

QUBO-Driven Divergent Design Index Tuning for Distributed Databases

Sam Bird
School of Computer Science
University of Oklahoma
Norman, OK, USA
sam.bird@ou.edu

Le Gruenwald
School of Computer Science
University of Oklahoma
Norman, OK, USA
ggruenwald@ou.edu

Sven Groppe
Institute of Computer Science
TU Bergakademie Freiberg
Freiberg, Germany
sven.groppe@informatik.tu-freiberg.de

Abstract—Cloud databases frequently employ replication to improve performance through parallel processing. Divergent design tuning (DDT)—the joint optimisation of index configurations and query routing functions across replicas in a distributed database—is critical for query performance but intractable at scale. Existing classical design heuristics struggle to navigate the large, densely-connected combinatorial search space, while machine learning approaches require recommendation times on the order of hours or days even for modest problem sizes. To overcome these computational barriers, we reformulate divergent design tuning as a single quadratic unconstrained binary optimisation (QUBO) problem, jointly optimising index configurations, query routing, and replica-failure constraints. We present ADDA, an annealing-based divergent design advisor, which solves this QUBO using annealing-based samplers, obtaining high-quality divergent designs in a reasonable runtime. We give theoretically motivated QUBO penalty constants with correctness guarantees and analyse ADDA’s structural resource requirements. We evaluate ADDA with two annealing implementations—classical simulated annealing and path integral Monte Carlo—on the TPC-H and TPC-C benchmarks across multi-replica clusters. We find that ADDA more than halves query workload execution time over state-of-the-art DDT algorithms and reduces recommendation time by 99% compared to reinforcement-learning-based approaches.

Index Terms—index selection, replicated database, mathematical optimisation

I. INTRODUCTION

An important consideration in database administration is the creation of indexes to improve query performance. The challenge of choosing the best indexes is the *index selection problem*, which is known to be NP-hard [1]. Given a workload W of queries, the index selection problem is to choose some indexes $\mathcal{I}$ to construct on the database that minimises the execution cost of the query workload, subject to a constraint on the memory consumed by the chosen indexes. Numerous heuristic, optimisation, and machine learning approaches to the index selection problem have been proposed [2].

Modern cloud databases, such as Google Cloud SQL and Amazon RDS, allow the deployment of readable secondary database replicas, to offload the processing of analytical queries from the primary [3], [4]. These systems create full database replicas across several nodes, allowing database

administrators to create a divergent set of indexes on each read replica, optimising each one for a different subset of read queries. A crucial challenge in creating these divergent index configurations is the joint optimisation of both physical design (what indexes to create on each replica) and query routing (which replica processes each incoming query). This is the *divergent design tuning* (or ‘DDT’) problem. To avoid overloading a single replica and mitigate tail-latencies, state-of-the-art algorithms evaluate optimal performance by minimising the maximum replica execution cost across the cluster. Despite its growing relevance, maximum cost DDT is not well-studied theoretically.

A major challenge that arises in developing an algorithm for DDT is the problem’s tractability. The search space for solutions to even centralised index selection grows faster than exponentially with the number of candidate indexes [5], before considering the added complexity of deciding where queries ought to be routed. Consequently, algorithms attempting to solve DDT must balance solution quality with an acceptable recommendation time. Existing machine learning approaches for DDT employ a reinforcement learning (RL) framework that rely on thousands of repeated, expensive invocations of the DBMS query optimiser’s ‘what-if’ cost estimator [6], [7]. This causes RL recommendation times to scale poorly, often becoming impractical even for modest clusters of 4–7 replicas. Conversely, alternative heuristic approaches maintain tractability only by trading away significant solution quality.

To overcome these computational barriers, we reformulate the joint indexing and routing constraints of maximum-cost DDT as a single quadratic unconstrained binary optimisation (QUBO) problem [8]. This reformulation directly targets both limitations of existing approaches. Because indexing and routing variables are optimised jointly within one objective, the solver can exploit interactions between index placement and query routing that decoupled heuristics cannot see. We evaluate ADDA using both classical simulated annealing [9] and path integral Monte Carlo (PIMC) annealing [10], and show both produce high-quality recommendations at practical recommendation times, demonstrating the advantage of our approach.

In this paper, we present the following contributions:

- A general definition for maximum cost DDT that incor-

porates the possibility of replica failure

- A formal proof that maximum cost DDT is NP-hard
- ADDA, an annealing-based divergent design advisor, a QUBO formulation of DDT
- Theoretically motivated penalty constants for the quadratic program
- An empirical evaluation showing ADDA scales to multi-replica clusters, significantly outperforming state-of-the-art baselines DINA [6], qDINA [7], and DiversityClusterDB [11] on the TPC-H and TPC-C benchmarks

The remainder of this paper is structured as follows. Section II gives a background on divergent design tuning and annealing-based optimisation. Section III discusses existing algorithms for DDT. Section IV proves that divergent design tuning is NP-hard. Section V introduces ADDA, an end-to-end algorithm for DDT. Section VI discusses our implementation of ADDA and compares its performance against existing DDT algorithms. Finally, Section VII summarises our findings and discusses directions for future research.

II. BACKGROUND

To begin, we give a background on the divergent design tuning problem and introduce our new definition for DDT on a maximum cost basis. Then, we discuss different approaches to mathematical optimisation to contextualise our algorithm’s approach to solving DDT.

A. Divergent Design Tuning

The problem of divergent design tuning was first posed in 2012, when the DIVGDESIGN algorithm was proposed [12]. This definition was extended in [13] to incorporate the possibility of replica failure, as well as a broader set of possible constraints on the recommended design. Later work has since restated the problem in terms of minimising the processing time for a distributed workload, allowing the replicas to work in parallel [11].

We illustrate the optimisation abstraction used in DDT. A workload of incoming queries is received by a query dispatcher (or router) inside the primary DBMS. The dispatcher consults a DDT optimiser, which jointly determines (i) an index configuration for each replica and (ii) a query-to-replica routing policy. The underlying system consists of a set of homogeneous read-only replicas that each maintain the same logical database state as the primary but may differ in their physical design structures. Each read query is executed on exactly one selected replica, and its cost is estimated using a standard what-if query optimiser cost model conditioned on the replica’s physical design. To isolate the physical design problem, the underlying replication mechanism (e.g., consistency maintenance and update propagation) is abstracted away and is not part of the optimisation problem objective. The DDT optimiser operates periodically or offline and outputs a configuration that is then deployed to the system.

Recent work on divergent design tuning defines the problem as the minimisation of the maximum execution cost on any one individual replica, reducing the time required to execute a full

query workload [7], [11]. We refer to this as the ‘maximum cost basis’ of DDT. Earlier work aimed to minimise the total cost, across all of the replicas, of executing some query workload [13], which we refer to as the ‘total cost basis’. This paper discusses the maximum cost basis, as it is the problem formulation used in recent DDT literature and is the preferred metric in other areas of distributed database research [14]. Despite its prevalence, existing work does not provide a formal definition of maximum cost DDT that incorporates the possibility of replica failure and a general class of user-specified constraints on the divergent design.

Before giving the definition, it is necessary to introduce some terminology. Let $\text{cost}(q, \mathcal{I})$ be the cost of executing a query q , given that an index configuration $\mathcal{I}$ is present on the database. An index configuration $\mathcal{I}$ is a set of indexes on the database tables. We assume that the database is fully replicated across n machines. We denote the set of replicas as R , one present on each node. Let Q be the set of analytical queries (which can be executed on any replica) and U be the set of update queries. While we defer the mechanism for enforcing consistency to the distributed DBMS, we account for the impact of the chosen indexes on the update workload to model the cost of the replication mechanism updating the indexes on each read replica. m is the *routing multiplicity factor*, the number of relatively low-cost replicas to which each query could be routed. h is a routing function that determines which replica a query is sent to; for instance, $h(q) = r$ determines that query q is routed to replica r . $f(q)$ is a function that weights the relative importance of each query. Finally, α is the probability one of the replicas fails, which we assume is uniform across all of the replicas.

The divergent design tuning problem, then, is to find an index configuration $\mathcal{I}_r$ for each replica r and a routing function h that minimises the maximum time it takes a replica to process the queries which are routed to it. We express this quantity as $\text{MaxCost}(\mathcal{I}, h)$, where $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_n\}$ is the set of index configurations. The cost for each replica is the cost of executing the entire update workload, plus the cost of executing the portion of the analytical workload routed to that replica. We also divide the cost of each query in the analytical workload by m , as if a query may be routed to m replicas, the probability it is actually sent to one of those replicas is $1/m$.

Additionally, we want to account for the possibility a replica fails. Let $\text{FMaxCost}(\mathcal{I}, h, j)$ be the maximum replica processing cost if the j th replica fails. We need to develop a new routing function to account for the fact queries cannot be sent to the failed replica. We denote this new routing function h_j , and call the original routing function (where no replicas have failed) h_0 . FMaxCost is similar to MaxCost , but accounts for the new routing function h_j and the possibility that $m = |R|$, in which case m becomes $|R| - 1$ (to reflect the new maximum number of replicas in the cluster). We make the modelling assumption that our recommendations must account for at most one replica failing, and further failures require a new recommendation. This is done to keep the problem size

tractable, which we discuss further in Section V-E.

When a replica fails, queries will be routed to different replicas, but the index configurations on those replicas do not change, as the failed replica may come back online and rebuilding the indexes is expensive. Accordingly, we want the index configurations to account for the possibility that they will be processing queries that would not be sent to each replica under normal operation. We denote the probability of a single replica in the cluster failing as α . Increasing α induces the recommended indexes to be more responsive to queries that *might* be routed to a given replica in the event another fails, while decreasing α maximises the performance when no failures occur at the expense of processing time when a failure does occur.

Finally, we want to consider the case where the user wishes to state additional constraints on the physical design of the system. A common constraint is to impose an index space budget on each replica, so the indexes materialised do not take up more than a predetermined amount of space. We denote these additional constraints as C , and allow the user to specify any constraint as long as it is expressible in a binary integer program.

Now, we are ready to give a formal definition for maximum cost DDT.

Definition 1. *Given a query workload $W = Q \cup U$, a weight function f , full database replicas R , a set of constraints C , a routing multiplicity factor m , and a failure probability α , the **divergent design tuning problem on a maximum cost basis** is to find a set of index configurations $\mathcal{I}_r$ and routing functions $h : Q \mapsto R^m$ that solves*

$$\begin{aligned} \min_{\mathcal{I}, h} \quad & (1 - \alpha) \cdot \text{MaxCost}(\mathcal{I}, h_0) \\ & + \sum_{j \in R} \frac{\alpha}{|R|} \text{FMaxCost}(\mathcal{I}, \mathbf{h}, j) \\ \text{subject to} \quad & C \end{aligned}$$

Where

$$\begin{aligned} \text{MaxCost}(\mathcal{I}, h_0) = \\ \max_{r \in R} \left(\sum_{\{q | h_0(q) = r\}} \frac{f(q)}{m} \text{cost}(q, \mathcal{I}_r) + \sum_{u \in U} f(u) \text{cost}(u, \mathcal{I}_r) \right) \end{aligned}$$

$$\begin{aligned} \text{FMaxCost}(\mathcal{I}, \mathbf{h}, j) = \\ \max_{r \in R \setminus \{j\}} \left(\sum_{\{q | h_j(q) = r\}} \frac{f(q)}{\min\{m, |R| - 1\}} \text{cost}(q, \mathcal{I}_r) \right. \\ \left. + \sum_{u \in U} f(u) \cdot \text{cost}(u, \mathcal{I}_r) \right) \end{aligned}$$

h_j is the routing function when the j th replica has failed;
 h_0 is the routing function when no replicas have failed.

In the remainder of this paper, when we refer to the divergent design tuning problem, we understand it to mean DDT in the maximum cost basis, as in the above definition.

B. QUBO Problems and Annealing

The coupled, discrete nature of joint index selection and query routing under a maximum-cost objective yields a highly non-convex and discontinuous optimisation landscape. Traditional exact mathematical programming solvers (e.g., IBM CPLEX) rely on branch-and-bound or branch-and-cut algorithms that struggle to navigate these rugged cost surfaces efficiently, often experiencing exponential runtime scaling even for modest numbers of replicas. To achieve practical recommendation times, we must trade exact global optimality for high-quality solutions by leveraging annealing-based samplers.

To employ certain annealing algorithms, it is necessary to reduce the combinatorial problem into a standardised mathematical target known as a *Quadratic Unconstrained Binary Optimisation* (QUBO) problem. Mathematically, a QUBO instance is stated as $\min_{x \in \{0,1\}^n} x^T Q x + c^T x$, where x is an n -dimensional vector of binary decision variables, $c \in \mathbb{R}^n$ is a vector of linear coefficients, and $Q \in \mathbb{R}^{n \times n}$ is a matrix containing quadratic coefficients between variables. Any linear equality or inequality constraints within the original problem space (such as forcing a query to route to exactly one replica) are mapped into the objective function as quadratic penalty terms that increase the total energy (cost) of invalid states. The fundamental advantage of a QUBO formulation is its direct mathematical isomorphism to the Ising spin-glass model in statistical mechanics, allowing the problem to be solved using algorithms that simulate physical systems.

The most traditional method for solving QUBO problems classically is simulated annealing (SA). SA navigates the optimisation landscape by mimicking thermal fluctuations; it iteratively perturbs the binary solution vector x , using a decreasing temperature schedule to probabilistically accept higher-cost states to escape poor local minima by climbing over energy barriers. Other heuristics, such as the *Path Integral Monte Carlo* (PIMC) sampler [10], attempt to simulate quantum tunnelling effects to move through energy barriers in the cost landscape. While these algorithms differ in how they solve QUBOs, they share the same fundamental advantage in obtaining good quality solutions to large combinatorial optimisation problems with a faster runtime than exact solvers.

III. RELATED WORK

With a background on DDT and QUBO methods established, we next discuss existing approaches to database index selection. Despite a wealth of existing research on centralised index tuning advisors [2], the divergent design tuning field remains smaller by comparison. DIVGDESIGN [12] and RITA [13] model DDT on a total cost basis as a binary integer program and use classical solvers to provide recommendations. DRLinda [15] and DINA [6] use deep reinforcement learning to learn a good set of indexes, then route each query to the replica that has the lowest execution cost. qDINA [7]

extends DINA's RL architecture with a gate-based quantum neural network, but inherits the same search-space-scaling limitations. Finally, DiversityClusterDB [11] uses a cluster-and-tune heuristic to iteratively improve a divergent design, which can in principle use any centralised index selection algorithm. Of these, DIVGDESIGN and RITA are designed for the total cost basis, while DiversityClusterDB, DRLinda, DINA, and qDINA are designed for the maximum cost basis.

The existing maximum cost DDT algorithms suffer from the same problem: they struggle to scale beyond a small number of database replicas. In the case of DRLinda and DINA, the growing number of possible index configurations to consider requires more reinforcement learning actions to fill the space budgets and complete a learning episode, leading to a proportional increase in the already substantial recommendation time. In DiversityClusterDB's case, the quality of the heuristic recommendation simply becomes poor, with some replicas having no queries routed to them at all. This scaling issue poses a serious problem to deploying these algorithms in real-world environments, as large clusters of database replicas are common for big data applications [16].

QUBO methods have been successfully applied to centralised index selection, in the context of quantum algorithms. Trummer et al. [17] demonstrate that quantum annealing can find high-quality index configurations faster than classical solvers on benchmark workloads, establishing the viability of QUBO-based formulations for index selection. Kesarwani et al. [18] extend this to gate-based hardware. However, these algorithms rely on specialised quantum hardware with limited resources and constrained problem embeddings, making them impractical for immediate deployment.

IV. NP-HARDNESS OF DIVERGENT DESIGN TUNING

Next, we present a proof that the divergent design tuning problem on the maximum cost basis is NP-hard. Existing work has already demonstrated that solving DDT on the total cost basis is NP-hard; the theorem is introduced in [13] and proved in [19]. However, the subset-sum reduction in that proof does not extend to the maximum cost basis, so we provide a separate reduction now.

Theorem 1. *Solving the divergent design tuning problem stated in Definition 1 is NP-hard.*

Proof. We prove NP-hardness by reduction from the makespan minimisation problem on identical machines ($P||C_{\max}$), which is shown to be NP-hard in [20]. Consider the case where $\alpha = 0$ (there is no possibility of replica failure), $U = \emptyset$ (the workload is only composed of analytical queries), $f(q) = 1$ for all $q \in Q$, $m = 1$, and $\mathcal{I} = \emptyset$ (there are no index candidates, so the cost of each of the queries is not reduced by any index configuration). Further, let there be no additional constraints C . The problem then becomes a restricted case of divergent design tuning, namely, assigning queries to replicas to minimise the maximum cost of a particular replica's workload.

Let $c_q = \text{cost}(q, \emptyset)$ be the cost of executing a query on any replica and $t_q^r \in \{0, 1\}$ be a decision variable encoding h_0

that denotes whether the q th query is routed to the r th replica (that is, $t_q^r = 1$ if $r \in h_0(q)$, or 0 otherwise). The restricted DDT then becomes

$$\min_t \quad \max_{r \in R} \sum_{q \in Q} c_q t_q^r \quad (1a)$$

$$\text{subject to} \quad \sum_{r \in R} t_q^r = 1 \quad \forall q \in Q \quad (1b)$$

Let R_r be the execution costs of the queries assigned to the r th replica; that is, $R_r = \{c_q : t_q^r = 1\}$, and $S = \{R_1, \dots, R_{|R|}\}$ be the schedule of execution costs. The finish time of one replica r processing queries is then $F_r(S) = \sum_{q \in R_r} c_q$, and the restricted divergent design tuning problem is then to minimise the finish time of the entire schedule, $F(S) = \max_{r \in R} \{F_r(S)\}$, subject to the requirement that every task (query) is scheduled (routed to one replica), as required by (1b). Under this definition, the objective in (1a) is exactly $F(S)$.

Taken together, these statements are the makespan minimisation problem shown in [20]. This construction is computable in polynomial time. Accordingly, makespan minimisation reduces in polynomial time to divergent design tuning on a maximum cost basis. Therefore, DDT is at least as hard as makespan minimisation and thus NP-hard. $\square$

The NP-hardness result is not surprising, given that the same is true of both total cost DDT and centralised index selection. However, the reduction to makespan minimisation is a noteworthy insight into the challenge posed by this problem. The routing component of DDT is equally as difficult as the design component of selecting indexes, which can render static routing heuristics such as those used in DINA [6], qDINA [7], and DRLinda [15] insufficient for obtaining good solutions. Accordingly, we look to a joint optimisation technique to solve the design and routing problem together.

V. PROPOSED ALGORITHM

In this section, we present our divergent design advisor, ADDA. ADDA models divergent design tuning as a binary integer program (BIP), then reduces it to a QUBO that can be solved efficiently using annealing-based samplers. Doing so requires the constraints in the BIP to be converted to penalty terms in the objective function of the QUBO, so penalty coefficients must be chosen to enforce these constraints.

The remainder of this section is structured as follows. Section V-A describes the inputs to the integer program. In Section V-B, we state the full BIP formally. Section V-C shows the reduction to the QUBO. Section V-D states the penalty constants for each of the constraint terms and proves they are sufficient to maintain the optimality and feasibility of the program. Section V-E analyses how the number of variables in the QUBO scales with the problem size. Finally, Section V-F gives an end-to-end algorithm for solving DDT.

A. Input Parameters

We begin by defining three sets of binary variables:

- $x_i^r = 1$ if the i th index candidate is present on the r th database replica, or 0 otherwise.
- $t_q^r = 1$ if the q th query is routed to the r th database replica, or 0 otherwise. Put another way, $t_q^r = 1$ if and only if $r \in h_0(q)$.
- $t_q^{r,j} = 1$ if the q th query is routed to the r th database replica if the j th replica has failed. Equivalently, $t_q^{r,j} = 1$ if and only if $r \in h_j(q)$. These variables are only used for the case where replica failures are possible, and may be omitted if $\alpha = 0$.

Additionally, we introduce the following sets of integer constants. These are provided as inputs to the binary integer program; the manner in which the algorithm produces them is discussed further in Section V-F.

- c_w is the cost of executing a statement $w \in W$ on a replica with no indexes present: $c_w = \text{cost}(w, \emptyset)$.
- v_i^w is the impact, or the *marginal benefit* of the i th index candidate on the execution cost of statement w : $v_i^w = \text{cost}(w, \{x_i\}) - \text{cost}(w, \emptyset)$.

Finally, we require the following parameters provided by the database administrator, which were defined in Section II-A.

- m is the routing multiplicity factor.
- α is the probability that a database replica fails.

B. Binary Integer Program for DDT

Now, we are ready to present a binary integer program that solves the problem given in Definition 1. Let $\text{pcost}(w, \mathcal{I}_r)$ be the predicted cost of executing a statement w on a replica with index configuration $\mathcal{I}_r = \{i \mid x_i^r = 1\}$. To predict the query cost, we start with the statement's cost when no indexes are present, then sum the marginal benefits of all of the indexes on that query. We express this predicted cost as $\text{pcost}(w, \mathcal{I}_r) = c_w + \sum_i v_i^w x_i^r$. Predicting the cost of a statement from the sum of the marginal benefits of the indexes is a standard modelling assumption used in optimisation-based approaches to index selection [13], [21], but we demonstrate empirically in Section VI that these estimates yield good results. Additionally, we require $\text{pcost}(w, \mathcal{I}_r) \geq 0$, as a negative cost to execute a query is not physically representable.

Let $\text{qcost}(q, \mathcal{I}_r)$ be the cost of executing a query q on replica r with an index configuration $\mathcal{I}_r$. We wish to account for the weight function f and multiplicity factor m , as in the MaxCost expression in Section II-A. Additionally, we want to ensure that the cost is 0 if the query is not routed to the current replica. We express this cost as $\text{qcost}(q, \mathcal{I}_r) = \frac{f(q)}{m} \cdot \text{pcost}(q, \mathcal{I}_r) \cdot t_q^r$.

Similarly, let $\text{ucost}(u, \mathcal{I}_r)$ be the cost of executing an update statement u on replica r with an index configuration $\mathcal{I}_r$. The multiplicity factor and routing variable can be removed, as update statements are sent to every replica. We express this cost as $\text{ucost}(u, \mathcal{I}_r) = f(u) \cdot \text{pcost}(u, \mathcal{I}_r)$.

Finally, we must account for the cost when a replica j has failed. Update costs are still processed on every functional replica, so no change to the update cost function is

needed. However, the query cost function must incorporate the additional variables $t_q^{r,j}$ to account for the backup routing function $h_j(q)$. Let $\text{fqcost}(q, \mathcal{I}_r, j)$ be the cost of executing a query q on replica r with an index configuration $\mathcal{I}_r$ when replica j has failed. We express this cost as $\text{fqcost}(q, \mathcal{I}_r, j) = \frac{f(q)}{\min\{m, |R|-1\}} \cdot \text{pcost}(q, \mathcal{I}_r) \cdot t_q^{r,j}$.

The binary integer program that solves the divergent design tuning problem given in Definition 1 is as follows:

$$\begin{aligned} \min \quad & (1 - \alpha) \max_{r \in R} \left(\sum_{q \in Q} \text{qcost}(q, \mathcal{I}_r) \right. \\ & \left. + \sum_{u \in U} \text{ucost}(u, \mathcal{I}_r) \right) \end{aligned} \quad (2a)$$

$$+ \sum_{j \in R} \frac{\alpha}{|R|} \max_{r \in R \setminus \{j\}} \left(\sum_{q \in Q} \text{fqcost}(q, \mathcal{I}_r, j) \right. \\ \left. + \sum_{u \in U} \text{ucost}(u, \mathcal{I}_r) \right)$$

$$\text{subject to} \quad \sum_{r \in R} t_q^r = m \quad \forall q \in Q \quad (2b)$$

$$\sum_{r \in R} t_q^{r,j} = \min\{m, |R| - 1\} \quad \forall q \in Q \quad (2c)$$

additional constraints C stated by the user (2d)

C. QUBO Reduction

We have presented a binary integer program that solves DDT, but as stated it cannot be used for path integral Monte Carlo annealing. We apply the same reduction when using simulated annealing, avoiding the need for bespoke constraint-respecting search moves and ensuring both samplers operate on an identical formulation.

The first step is to ensure the program is quadratic. We assume that all of the user-specified constraints C are at most quadratic. The stated objective in (2a)–(2d) is not, however, as the max terms are present in the objective function, so it must undergo reduction to be linearised. Then, we must remove the constraints and replace them with penalty terms. Finally, if adding the constraints to the objective function as penalty terms has introduced terms of an order higher than 2, we must reduce them to quadratic through additional auxiliary variables.

The initial problem that we encounter with the BIP in (2a)–(2d) is the maximum operator in the objective function, which is not quadratic. A standard linearisation strategy is to replace the minimum-of-maximum operator with an objective that minimises some arbitrary value, then introduce constraints to require that arbitrary value to be the maximum of the original values, described in Section 2.4 of [22]. To apply this method to a binary integer program, we introduce sets of *slack variables* $\mathbf{z} = \sum_{k=0}^{\lceil \log Z_{\max} \rceil} 2^k z_k$ to represent any value up to $Z_{\max}$. We then restate (2a)–(2d) as:

$$\min (1 - \alpha) \cdot \mathbf{z}^{(0)} + \sum_{j \in R} \frac{\alpha}{|R|} \cdot \mathbf{z}^{(j)} \quad (3a)$$

$$\text{subject to } \mathbf{z}^{(0)} \geq \sum_{q \in Q} \text{qcost}(q, \mathcal{I}_r) + \sum_{u \in U} \text{ucost}(u, \mathcal{I}_r) \quad (3b)$$

$$\forall r \in R$$

$$\mathbf{z}^{(j)} \geq \sum_{q \in Q} \text{fqcost}(q, \mathcal{I}_r, j) + \sum_{u \in U} \text{ucost}(u, \mathcal{I}_r) \quad (3c)$$

$$\sum_{r \in R} t_q^r = m \quad \forall q \in Q \quad (3d)$$

$$\sum_{r \in R} t_q^{r,j} = \min\{m, |R| - 1\} \quad \forall q \in Q \quad (3e)$$

$$\text{additional constraints } C \text{ stated by the user} \quad (3f)$$

Choosing $Z_{\max}$ is difficult because it must encode the maximum replica cost, which is generally not known in advance. The choice of $Z_{\max}$ is left as an input parameter for the algorithm, discussed further in Section V-D.

Next, we need to remove the program's constraints. There are several constraints (3b)–(3f) in the binary integer program, and we wish to restate them as penalty terms in the objective function. The reduction of the constraints to penalty terms is widely used, and a summary is given in Equations (3) and (4) of [23]. The technique converts a constraint of the form $A\mathbf{x} \leq b$ to a penalty term of the form $\lambda(A\mathbf{x} - b + s)^2$, where λ is a large positive constant and $s = \sum_j 2^j s_j$ are slack variables allowing the optimiser to reduce the penalty to zero if the inequality is not violated.

$$\min (1 - \alpha) \cdot \mathbf{z}^{(0)} + \sum_{j \in R} \frac{\alpha}{|R|} \cdot \mathbf{z}^{(j)} \quad (4a)$$

$$+ \lambda^{(r)} \left(\mathbf{z}^{(0)} - \sum_{q \in Q} \text{qcost}(q, \mathcal{I}_r) - \sum_{u \in U} \text{ucost}(u, \mathcal{I}_r) - \mathbf{s}^{(r)} \right)^2 \quad \forall r \in R \quad (4b)$$

$$+ \lambda^{(j,r)} \left(\mathbf{z}^{(j)} - \sum_{q \in Q} \text{fqcost}(q, \mathcal{I}_r, j) - \sum_{u \in U} \text{ucost}(u, \mathcal{I}_r) - \mathbf{s}^{(j,r)} \right)^2 \quad \forall j \in R, r \in R \setminus \{j\} \quad (4c)$$

$$+ \lambda^{(q)} \left(\sum_{r \in R} t_q^r - m \right)^2 \quad \forall q \in Q \quad (4d)$$

$$+ \lambda^{(f)} \left(\sum_{r \in R} t_q^{r,j} - \min\{m, |R| - 1\} \right)^2 \quad \forall q \in Q \quad (4e)$$

$$+ \lambda^{(c)} \cdot \text{reduced constraints } C \text{ stated by the user} \quad (4f)$$

This equation must undergo further reduction as (4b), (4c), and potentially (4f) are not quadratic. This situation arises because the conversion of an inequality constraint to a penalty term increases the order of the variables, but the inequality constraint may already be quadratic. The solution is to apply a further reduction, such as the one found in Equation 15 of [24]. Typically, the reduction is done automatically in software, dynamically selecting the best method for the given problem structure [25]. As the specific reduction chosen will have the same optimal solution and feasible region, it does not give further insight into the problem structure, so we treat the reduction as an implementation detail and omit the fully-reduced QUBO.

D. Penalty Constant Selection

In converting the constrained quadratic program given in (3a)–(3f) to an unconstrained one in (4a)–(4f), we have introduced large, positive constants λ to enforce the penalty terms. In theory, any choice of λ that guarantees a particular constraint will be greater than the objective will ensure that an infeasible solution will never be the optimal one. In practice, however, annealers are heuristic samplers that are not guaranteed to return the global optimum in finite time: they return low-energy solutions that may differ from the true optimum by some gap determined by the annealing schedule and energy landscape. Accordingly, we must select values for λ that shape the energy landscape in such a way that low-energy, near-optimal solutions sampled by the annealer correspond to valid and high-quality divergent designs.

First, we will consider the theoretical bounds for each λ term in the quadratic program.

Theorem 2 (Penalty constants). *Let Ω be a value that is greater than the maximum cost of any divergent design; that is $\Omega \geq \max\{\text{MaxCost}(\mathcal{I}, h), \text{FMaxCost}(\mathcal{I}, h, j)\}$ for all $\mathcal{I}, h, j$. Let $\varepsilon_C > 0$ be the smallest nonzero value that a constraint term (4b), (4c), (4d), (4e), or (4f) could take before multiplication by their respective penalty constants $\lambda^{(C)}$. Additionally, let $Z_{\max} \geq \Omega$, so that every feasible objective value is representable in the objective function (4a). An optimal solution to (4a)–(4f) is an optimal solution to (2a)–(2d) if $\lambda^{(C)} > \frac{\Omega}{\varepsilon_C}$.*

Proof (sketch). The proof is by contradiction and follows in two parts. First, we show that any optimal solution $\mathbf{x}^*$ to (4a)–(4f) must be feasible in (2a)–(2d). Suppose that there is a contradiction: then, at least one constraint term has energy at least ε_C .

Meanwhile, any feasible $\mathbf{x}^f$ has all constraint terms zero, and since $Z_{\max} \geq \Omega$ ensures $\mathbf{z}^{(0)}, \mathbf{z}^{(j)} \leq \Omega$ at feasibility, $E(\mathbf{x}^f) \leq \Omega$. If $\lambda^{(C)} > \Omega/\varepsilon_C$, then $E(\mathbf{x}^*) > \Omega \geq E(\mathbf{x}^f)$, contradicting the optimality of $\mathbf{x}^*$. So $\mathbf{x}^*$ is feasible.

Second, a feasible optimal $\mathbf{x}^*$ is also optimal in (2a)–(2d): if some feasible $\mathbf{x}^\dagger$ achieved a strictly lower objective there, then since both solutions have all constraint terms zero, their QUBO energies equal their original objective values, so $E(\mathbf{x}^\dagger) < E(\mathbf{x}^*)$, contradicting the optimality of $\mathbf{x}^*$ in (4). $\square$

The choice of Ω is left as an input parameter for the algorithm. The theorem will be satisfied as long as Ω is greater than $\text{MaxCost}(\mathcal{I}, h)$ and $\text{FMaxCost}(\mathcal{I}, \mathbf{h}, j)$ for all $\mathcal{I}, \mathbf{h}, j$. However, finding those largest values is a difficult problem in and of itself. Instead, we rely on the following approximation for use in the algorithm: $\Omega = \left(\sum_{q \in Q} f(q) \cdot \text{pcost}(q, \emptyset) \right) + (|R| \cdot \sum_{u \in U} f(u) \cdot \text{pcost}(u, \emptyset))$.

To initialise the slack variables $\mathbf{z}$ in (4a)–(4c), we also set $Z_{\max} = \Omega$. The intuition motivating this equation is that each query must be considered for use on m replicas (eliminating the term, as $m \cdot \frac{f(q)}{m} = f(q)$), while each update statement must be run on $|R|$ replicas. Here, Ω is equal to the divergent design cost when no indexes are present and each query is run once. This choice of Ω may not satisfy the assumptions stated in Theorem 2 in all cases. However, it is a practical upper bound on reasonable workloads where indexes reduce query execution costs, and we validate empirically in Section VI that ADDA produces valid, high-quality divergent designs under this approximation.

Now, we will discuss the empirical choices for each λ . As stated previously, values for λ that meet the requirements in the above theorem will guarantee that the optimal solution to the appropriate QUBO problem is the optimal solution to DDT. In practice, the solutions sampled from an annealer are unlikely to be the optimal ones. If the λ terms are not well considered, these solutions may be infeasible.

There are four sets of constraints: (4b) and (4c), to enforce the maximum operators; as well as (4d), (4e), and (4f), to enforce routing multiplicity and other user-specified constraints. The former set of constraints are introduced to linearise the objective function: if they are violated, the QUBO becomes only an approximation of the objective function in (2a)–(2d). The latter set of constraints enforce the feasibility of the solution with respect to the initial integer program. Accordingly, while we can tolerate a small violation of the former constraints, we cannot tolerate any violation of the latter. To guarantee that any violation of the feasibility constraints is penalised more than a violation of the maximum operator constraints, we set

$$\lambda^{(r)} = \lambda^{(j,r)} = \Omega \quad (5)$$

$$\lambda^{(q)} = \lambda^{(f)} = \lambda^{(C)} = \Omega^3 + 1 \quad (6)$$

The rationale motivating the selection of $\Omega^3 + 1$ in (6) follows from the maximum value that the constraints (4b) and (4c) can take. Those constraints could take values of at most Ω (when $\mathbf{z} = \Omega$ and $\text{qcost}(q, \mathcal{I}_r) = \text{ucost}(q, \mathcal{I}_r) = \mathbf{s} = 0$). In the QUBO, the constraint value is squared and multiplied by $\lambda^{(r)}$ or $\lambda^{(j,r)}$, as the case may be. Accordingly, for $\lambda^{(q)}, \lambda^{(f)}, \lambda^{(C)}$, we require $\lambda > \frac{\Omega^3}{\varepsilon}$. For our binary integer programs with integer coefficients, $\varepsilon = 1$ by construction in all three cases, so (6) satisfies this condition.

E. Scaling Analysis

To perform annealing-based optimisation, it is necessary to sample over the entire solution space, which grows with the

number of variables. Accordingly, the recommendation time of our algorithm scales with the number of variables. Our original binary integer program for DDT introduced decision variables for index configurations and routing functions (both default and failure-aware). Additionally, we have introduced slack variables to linearise the maximum operator, convert the constraints, and reduce the equation to a polynomial.

Theorem 3 (Scaling). *Let $|\mathcal{U}|$ be the number of candidate indexes, $|R|$ be the number of replicas in the cluster, $|Q|$ be the size of the analytical query workload, and $B = \lceil \log Z_{\max} \rceil$, the number of variables required to encode $Z_{\max}$ in a binary encoding. To solve the divergent design tuning problem on a maximum cost basis, ADDA requires*

$$B + |\mathcal{U}||R| + |Q||R| + |R||\mathcal{U}||Q| + B|R|$$

variables when $\alpha = 0$ (there is no possibility of failure), plus whatever additional variables are required to encode the user-specified constraints C .

If $\alpha > 0$ and failure routing must be considered, ADDA requires an additional

$$B|R| + |Q||R|(|R| - 1) + |R|(|R| - 1)|\mathcal{U}||Q| + B|R|(|R| - 1)$$

variables.

Proof. The variable count arises as follows. The $\mathbf{z}^{(0)}$ term requires B variables, by definition. There are $|\mathcal{U}||R|$ x_i^r terms, one for each index candidate and replica. There are $|Q||R|$ t_q^r terms, one for each query in the query workload and replica. The final two terms occur for every replica in the penalty term (4b), which accounts for the $|R|$ term in both. First, in order to ensure that squaring that term does not introduce interactions of a higher order than quadratic, we must introduce auxiliary variables $w_{i,q}^r = x_i^r t_q^r$ to apply the reduction in [24]. This requires an auxiliary variable for each of the $|\mathcal{U}||Q|$ combinations of the x, t terms. Finally, each penalty term introduces its own set of slack variables $\mathbf{s}^{(r)}$, for an additional B variables per replica.

Next, we consider the failure terms. For each replica, $|R| - 1$ failure costs are considered, so there are in total $|R|(|R| - 1)$ failure penalty terms to consider in (4c). The first two counts are from the new sets of variables introduced. Each $\mathbf{z}^{(j)}$ adds B variables, while there are new failure routing variables $t_q^{r,j}$ for each of the $|Q|$ queries in the $|R|(|R| - 1)$ replicas in failure scenarios. The final two terms are additional auxiliary variables $w_{i,q}^{j,r} = x_i^r t_q^{r,j}$ and slack terms $\mathbf{s}^{(j,r)}$, for which the same counting logic as the non-failure terms applies. $\square$

As an example, consider the TPC-H benchmark: $|\mathcal{U}| = 36$ and $|Q| = 22$, and we assume that $B = 13$ can suffice. If no replica failures are possible, TPC-H requires $863|R| + 13$ variables. If replica failures are possible, then TPC-H requires an additional $827|R|^2 - 814|R|$ variables.

F. End-to-End Algorithm

Finally, we give a complete algorithm for divergent design tuning that can produce the integer coefficients necessary for the QUBO’s optimisation. This is stated in Algorithm 1 below.

First, line 1 creates the universe of candidate indexes, $\mathcal{U}$. We derive this from the predicates of the queries, as these are the columns we may be interested in searching over. Next, we obtain the baseline query costs and the marginal index benefits. Lines 2–4 invoke the cost estimator to obtain the baseline costs when no indexes are present, then lines 5–9 iterate over every query and index candidate to compute the marginal benefits. Then, the annealing step is taken in line 10, which provides the solution to the DDT problem instance. Finally, the solution is parsed in lines 11–14.

Algorithm 1: ADDA

Data: Replicas R , workload $W = Q \cup U$, penalty constant Ω , additional constraints C
Result: Index configurations $\mathcal{I}$, routing tables $\mathbf{h}$

```

1  $\mathcal{U} \leftarrow$  columns referenced in the predicates of  $Q$ ;
2 for  $w \in W$  do
3    $c_w \leftarrow \text{cost}(w, \emptyset)$ ;
4 end
5 for  $i \in \mathcal{U}$  do
6   for  $w \in W$  do
7      $v_i^w \leftarrow \text{cost}(w, \{i\}) - c_w$ ;
8   end
9 end
10 Optimise the QUBO given in (4a)–(4f) via desired annealing method;
11  $\mathcal{I}_r \leftarrow x_i^r \quad \forall i \in \mathcal{U}, r \in R$ ;
12  $h_0 \leftarrow \arg \max_r t_q^r \quad \forall q \in Q$ ;
13  $h_j \leftarrow \arg \max_r t_q^{r,j} \quad \forall q \in Q, j \in R, r \in R \setminus \{j\}$ ;
14 return  $\mathcal{I}, \mathbf{h}$ 

```

VI. PERFORMANCE EVALUATION

Next, we analyse the performance of ADDA against competing algorithms for DDT. Section VI-A describes the experiments we conduct. Section VI-B evaluates the algorithms’ scalability as the number of replicas in the cluster grows, increasing the problem size and attendant search space. Section VI-C evaluates the performance of the algorithms under increasing probability of a single-replica failure occurring, comparing ADDA’s recommended fallback routing functions over a random rerouting policy.

A. Experimental Setup

To compare the performance of several DDT algorithms, we ran experiments on the standard TPC-H [26] and TPC-C [27] benchmarks. For TPC-H, we generated 50 queries from each of the 22 TPC-H query templates, for a total workload of 1,100 queries. For TPC-C, we ran the five provided transactions 10 times each to generate a workload of 59 analytical queries and

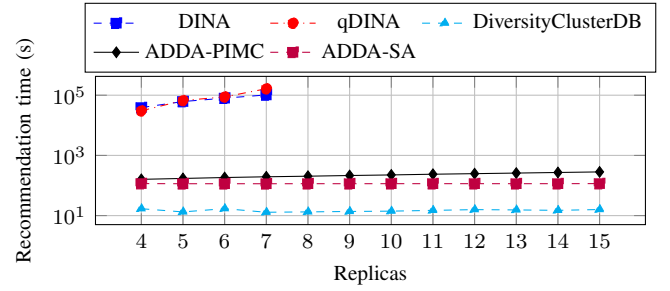


Fig. 1: TPC-H recommendation time (1,100 queries), by number of replicas (log scale).

65 update queries. In both cases, we populated each replica with 10 GB of data.

We compared the performance of five DDT algorithms: two variants of ADDA, DINA [6], qDINA [7], and DiversityClusterDB [11]. We ran our experiments using `c6620` nodes in CloudLab [28]. Each ADDA execution sampled 10 reads from the annealer and selected the solution with the lowest energy value. We denote ADDA with simulated annealing as ‘ADDA-SA’, and ADDA with the path integral Monte Carlo sampler as ‘ADDA-PIMC’. Because DiversityClusterDB does not support DDT problems with an update workload, we excluded it from our experiments on the TPC-C benchmark. Additionally, while other work has included comparisons to DIVGDESIGN and RITA [6], [11], it is not appropriate to make such a comparison when discussing an algorithm that solves maximum cost DDT. As discussed in Section II-A, DIVGDESIGN and RITA attempt to solve DDT on a total cost basis, which leads to a fundamentally different optimisation objective. Accordingly, any comparison of query execution times while the replicas process in parallel would be misleading. Finally, we limit the recommendation time of the DDT algorithms to 100,000 seconds (about 28 hours), which limited our analysis of DINA and qDINA past 7 replicas. In these cases, the recommendation time tradeoff is already unacceptably large, so any theoretical performance gain would be outweighed.

B. Scalability and Solution Quality

We first evaluate how DDT algorithms scale as the number of replicas increases. Existing approaches exhibit a trade-off between recommendation time and divergent design quality. DINA and qDINA require several hours to compute a recommendation, whereas DiversityClusterDB provides recommendations quickly but yields similarly poor execution performance. In contrast, ADDA achieves both fast recommendation times and high-quality divergent designs. Across all tested configurations, ADDA completes in minutes while more than halving workload execution time on average compared to the competing approaches.

On the TPC-H benchmark, the ADDA variants consistently achieve the lowest query execution times (Figure 2) while keeping recommendation time below 300 seconds (Figure 1). This recommendation time is 2-3 orders of magnitude faster

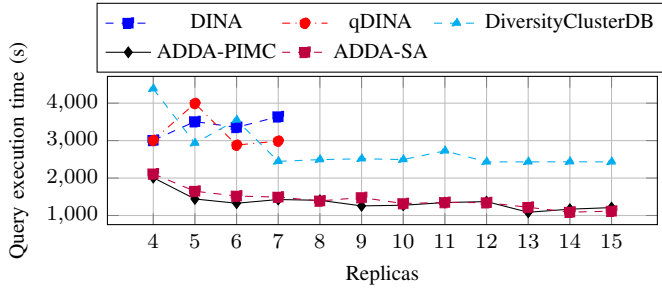


Fig. 2: TPC-H query workload execution time (1,100 queries), by number of replicas.

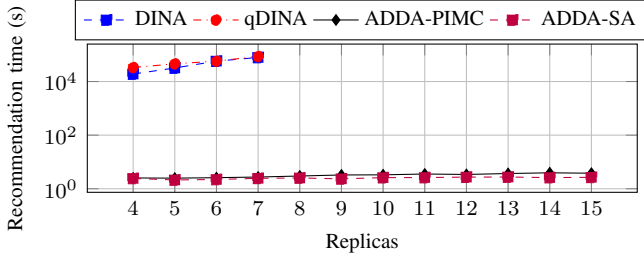


Fig. 3: TPC-C recommendation time (124 queries), by number of replicas (log scale).

than DINA or qDINA. On 7 replicas, ADDA-PIMC takes 194 seconds to produce a divergent design, where qDINA requires 45 hours. The difference in query execution time is substantial, achieving an execution time improvement of at least 31% (on 4 replicas) to as much as 61%, more than doubling the query throughput. While DiversityClusterDB produces faster recommendations, taking 17–22 seconds to run, the quality of its divergent designs is erratic, and consistently worse than those produced by either ADDA variant. While the improvement over competing state-of-the-art DDT algorithms is substantial, the difference between variants is not substantial. ADDA-PIMC reduces query execution time by a modest 4.54% on average over ADDA-SA, but this benefit is variable across the number of replicas, with no clear winner emerging. Additionally, the runtime of ADDA-SA does not appreciably increase with problem size, an advantage of its simplicity over the quantum-inspired heuristic in PIMC. Taken together, these results suggest that the more complex PIMC sampler may be able to better navigate the optimisation and produce slightly better results, but this comes at the expense of additional recommendation time.

On the TPC-C benchmark, the difference in query execution times is smaller in absolute terms (Figure 4), as the simplicity of the queries allows them to execute quickly. In relative terms, however, the difference remains substantial. On average, the ADDA variants yield a 67% reduction in query execution time over DINA and a 74% reduction over qDINA. The recommendation time difference is similar to the TPC-H case, with DINA and qDINA requiring 5–24 hours to execute while ADDA-SA produces a recommendation in about 7 seconds,

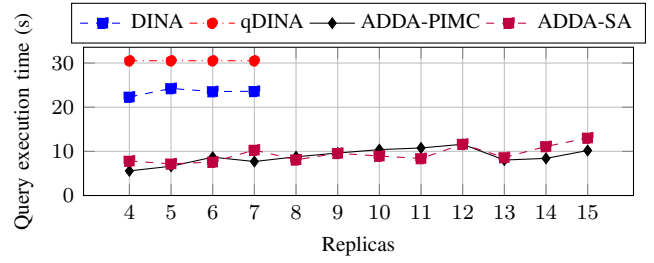


Fig. 4: TPC-C query workload execution time (124 queries), by number of replicas.

increasing to about 10 for PIMC (Figure 3). It is noteworthy that in this case, qDINA consistently fails to produce any meaningful divergent design recommendation, converging to a degenerate policy that repeatedly adds and removes the same index. This demonstrates that reinforcement-learning-based approaches struggle to explore the search space effectively as problem complexity increases. ADDA, on the other hand, is still able to recommend good divergent designs as the search space grows, regardless of the annealing method used.

The results suggest that ADDA achieves superior query execution performance because it jointly optimises index selection and query routing. In the QUBO formulation, routing variables and index-selection variables are optimised simultaneously, enabling both solvers to capture the interaction between index placement and routing decisions. Existing approaches such as DINA, qDINA, and DiversityClusterDB instead determine index configurations independently of routing and assign queries only after indexes have been selected. This decoupled optimisation limits their ability to exploit index–routing interactions, resulting in less effective divergent designs.

C. Failure-aware routing

We next evaluate the benefit of failure-aware routing specifically. DINA, qDINA, and DiversityClusterDB have no mechanism for rerouting queries away from a failed replica, so for these baselines we reroute affected queries to healthy replicas at random. Figure 5 shows query execution time with seven replicas in the cluster as the probability α of a single-replica failure increases. Averaged across the tested range of α , the ADDA variants reduce query workload execution time by an average of 29% over DiversityClusterDB, 32% over qDINA, and 49% over DINA. Similarly to the TPC-H results, the variants differ only slightly in performance, with PIMC improving over SA on query execution time by 7.67% while increasing recommendation times from 2.5 to 4 seconds. However, ADDA-SA degrades more gracefully than ADDA-PIMC as α approaches 1, suggesting ADDA-SA has an advantage over PIMC as the optimisation landscape becomes more highly constrained.

These results align with the scaling analysis discussed in the previous subsection. As ADDA’s QUBO formulation can jointly consider index selection, routing decisions, and fallback rerouting decisions, it can recommend divergent designs that

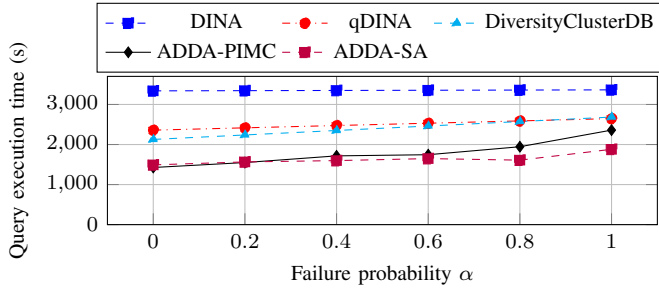


Fig. 5: TPC-H query workload execution time (1,100 queries) when one replica out of seven fails with probability α .

are robust to replica failure more effectively than relying on random rerouting and disjoint routing heuristics.

VII. CONCLUSION & FUTURE WORK

We studied the problem of reducing the maximum processing time across replicas in a database cluster with multiple read-only secondaries. We formalised this problem as divergent design tuning on a maximum cost basis and proved that it is NP-hard. We presented ADDA, an annealing-based divergent design advisor, which jointly optimises the index configuration and query routing function to provide a good divergent design. We showed our process for reducing DDT to a binary integer program, then a quadratic unconstrained binary optimisation problem that can be solved via annealing.

Our experiments show that ADDA achieves both fast recommendation times and high-quality divergent designs across multiple workloads using either annealing sampler, generating recommendations in minutes rather than hours while more than halving query execution times. We also compared the performance of ADDA against competing algorithms under increasing probability that a single-replica failure scenario occurs, demonstrating ADDA's optimised rerouting tables maintain a good distribution of queries across multiple replicas, substantially improving query execution times. These results show the advantage of ADDA's QUBO formulation, allowing for the joint optimisation of index placement and query routing decisions that decoupled heuristics struggle to navigate.

Future work includes extending ADDA to solve total-cost divergent design tuning. Another direction is extending the optimisation framework to include additional physical design choices such as materialised views or partitioning.

REFERENCES

- [1] D. Comer, "The difficulty of optimum index selection," *ACM Trans. Database Syst.*, vol. 3, no. 4, pp. 440–445, Dec. 1978.
- [2] Y. Wu, X. Zhou, Y. Zhang *et al.*, "Automatic Index Tuning: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 12, pp. 7657–7676, 2024.
- [3] "Create and manage indexes on read replicas | Cloud SQL for MySQL | Google Cloud Documentation." [Online]. Available: <https://docs.cloud.google.com/sql/docs/mysql/replication/read-replica-indexes>
- [4] "Amazon RDS Read Replicas." [Online]. Available: <https://aws.amazon.com/rds/features/read-replicas/>
- [5] G. Valentin, M. Zuliani, D. C. Zilio *et al.*, "DB2 Advisor: An Optimizer Smart Enough to Recommend its own Indexes." IEEE Computer Society, Feb. 2000, pp. 101–101.
- [6] Z. Sadri and L. Gruenwald, "A Divergent Index Advisor Using Deep Reinforcement Learning," in *Proc. DEXA '22*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 139–152.
- [7] S. Bird, L. Gruenwald, and S. Groppe, "A Divergent Index Tuning Advisor Using Quantum Machine Learning for Distributed Databases," in *Proceedings of the 2026 IEEE 42nd International Conference on Data Engineering Workshops*. Montreal, Canada: IEEE Computer Society, May 2026, pp. 1–7.
- [8] G. Kochenberger, J.-K. Hao, F. Glover *et al.*, "The unconstrained binary quadratic programming problem: a survey," *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 58–81, Jul. 2014.
- [9] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, Feb. 1993.
- [10] A. D. King, J. Raymond, T. Lanting *et al.*, "Scaling advantage over path-integral Monte Carlo in quantum simulation of geometrically frustrated magnets," *Nature Communications*, vol. 12, no. 1, p. 1113, Feb. 2021.
- [11] H. Hang, X. Tang, B. Zhou *et al.*, "Unlocking the Power of Diversity in Index Tuning for Cluster Databases," in *Proc. DEXA '24*. Cham: Springer Nature Switzerland, 2024, pp. 185–200.
- [12] M. P. Consens, K. Ioannidou, J. LeFevre *et al.*, "Divergent physical design tuning for replicated databases," in *Proc. SIGMOD '12*. New York, NY, USA: Association for Computing Machinery, 2012, pp. 49–60.
- [13] Q. T. Tran, I. Jimenez, R. Wang *et al.*, "RITA: an index-tuning advisor for replicated databases," in *Proc. SSDM '15*. New York, NY, USA: Association for Computing Machinery, 2015.
- [14] V. Rosenfeld, S. Breß, and V. Markl, "Query Processing on Heterogeneous CPU/GPU Systems," *ACM Comput. Surv.*, vol. 55, no. 1, pp. 11:1–11:38, Jan. 2022.
- [15] Z. Sadri, L. Gruenwald, and E. Leal, "DRLIndex: deep reinforcement learning index advisor for a cluster database," in *Proc. IDEAS '20*. New York, NY, USA: Association for Computing Machinery, 2020.
- [16] J. Wang, H. Wu, and R. Wang, "A new reliability model in replication-based big data storage systems," *Journal of Parallel and Distributed Computing*, vol. 108, pp. 14–27, Oct. 2017.
- [17] I. Trummer and D. Venturelli, "Leveraging Quantum Computing for Database Index Selection," in *Proc. Q-Data '24*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 14–26.
- [18] M. Kesarwani and J. R. Haritsa, "Index Advisors on Quantum Platforms," *Proc. VLDB Endow.*, vol. 17, no. 11, pp. 3615–3628, Jul. 2024.
- [19] J. LeFevre, "Physical Design Tuning Methods for Emerging System Architectures," Ph.D. dissertation, University of California, Santa Cruz, Jun. 2014.
- [20] E. Horowitz and S. Sahni, "Exact and Approximate Algorithms for Scheduling Nonidentical Processors," *J. ACM*, vol. 23, no. 2, pp. 317–327, Apr. 1976.
- [21] S. Halfpap, "Hybrid Index Selection Using Integer Linear Programming Based on Cached Cost Estimates of Heuristic Approaches," in *Proceedings of the 1st Workshop on Simplicity in Management of Data*, ser. SiMoD '23. New York, NY, USA: Association for Computing Machinery, Jul. 2023, pp. 1–4.
- [22] M. Asghari, A. M. Fathollahi-Fard, S. M. J. M. Al-e hashem *et al.*, "Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey," *Mathematics*, vol. 10, no. 2, Jan. 2022.
- [23] K. Yonaga, M. J. Miyama, and M. Ohzeki, "Solving Inequality-Constrained Binary Optimization Problems on Quantum Annealer," Dec. 2020, arXiv:2012.06119 [quant-ph].
- [24] K. Jun and H. Lee, "HUBO formulations for solving the eigenvalue problem," *Results in Control and Optimization*, vol. 11, p. 100222, Jun. 2023.
- [25] L. Schmidbauer, K. Wintersperger, E. Lobe *et al.*, "Polynomial Reduction Methods and their Impact on QAOA Circuits," in *2024 IEEE International Conference on Quantum Software (QSW)*, Jul. 2024, pp. 35–45.
- [26] Transaction Processing Performance Council (TPC), "TPC Benchmark H (Decision Support) Standard Specification Version 3.0.1," 2022. [Online]. Available: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf
- [27] —, "TPC Benchmark C Standard Specification Revision 5.11," 2010. [Online]. Available: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf
- [28] D. Duplyakin, R. Ricci, A. Maricq *et al.*, "The Design and Operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14.