Evaluating a Quantum Solution: How hard can it be?

Daniel Aarao Reis Arturi
McGill University
Montreal, Canada
daniel.aaraoreisarturi@mail.mcgill.ca

Emilie Fahber
McGill University
Montreal, Canada, Canada
emilie.fahber@mail.mcgill.ca

Stefanie Scherzinger
University of Passau
Passau, Germany
stefanie.scherzinger@uni-passau.de

Bettina Kemme
McGill University
Montreal, Canada
bettina.kemme@mcgill.ca

ABSTRACT

The database community has begun to express a wide range of database problems as QUBO formulations, which show promise of being solved on quantum computers. However, understanding the complexity of translating the abstract problem into programs that can actually run on quantum computers, finding a proper methodology to fine-tune the configuration parameters when communicating with the quantum engine, and understanding the results and the potential bottlenecks are non-trivial endeavors with which the standard DB researcher is unfamiliar. This paper presents an experience report where we have implemented an existing QUBO solution for the data allocation problem with various quantum library interfaces and evaluated the programs on D-Wave's quantum annealer hardware and two simulators. We discuss programming, parameter, and execution complexity, outline the results we obtained, and discuss how both measurements and results differ from traditional DB evaluations. The goal is to give newcomers to this field insight into how complex it is to implement and evaluate DB quantum solutions, and to outline the current challenges.

VLDB Workshop Reference Format:
Daniel Aarao Reis Arturi, Emilie Fahber, Stefanie Scherzinger, and Bettina Kemme. Evaluating a Quantum Solution: How hard can it be?. VLDB 2026 Workshop: QC&DKM 2026 - 2nd Workshop on Quantum Computing and Data/Knowledge Management.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/DISLMcGill/EvaluatingAQuantumSolution.

1 INTRODUCTION

Quantum computing has been a promise for decades. Finally, the first quantum computers are available. They are noisy and do not yet scale, but they can solve problems of reasonable complexity, in particular, optimization problems. Furthermore, there exists now a plethora of programming libraries that facilitate programming these optimization problems and execute them on quantum computers or simulators. Therefore, many different application domains are now exploring whether quantum computing can give them a significant advantage over classical computing. The hope is to obtain better solutions than current heuristic approaches for large-scale problems. The database community has also joined this quest, and first papers have been published in major database conferences and associated quantum workshops. While some efforts explore how database research can support quantum computing, i.e., *DB for Quantum* (see e.g. [8, 12, 16]), or build quantum data structures for DB (e.g. [17, 34]), most efforts so far have analyzed how quantum computing can accelerate traditional database tasks, i.e., *Quantum for DB* such as [2, 3, 6, 7, 9–11, 13–15, 19–21, 24, 25, 29–34]. Compute-intensive tasks that can be expressed as optimization problems are particularly promising for quantum computing [11]. Among them are multi-query optimization [6, 29, 32], join order optimization [27, 30, 35], cardinality estimation [15], schema matching [7, 9], transaction ordering [3, 24], index advisors [10, 13, 33, 34], and data allocation problems [31]. However, the Quantum-for-DB research community is still small. There is some hesitation within the larger database community to explore quantum, probably because of the assumption that it will require significant knowledge about physics and quantum architectures – which are quite different to traditional computer architectures – and that the research itself will require radically new methods and skills. That is, the move towards this new research area appears daunting. In fact, many of the papers published so far spend a significant effort on explaining the high-level concepts of quantum computing to provide an intuition of how the quantum programs look like and execute.

Nevertheless, when screening existing literature on Quantum-for-DB, some common patterns appear. Database tasks are commonly formulated as optimization problems: finding a cost function that needs to be minimized and determining the constraints on its parameters. Then, the optimization problem is reformulated into a quadratic unconstrained binary optimization (QUBO) problem that involves finding the combination of binary variables that minimizes a quadratic polynomial, without any constraints. QUBO is a standard high-level problem formulation for programming quantum annealer hardware, and thus a highly popular starting point. Furthermore, there exist mechanisms to translate QUBO problems into Quantum Approximate Optimization Algorithms (QAOA) used on gate-based quantum computers. Once the QUBO is formulated, experiments are conducted on simulators and/or real hardware. They

Proceedings of the VLDB Endowment. ISSN 2150-8097.

would *ideally* show that the quantum computing approach (i) finds the optimal solutions, (ii) finds them for large problem sizes (many joins, many queries, etc.), and (iii) finds these solutions faster than classical baselines would. With this, the research methodology has in fact similarity with traditional research in database systems: formulate the problem, find appropriate algorithms, implement them, compare their performance with existing solutions, improve over the solutions to achieve scalability, reduce bottlenecks to get better results, and adjust solutions to support adjacent DB problems.

Yet things are not that easy, in particular, because current performance results commonly do not exhibit properties (i)-(iii). First, quantum computers are noisy and probabilistic. Therefore, an analysis must contain an evaluation of the viability of the results. And with current quantum hardware, the results only start to be promising. Second, current hardware cannot yet handle large-scale problems, and simulators are slow. Third, the current interfaces to quantum hardware can lead to excessive execution times due to the current cloud- and batch-processing access. Therefore, current research is still in an explorative phase, providing more qualitative than quantitative evidence that quantum approaches are promising. In particular, the research often lacks proper tuning and scalability analysis w.r.t. larger problem spaces. It is not clear how much knowledge about the underlying hardware is required [22].

In summary, while we database researchers are familiar with the database problems and the overall research methodology, we lack knowledge of the particular quantum techniques, the mechanisms to evaluate results, and how to manipulate the tuning knobs. This raises the question of how difficult it is to acquire the knowledge needed to effectively contribute to Quantum-for-DB.

In this paper, we explore part of this question by analyzing how difficult it is to get familiar with the implementation and evaluation of a quantum solution for a particular database task, namely data allocation. We want to shed light on the options that exist to implement and evaluate a specific quantum solution, and how to evaluate results obtained from quantum computers that will help refine the problem and solution space. We approach the question from the perspective of database researchers that have limited knowledge about quantum, and analyze how far we can go considering the quantum computer as a black or at most gray box. We also ask at what point we need to dig deeper into the physical and computational aspects of the quantum computer or ask quantum experts to provide us with better interfaces or abstractions, so that we can develop better, more scalable Quantum-for-DB solutions.

In the following, Section 2 first introduces the QUBO formulation of the data allocation problem that we use [31], and gives an overview of how to program quantum computers. Section 3 then first discusses the complexities of programming and configuring our solution. We then present performance results for various options in Section 4, highlighting their differences, and discussing possible bottlenecks in the QUBO that limit scalability and the possibilities to avoid them. Section 5 provides discussion and lessons learned.

2 BACKGROUND

2.1 Data Allocation as a QUBO Problem

Many database problems can be expressed as optimization problems with constraints (e.g., minimize execution costs while not exceeding node capacity). The challenge is to express this as a QUBO problem, which has the form $\min_{x\in\{0,1\}^n}\left(\sum_i Q_{ii}x_i + \sum_{i<j} Q_{ij}x_ix_j\right)$ with x_i being binary variables and Q_{ij} biases (real-valued weight coefficients). A QUBO for data allocation is proposed by Trummer in [31] (without an implementation). It assumes a set of nodes N, where $n \in N$ has capacity $n.capacity$, and a set of partitions P where $p \in P$ has partition size $p.size$. For each node n and partition p the request rate r_{pn} reflects how frequently p is accessed by a client of n. Communication cost for a local partition is zero and $p.size$ when accessing a partition p that only resides on a remote node. The goal is to find an allocation of partitions to nodes that (a) minimizes overall communication costs (by assigning partitions to nodes whose clients most access them), (b) meets the safety constraint that each partition is replicated exactly on k nodes, and (c) meets the capacity constraint that the sum of the sizes of all partitions held at node n is smaller than or equal to n's capacity $n.capacity$.

The objective to be minimized is written as $Q = h(Q_R + Q_S) + Q_C$ where Q_R enforces replication, Q_S enforces capacity, Q_C is the communication cost, and h is large enough to ensure that the constraints are fulfilled. The communication cost is

$$Q_C = \sum_{n\in N}\sum_{p\in P} r_{pn}p.size(1 - A_{pn}), \tag{1}$$

where $A_{pn} \in \{0, 1\}$ are the binary variables whose assignment we want to find. $A_{pn} = 1$ indicates that partition p is stored on node n, $A_{pn} = 0$ that it is not. If for each $r_{pn} > 0$: $A_{pn} = 1$, then Q_C is 0.

We ensure that each partition is stored on exactly k nodes as

$$Q_R = \sum_{p\in P}\left(\sum_{n\in N} A_{pn} - k\right)^2. \tag{2}$$

Q_R will be 0 if for each p, $A_{pn} = 1$ for exactly k nodes.

The capacity constraint is more complex because it has an inequality. To transform this into an equality constraint, we can introduce a slack variable s_n so that $\sum_{p\in P} A_{pn}p.size + s_n = n.capacity$. This integer variable must be translated into a set of binary slack variables s_{in}. In [31] the full capacity is split into a set I_n of chunks of sizes 1, 2, 4, 8, .. so that $\sum_{i\in I_n} i.size = n$. With this, the final constraint is expressed as

$$Q_S = \sum_{n\in N}\left(\sum_{p\in P} A_{pn}\, p.size - \sum_{i\in I_n} i.size\, s_{in}\right)^2.$$

reaching its minimal value of zero with the right choices of s_{in} iff the storage consumed does not exceed the nodes' capacities.

2.2 Quantum Architectures and Processing

The two most common quantum architectures are quantum annealing and gate-based quantum computers. Both compute with qubits. A qubit can exist in a state of superposition, but upon measurement, it collapses into one of two definite states, spin-up or spin-down, which correspond to the classical binary values 0 and 1. Qubits are typically matched to the binary variables in the QUBO problem.

Quantum Annealer. D-Wave [5] is currently the only commercially available quantum annealer. Its Ocean SDK allows programmers to specify an optimization problem in various formats. Programs can execute on real quantum hardware or using a simulator.

Using the standard *QPU API* (QPU stands for Quantum Processing Unit), the programmer directly specifies the QUBO as a program which is then automatically translated into a quantum model that assigns the QUBO variables to qubits of the annealer. An annealing execution is then started with a strong magnetic field and qubits in superposition. The field is then slowly adjusted over a given annealing time so that the qubits align into either up or down states, aiming at a global minimum energy state. The final state is measured, representing the binary solution to the QUBO. As the computers are noisy, the execution is repeated many hundreds or thousands of times (called *reads*) so that, hopefully, one of the collected solutions is optimal or close to optimal.

Transforming constrained optimization problems into unconstrained versions can quickly involve many variables, especially when slack variables have to be introduced, ultimately limiting the scalability. Therefore, D-Wave offers *hybrid solvers* where constraints can be programmed explicitly and execution is hybrid, involving classical compute hardware. The idea is that smaller subproblems are submitted to the quantum hardware and classical computations guide the exploration space. For this paper, we explored the *CQM API*, which supports our Q_C objective function and allows to explicitly declare the replication and capacity constraints.

Gate-based Quantum Computing. Using a gate-based quantum computer, the QUBO problem is typically translated into a program that implements the Quantum Approximate Optimization Algorithm (QAOA). This is a hybrid algorithm, where some execution is done on the classical computer, and some on the quantum hardware (not the same as D-Wave's hybrid solvers!). The programmer translates the QUBO problem into a corresponding target cost function, called *Cost Hamiltonian* H_C, Additionally, a Mixer H_M function must be specified. Similar to the annealer, the variables of H_C will be mapped to qubits, and the values of these qubits that lead to the globally lowest-energy state are again the best solution to the problem. H_M allows the quantum state to move through the solution space, preventing it from getting stuck in suboptimal answers. The programmer then defines a circuit (sequence) with p layers. Each layer i is a concatenation of H_C and H_M with layer specific parameters γ_i and β_i (that represent angles). The overall execution then executes the circuit in a nested loop. The inner loop executes the circuit many times (called shots), and the outer loop adjusts the values γ and β using classical optimization until the changes to γ and β no longer improve the results.

SDKs[1] like PennyLane and Qiskit provide ample template code. The programmer's main tasks are to specify H_C, choose an H_M and a classical optimization method (pre-implemented options exist).

3 EXPERIENCE REPORT ON IMPLEMENTATION COMPLEXITY

Given the QUBO formulation, implementing a first version and testing it on the simulator was fast (less than a day for the annealer). Graduate students and advanced undergraduate students will only need a few days to get acquainted with the programming environments. The actual implementation of the quantum problem (excluding manipulation of results) was only a few hundred lines of code. These observations confirm the analysis performed in [28].

[1]Pennylane: https://pennylane.ai/, Qiskit: https://www.ibm.com/quantum/qiskit.

The two main challenges are the many configuration parameters that need careful tuning and the interpretation of the results. Both need to be done well in order to drive further experiments.

Implementation for Quantum Annealer. We used two different APIs of the D-Wave SDK 9.3.0: The QPU API using the full unconstrained QUBO formula, and using the hybrid CQM solver. The QPU API version is also used for simulation[2].

The QPU API offers roughly 20 parameters. Around 5 were easy to understand for a relative quantum "novice", around 10 needed some understanding of how the quantum hardware works, and the remaining 5 required advanced knowledge that was well beyond the learning capacity of our short project. In our experiments, we mainly explored the number of reads and the annealing time, leaving the parameters that require advanced knowledge at their default values. Although number of reads and annealing time are conceptually easy to understand, it is not straightforward to configure them. Obviously, the more reads and the longer the annealing time, the more expensive the execution. Too short or too long times will not provide the best results, as discussed in [30]. D-Wave recommends to start testing with small values and then increase exponentially until results do not improve anymore. Yet we could not find actionable guidelines that would allow one to set default values based on the QUBO problem itself, e.g., based on the number of variables, how they connect, or the biases. Good values might depend not only on the QUBO, but also on the hardware itself [22]. Thus, proper tuning of even these easy-to-understand parameters will require more than a superficial understanding.

The CQM API only allows to specify the overall time for the hybrid computation. It is restricted by several properties that D-Wave sets internally, depending on the complexity of the problem.

Implementation for Gate-Based Quantum Computing. We used the Qiskit Aer 0.17.2 simulator, as we did not have access to gate-based quantum hardware. The simulator provides standard implementations for the mixer and classical optimization routines. We implemented the QUBO-specific code in the cost Hamiltonian and used the standard transverse-field mixer provided by the SDK. Problem-specific mixers can improve performance through effective search space exploration while preserving constraints. However, significant background knowledge is needed to choose the right one. For the classical optimization procedure used to determine the γ and β values of the QAOA circuit, we tested the COBYLA and SPSA optimizers, the latter being specifically designed to perform well in the presence of noisy objective function evaluations. Additional tunable parameters include the circuit depth p and the number of measurement shots. We fixed circuit depth at $p = 2$, as increasing the depth did not significantly improve performance.

There is a wide range of parameters, and choices for Mixer and optimizer. Default options are provided, but a deeper understanding of how the problem maps to the quantum architecture would allow for more targeted circuit creation.

4 EXPERIMENTAL RESULTS

We present more comprehensive results for the annealer, but we also provide short insights into our gate-based evaluations. In our

[2]D-Wave SDK 9.3.0 offers two simulators, Simulated Annealing and Simulated Quantum Annealing (SQA). We are using SQA, as recommended by our D-Wave contacts.

experiments, we tested configurations with increasing number of nodes and partitions. For each configuration, we generated multiple random problem instances with varied access frequencies , node capacities and partition sizes (all given as integers), and results presented are on average over these instances. We ensured that for each configuration, all partitions with correct replication factor fit into the nodes with some extra capacity[3,4].

Recall that the optimization problem aims to allocate partitions across nodes so that communication cost to remote nodes is minimized. For our quantum executions, we check all returned solutions and take from the valid solutions (that do not violate the constraints), the one with the lowest communication cost. We compare this solution with a solution provided by a baseline classical solver. For (nearly) all our experiments, this classical solver provided better solutions in terms of communication costs than our best quantum solution[5]. Therefore, our figures indicate the percentage gap by which the quantum solution is worse than the baseline.

4.1 Results from Annealer

QPU Results on the Quantum Annealer Simulator. We first ran the implementation and problem specification using the QPU API on the D-Wave quantum simulator. For all experiments, we used 50 μs annealing time, 500 reads by default, and 1000 reads for node count ≥ 5 or partition count ≥ 25. Figure 1a shows the performance gap for up to 9 nodes and 50 partitions. For very small problem sizes, the quantum simulator finds solutions close to optimal, but with increasing number of nodes and partitions, the performance quickly becomes significantly worse than the baseline.

We explored different solution spaces. First, we replaced the slack variables for the capacity constraint with an unbalanced penalization function *SF* that has higher values when the constraint is violated than when it is not [23]. Yet Figure 1b shows that the performance gap is even larger. The valid solutions it returns are not rejected, but are of lower quality: the unbalanced penalty spreads replicas evenly across nodes rather than concentrating them on the nodes that request them most, which raises communication cost.

We then simplified the capacity constraint by assuming all partitions have the same size. With this, we can remove $p.size$ while $n.capacity$ expresses how many partitions the node can hold. This makes sense, for example, when files are split into equal-sized blocks and distributed across nodes. However, note that this modification does not change the number of qubits needed. Figures 1c and 1d show the results for QUBO with slack variables and *SF*, respectively. Indeed, with the simplified QUBO formula, the simulator could find better solutions and the performance gap is lower.

Note that in nearly all configurations most of the tests returned a valid result. The only exception is *SF* with arbitrary partitions and 9 nodes, where success rates are as low as 60%.

[3]As the programs were created by different students, the generation of problem instances and the choice of baseline comparison differ slightly between annealer and gate-based approaches. The differences have little impact at the level of the results presented in this section. The detailed implementations can be found in the repository.
[4]Annealer experiments run from a MacBook Pro with Apple M1 Max chip and 64 GB of RAM, gate-based simulations with an 8-core MacBook Air and 8 GB of RAM.
[5]For the annealing experiments, this is always true, because the baseline is an ILP that always returns the minimum cost. The gate-based experiments use a greedy algorithm as baseline which does not always provide the lowest-cost solution.

QPU Results on D-Wave Quantum Hardware. The programs using the QPU API were further sent to D-Wave's cloud hardware and executed on the Advantage System, which provides over 5000 qubits in a Pegasus topology with 15-way qubit connectivity. Because our QUBO variables are far more densely connected than this topology, the problem must be minor-embedded, representing each logical variable by a chain of physical qubits. For this we used the SDK's automatic embedding (EmbeddingComposite)We ran the exact same experiments as on the simulator. The results are shown in Figure 2. Using the original QUBO problem with slack variables, we do not get any valid results beyond 12 partitions, and that for arbitrary and fixed-sized partitions (Figures 2a and 2c). For the smaller configurations, the valid results are worse than on the simulator, in particular for arbitrary partition sizes. This means that the simulator might not truly reflect the performance of the actual, more noisy hardware. Furthermore, scalability could possibly be limited by the number of qubits of the current D-Wave hardware.

Replacing slack variables with *SF* (Fig. 2b and 2d) allows to scale up to 50 partitions, but only for small node sizes. The communication costs of the best solutions are much better than with slack variables and more in line with the simulator results.

Overall, the experiments show that we can indeed run our DB optimization problem on quantum hardware, but only for a limited problem space. The results returned are far from optimal, and get worse as the problem size increases.

D-Wave CQM Hybrid Solver Results. Figure 3 shows the results for the hybrid solver. We kept the default run time. The results are much better than when using the QPU API, and relatively close to the optimal even for some of the larger problem sizes. The reason is that the QUBO problem submitted to the hardware is likely to be much easier because D-Wave will perform simple validations and parameter space exploration on classical hardware.

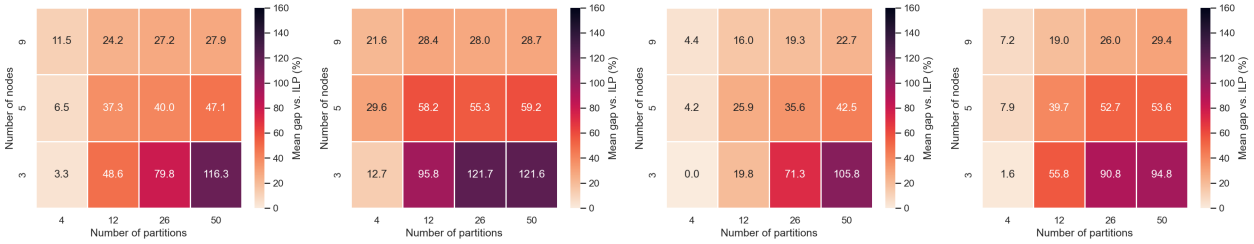
Run time. The classical baseline (ILP) for one optimization takes between 30-700 ms, but remains mostly below 200 ms. Even for the larger problem sizes, some instances were solved very fast.

For the actual quantum hardware, run times for the QPU API implementations vary widely and could be tens of seconds. One cause is the communication to the cloud hardware as well as queuing within D-Wave. For the actual access time on the quantum hardware, a first intuition would expect 25 ms with 500 reads and 50 ms with 1000 reads as annealing for one read is set to 50 μs. However, it is much higher because of the one-time initialization cost, a readout time (around 150-200 μs per read), and a delay time (20 μs per read). As such, 1000 reads take around 300 ms, which is actually very comparable with the baseline. While we can expect this time to increase with the problem size (as readout times will increase somewhat), we can expect the run times for the classical baseline to increase faster, making the quantum solution more interesting.

For the CQM hybrid solver, we did not specify a total execution time but used the default (although an exploration might be very useful). In our experiments, this led to around 3 seconds total, but only around 100 ms on quantum hardware.

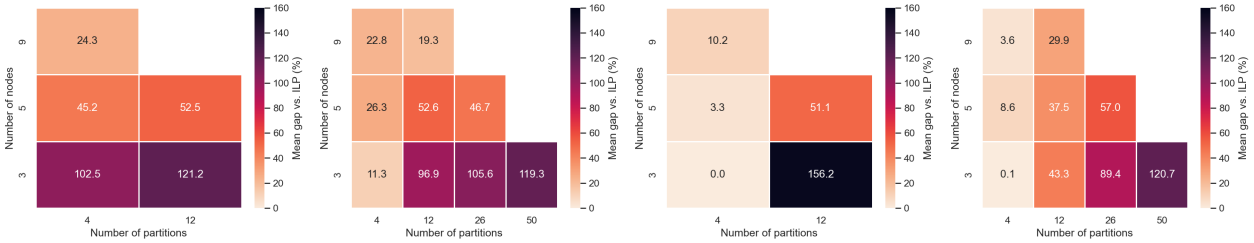
4.2 Results from Gate-based Simulations

The initial results for our gate-based implementation showed severe limitations due to the slack variables. Even small simulations took



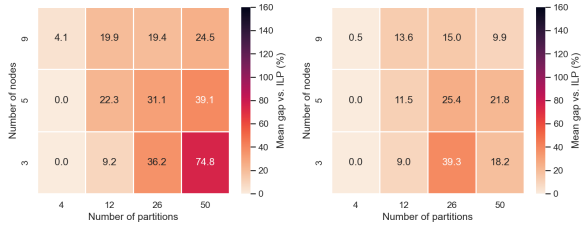
(a) Arbitrary Partitions (b) Arbitrary Partitions, SF (c) Fixed-Sized Partitions (d) Fixed-Sized Partitions, SF

Figure 1: Simulated Quantum Annealer, average mean gap vs. baseline



(a) Arbitrary Partitions (b) Arbitrary Partitions, SF (c) Fixed-Sized Partitions (d) Fixed-Sized Partitions, SF

Figure 2: D-Wave Annealer, average mean gap vs. baseline



(a) Arbitrary Partitions (b) Fixed-Sized Partitions

Figure 3: CQM Hybrid Solver, average mean gap vs. baseline

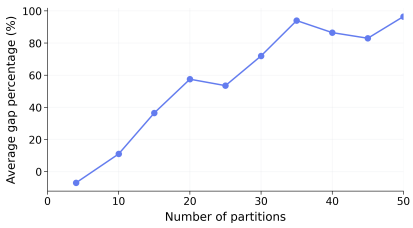


Figure 4: Simulated Gate-based Quantum Computer, average mean gap vs. baseline

hours and still failed to return a valid solution. To mitigate this issue, we simplified the QAOA formulation by retaining only the cost and redundancy terms in the QUBO and checking capacity violations classically after sampling.

Scaling the Number of Partitions/Nodes. Figure 4 indicates the average performance gap across all tested configuration instances vs. baseline for a fixed set of 3 nodes and increasing number of partitions. Similarly to the annealing results, the performance gap becomes worse. Note that in these experiments, we used a greedy baseline. As such, it is worth noting that the quantum optimizer occasionally outperforms the greedy heuristic on specific instances, mainly with a low number of partitions, but sometimes even when there are 15 or 20 partitions. Given that our QUBO problem did not encode the capacity constraint, an analysis of how many valid results the simulator returns provides little insight. In a second experiment, we increased the number of nodes (and accordingly the number of partitions to avoid trivial solutions). In this regime, QAOA deteriorates rapidly as the lack of expressing capacity constraints resulted in many non-valid solutions. We could only get reasonable results up to 7 nodes. The gap in solution quality again increased with problem complexity.

Given that we only performed experiments on the simulator, a realistic runtime comparison is not possible. As expected, the simulator runs orders of magnitude slower than the greedy baseline.

5 DISCUSSION

Implementation and Tuning Complexity. First implementations are easy, but the choice of configuration parameters can seem overwhelming. Defaults are given, but appropriate tuning of many of them requires advanced knowledge of how the QUBO/QAOA program matches the quantum architecture and what exactly happens during the quantum process. Analysis tools have been proposed [22], but they are not easy to understand themselves.

A further challenge was the terminology used in the Quantum SDK documentation, handbooks and tutorials, containing many notations and terms from physics/maths that are likely unfamiliar

for DB researchers. We believe that in some cases it would be possible to find simple translations to more familiar terminology or to provide descriptions at a different level of abstraction. This would make it easier to get started. But in order to get truly better solutions, one will need to dig deeper into the actual functioning of the quantum computer, and thus, will need to understand the common terminology and language used in this domain. This will also be necessary so that we, who use Quantum-for-DB, can explain to the quantum experts what we need from the quantum hardware and lower-level libraries to make our solutions work better.

Current run times. Simulators are useful for development and test, yet when simulating quantum hardware, run times can be long, even for relatively small inputs. In our experiments, the simulators were executed on commodity laptops. A preferable setup would be a multi-core workstation or server with ample main memory and, for GPU-accelerated simulators, one or more GPUs. Also, quantum hardware vendors provide cloud-hosted simulators that run on high-performance hardware and are available as paid services.

Using the actual D-Wave computer, requests have to be sent to the cloud and are queued in batch mode before execution. Thus, while the actual quantum access time appears to be comparable with baselines for the small problems we analyzed, the overall run time is large. This makes it difficult to integrate quantum solutions dynamically into a data management pipeline. However, one can expect the interfaces to quantum computers to become more versatile and powerful, leading to less "wasted" time and allowing quantum computers to become competitive with classical baselines.

Solution Quality. Already for our relatively small problem sizes, the solutions returned deviated significantly from the optimal solutions. This is in line with some but not all of the results from the research literature. For instance, [18, 19, 21] integrate join order optimization using D-Wave's CQM hybrid quantum annealing into the PostgreSQL query optimizer pipeline. Here, the join orders found by quantum annealing on D-Wave are better than those produced by PostgreSQL's query optimizer. In fact, as much so, that for some queries, although the quantum optimization takes a long time, the total execution (optimization + query execution) is *faster* with quantum annealing than PostgreSQL's baseline.

Configuration Parameter Tuning. The limited availability of quantum hardware and the long run times when simulating this hardware make it essential to configure experiments carefully. Configuration management is a challenge in many domains. Machine learning requires hyper-parameter tuning; traditional DBMS, key-value stores and distributed computing platforms also offer a plethora of configuration parameters. Moreover, there exists significant research in auto-tuning these systems. There will be no doubt research needed in tuning DB quantum computations. The question is how much knowledge about the underlying quantum architecture and physics is ultimately needed.

To some extent, we have to rethink the evaluation process. A typical evaluation of a DBMS uses system-level performance metrics like response time and throughput, and performs a bottleneck analysis considering the classical resources (e.g., CPU, memory, network). This motivates the development of new software, new algorithms, and new data structures that reduce resource consumption and improve performance. In principle, we can use the same methodologies, yet we are confronted with very different resources.

Scalability. In many cases, optimal solutions for small problems can be solved efficiently with classical solvers. The hope is that quantum computing might eventually outperform classical solutions, namely, once the quantum hardware can handle larger problem spaces where classical computing has to fall back to heuristics.

For the data allocation problem, [31] provides a lower bound of the number of qubits where the dominating term contains the product $|N| * |P|$. Just taking 10 nodes and 500 partitions, this amounts to 5000 qubits that need to be *fully* connected, which is not the case for current hardware. The Advantage System we used has just over 5000 qubits with 15-way connectivity, and even the newer Advantage2 generation (about 4400 qubits with 20-way connectivity) does not change this fundamental limitation.

Considering real-world use cases such as Google's file system, with thousands of nodes and millions of data chunks, the QUBO formulation used here might not be realistic to start with, even for next-generation quantum computers.

Problem Formulation. Given the scalability constraints of current and probably also near-future quantum computers, it will be necessary to tackle the scalability problem at the time the optimization problem is formulated. There have already been approaches in database research to dissect QUBO problems into smaller subcomponents or into formats that can map to fewer qubits [26, 29].

For our evaluation, the slack variables were very expensive, and simply allowing only for a constant partition size ($p.size = 1$) considerably improved performance. Yet there may exist a completely different optimization formulation that has lower bounds on the minimum number of qubits, making the solution more scalable.

Even more challenging is that current work is often based on simplifying assumptions to keep the complexity at bay. For the data allocation problem at hand and looking at the existing literature on partitioning and replication [1, 4], the replication factor might not be fixed, costs might be more complicated than simply communication costs, and workloads might shift, requiring incremental data allocation and dynamic migration.

6 CONCLUSIONS

In this paper, we present our experience report of implementing and evaluating a known QUBO-based quantum solution for the data allocation problem. Implementation and initial setup are relatively straightforward, even with limited quantum background, allowing for a quick start and useful insights into the possibilities and limitations of current quantum solutions and quantum architectures. Although we believe that interesting open problems can be tackled after a relatively short exposure to the field, sufficient knowledge of the underlying quantum mechanisms is important to exploit the many fine-tuning options and will be crucial to find truly ground-breaking quantum solutions for database problems.

7 ACKNOWLEDGEMENTS

D-Wave provided us access to the Leap™ Quantum Cloud Service through the Leap Quantum LaunchPad™ Program. The Passau International Centre for Advanced Interdisciplinary Studies (PICAIS) of the University of Passau, Germany, provided partial funding.

REFERENCES

[1] Omar Asad and Bettina Kemme. 2016. Adaptcache: Adaptive data partitioning and migration for distributed object caches. In *Proceedings of the 17th International Middleware Conference*. 1–13. https://doi.org/10.1145/2988336.2988343

[2] Tim Bittner and Sven Groppe. 2020. Hardware Accelerating the Optimization of Transaction Schedules via Quantum Annealing by Avoiding Blocking. *Open Journal of Cloud Computing* 7, 1 (2020), 1–21.

[3] Umut Çalikyilmaz, Sven Groppe, Jinghua Groppe, Tobias Winker, Stefan Prestel, Farida Shagieva, Daanish Arya, Florian Preis, and Le Gruenwald. 2023. Opportunities for quantum acceleration of databases: Optimization of queries and transaction schedules. *Proceedings of the VLDB Endowment* 16, 9 (2023), 2344–2353.

[4] Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. 2010. Schism: a workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 48–57. https://doi.org/10.14778/1920841.1920853

[5] D-Wave Quantum Inc. 2026. D-Wave. https://www.dwavequantum.com/ Accessed: 2026-July-10.

[6] Tobias Fankhauser, Marc E. Solèr, Rudolf Marcel Füchslin, and Kurt Stockinger. 2023. Multiple Query Optimization Using a Gate-Based Quantum Computer. *IEEE Access* 11 (2023), 114031–114043. https://doi.org/10.1109/ACCESS.2023.3324253

[7] Kristin Fritsch and Stefanie Scherzinger. 2023. Solving Hard Variants of Database Schema Matching on Quantum Computers. In *PVLDB*, Vol. 16. 3990–3993. https://doi.org/10.14778/3611540.3611603

[8] Floris Geerts and Rihan Hai. 2025. QC meet CQ: Quantum Conjunctive Queries. In *Proceedings of the 2nd Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications*. 25–25. https://doi.org/10.1145/3736393.3736696

[9] Luisa Gerlach, Tobias Köppl, Stefanie Scherzinger, Nicole Schweikardt, and René Zander. 2025. A Quantum-Leap into Schema Matching: Beyond 1-to-1 Matchings. *Proc. ACM Manag. Data* 3, 2 (2025), 89:1–89:24. https://doi.org/10.1145/3725226

[10] Le Gruenwald, Tobias Winker, Umut Çalıkyılmaz, Jinghua Groppe, and Sven Groppe. 2023. Index Tuning with Machine Learning on Quantum Computers for Large-Scale Database Applications. In *CEUR Workshop Proceedings*.

[11] Rihan Hai, Shih-Han Hung, and Sebastian Feld. 2024. Quantum Data Management: From Theory to Opportunities. In *40th IEEE International Conference on Data Engineering, ICDE 2024*. IEEE, 5376–5381. https://doi.org/10.1109/ICDE60146.2024.00410

[12] Rihan Hai, Shih-Han Hung, Tim Coopmans, Tim Littau, and Floris Geerts. 2025. Quantum data management in the NISQ era. *Proceedings of the VLDB Endowment* 18, 6 (2025), 1720–1729. https://doi.org/10.14778/3725688.3725701

[13] Manish Kesarwani and Jayant R. Haritsa. 2024. Index Advisors on Quantum Platforms. *PVLDB* 17, 11 (2024), 3615–3628. https://doi.org/10.14778/3681954.3682025

[14] Manish Kesarwani and Jayant R. Haritsa. 2024. Is Quantum-Based SQL Query Execution Viable?. In *International Workshop on Quantum Data Science and Management (QDSM)*.

[15] Florian Kittelmann, Pavel Sulimov, and Kurt Stockinger. 2024. QardEst: Using Quantum Machine Learning for Cardinality Estimation of Join Queries. In *Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications, Q-Data 2024*. https://doi.org/10.1145/3665225.3665444

[16] Tim Littau and Rihan Hai. 2025. Qymera: Simulating quantum circuits using RDBMS. In *Companion of the 2025 International Conference on Management of Data*. 179–182. https://doi.org/10.1145/3722212.3725126

[17] Tim Littau, Ziyu Li, and Rihan Hai. 2024. Towards Quantum Data Structures for Enhanced Database Performance. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024*.

[18] Hanwen Liu, Abhishek Kumar, Federico M. Spedalieri, and Ibrahim Sabek. 2025. Hybrid Quantum-Classical Optimization for Bushy Join Trees. In *Proceedings of the 2nd Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications, Q-Data 2025*. 20–24. https://doi.org/10.1145/3736393.3736695

[19] Hanwen Liu and Ibrahim Sabek. 2026. Is Quantum Computing Ready for Real-Time Database Optimization? *arXiv preprint arXiv:2601.12123* (2026).

[20] Hanwen Liu and Ibrahim Sabek. 2026. Towards a Hybrid Quantum-Classical Computing Framework for Database Optimization Problems in Real Time Setup. *arXiv preprint arXiv:2602.14263* (2026).

[21] Hanwen Liu, Federico M. Spedalieri, and Ibrahim Sabek. 2025. A Demonstration of Q2O: Quantum-augmented Query Optimizer. *Proc. VLDB Endow.* 18, 12 (2025), 5439–5443. https://doi.org/10.14778/3750601.3750691

[22] Wolfgang Mauerer and Manuel Schönberger. 2026. A Toolbox to Understand the Physics of Quantum Data Management. In *Proceedings of the Third Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications (Q-Data '26)*. arXiv:2605.14719 https://arxiv.org/abs/2605.14719

[23] J A Montañez-Barrera, Dennis Willsch, A Maldonado-Romo, and Kristel Michielsen. 2024. Unbalanced penalization: a new approach to encode inequality constraints of combinatorial problems for quantum optimization algorithms. *Quantum Science and Technology* 9, 2 (April 2024), 025022. https://doi.org/10.1088/2058-9565/ad35e4

[24] Nitin Nayak, Alexandru Prisacaru, Umut Çalıkyılmaz, Jinghua Groppe, and Sven Groppe. 2025. Quantum-enhanced transaction scheduling with reduced complexity via solving qubo iteratively using a locking mechanism. In *Proceedings of the 2nd Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications*. 26–35.

[25] Nitin Nayak, Jan Rehfeld, Tobias Winker, et al. 2023. Constructing Optimal Bushy Join Trees Solving QUBO Problems on Quantum Hardware and Simulators.. In *Big Data in Emergent Distributed Environments*. https://doi.org/10.1145/3579142.3594298

[26] Nitin Nayak, Tobias Winker, Umut Çalikyilmaz, Sven Groppe, and Jinghua Groppe. 2024. Quantum Join Ordering by Splitting the Search Space of QUBO Problems. *Datenbank-Spektrum* 24, 1 (2024), 21–32. https://doi.org/10.1007/S13222-024-00468-3

[27] Pranshi Saxena, Ibrahim Sabek, and Federico Spedalieri. 2024. Constrained Quadratic Model for Join Orders. In *Q-Data Workshop*. 38–44. https://doi.org/10.1145/3665225.3665447

[28] Manuel Schönberger, Maja Franz, Stefanie Scherzinger, and Wolfgang Mauerer. 2022. Peel | Pile? Cross-Framework Portability of Quantum Software. In *IEEE 19th International Conference on Software Architecture Companion, ICSA Companion 2022*. 164–169. https://doi.org/10.1109/ICSA-C54293.2022.00039

[29] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2025. Large-Scale Multiple Query Optimisation with Incremental Quantum(-Inspired) Annealing. *Proc. ACM Manag. Data* 3, 4 (2025), 253:1–253:25. https://doi.org/10.1145/3749171

[30] Manuel Schönberger, Stefanie Scherzinger, and Wolfgang Mauerer. 2023. Ready to Leap (by Co-Design)? Join Order Optimisation on Quantum Hardware. *ACM on Management of Data* 1, 1 (2023), 1–27. https://doi.org/10.1145/3588946

[31] Immanuel Trummer. 2025. Leveraging Quantum Computing for Optimal Data Allocation in Distributed Systems. In *Proceedings of the 2nd Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications, Q-Data 2025*. ACM, 10–17. https://doi.org/10.1145/3736393.3736692

[32] Immanuel Trummer and Christoph Koch. 2016. Multiple query optimization on the D-Wave 2X adiabatic quantum computer. *PVLDB* 9, 9 (2016), 648–659. https://doi.org/10.14778/2947618.2947621

[33] Immanuel Trummer and Davide Venturelli. 2024. Leveraging Quantum Computing for Index Selection. In *Q-Data Workshop*. 14–26. https://doi.org/10.1145/3665225.3665445

[34] Yannis Tzitzikas and Haridimos Kondylakis. 2025. Knowledge graphs and quantum computing: first blood. In *Knowledge Graphs: 14th International Joint Conference, IJCKG 2025*. Springer, 3–18. https://doi.org/10.1007/978-981-95-5009-8_1

[35] Tobias Winker, Umut Çalıkyılmaz, Le Gruenwald, and Sven Groppe. 2023. Quantum Machine Learning for Join Order Optimization using Variational Quantum Circuits. In *BigData Workshop*. 1–7. https://doi.org/10.1145/3579142.3594299