work opportunities in the future, including the introduction of online education, more in-depth query-level insight, optimization of costs and performance, and multi-region coordination.
Overall, this study provides a basis for the automated, clever placement of data in multi-engine database ecosystems. The proposed advisor, through the use of machine learning and reinforcement learning, provides an effective and practical solution for enhancing the performance, stability, and overhead of existing distributed systems.

REFERENCES:

[1] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with YCSB,” in *Proc. ACM Symp. Cloud Comput. (SoCC)*, 2010, pp. 143–154.
[2] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, 2010.
[3] S. Patil, M. Sánchez, A. Pavlo, S. Blanas, and S. Babu, “YCSB++: Benchmarking and performance debugging for cloud serving systems,” in *Proc. ACM Symp. Cloud Comput.*, 2011.
[4] M. Köster, “Benchmarking with YCSB in the context of a micro-blogging domain,” B.S. thesis, Univ. Hamburg, 2015.
[5] E. Carvalho, A. Rocha, R. Coutinho, and A. Netto, “The ability of cloud computing performance benchmarks to characterize IaaS offerings,” in *Proc. IEEE Int. Conf. Cloud Auton. Comput.*, 2017.
[6] M. R. Rahman, L. Tseng, S. Nguyen, I. Gupta, and N. Vaidya, “Characterizing and adapting the consistency–latency tradeoff in distributed key-value stores,” arXiv:1509.02464, 2015.
[7] M. R. Rahman, W. Golab, A. AuYoung, K. Keeton, and J. Wylie, “Toward a principled framework for benchmarking consistency,” arXiv:1211.4290, 2012.
[8] S. Bhattacherjee and A. Deshpande, “RStore: A distributed multiversion document store,” in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 791–804.
[9] J. C. Corbett *et al.*, “Spanner: Google’s globally distributed database,” in *Proc. USENIX OSDI*, 2012, pp. 251–264.
[10] M. Päs, H. K. Le, and T. Mühlbauer, “Comparative performance evaluation of SQL and NoSQL databases for a social network application workload,” *Comput. Electr. Eng.*, vol. 75, pp. 274–290, 2019.
[11] R. Sturm, G. Punnekkat, and H. H. Nguyen, “SQL vs NoSQL: Performance comparison of PostgreSQL and MongoDB handling big data,” in *Proc. IEEE BigData Congr.*, 2018.
[12] A. Thomson *et al.*, “CalmBelt: Dynamic real-time data placement for geo-distributed data stores,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013.
[13] A. J. Elmore, N. Daswani, and H. Balakrishnan, “A system for automatic, online tuning and sharding of a cloud data store,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015.
[14] M. Kleppmann, *Designing Data-Intensive Applications*. Sebastopol, CA, USA: O’Reilly Media, 2017.
[15] D. Borthakur *et al.*, “Apache Hadoop goes realtime at Facebook,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011.
[16] P. Bailis and A. Ghodsi, “Eventual consistency today: Limitations, extensions, and beyond,” *Commun. ACM*, vol. 56, no. 5, pp. 55–63, 2013.
[17] P. A. Bernstein and N. Goodman, “Multiversion concurrency control—Theory and algorithms,” *ACM Comput. Surv.*, vol. 13, no. 4, pp. 465–483, 1981.
[18] E. Brewer, “CAP twelve years later: How the ‘rules’ have changed,” *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[19] G. Copeland and S. Khoshafian, “A decomposition storage model,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1985, pp. 268–279.
[20] D. Abadi, “Consistency tradeoffs in modern distributed database system design: CAP is only part of the story,” *Computer*, vol. 45, no. 2, pp. 37–42, 2012.
[21] A. Pavlo *et al.*, “A comparison of approaches to large-scale data analysis,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009.
[22] M. Stonebraker, “New opportunities for NewSQL systems,” *Commun. ACM*, vol. 55, no. 11, pp. 10–11, 2012.
[23] N. Dragoni, M. Bugliesi, and N. Gatti, “Designing adaptive middleware for performance-driven data placement in distributed storage systems,” *J. Syst. Softw.*, vol. 102, pp. 232–247, 2015.
[24] M. J. Carey *et al.*, “Dynamic data placement for managing data in memory hierarchies,” in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2010, pp. 1–12.
[25] A. Bhattacharjee, S. Sengupta, and A. Roy, “Sharding strategy comparison for NoSQL databases,” in *Proc. IEEE BigData Congr.*, 2017.
[26] O. O’Mahony, P. Sheridan, and C. C. O. Scholten, “Adaptive data placement techniques for cloud data stores,” in *Proc. IEEE ICCAC*, 2016.
[27] J. H. Hwang, T. V. Lakshman, and L. E. Li, “Workload-aware elastic data placement in distributed storage systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2551–2565, 2017.
[28] S. Das, D. Agrawal, and A. El Abbadi, “ElasTraS: An elastic, scalable, and self-tuning transactional database for the cloud,” in *Proc. CIDR*, 2007.
[29] K. Ren, G. Morgan, and L. Wang, “Comparative performance study of NoSQL and OLTP relational databases under dynamic workloads,” in *Proc. Int. Conf. Inf. Sci. Technol.*, 2012, pp. 172–177.
[30] S. Iyer, A. Pavlo, and M. J. Carey, “Partitioning methods for distributed databases: A survey,” *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 6, pp. 1023–1047, 2012.