Leveraging Hybrid Classical-Quantum Methods for Efficient Load Rebalancing in HPC

Justyna Zawalska[1,2§], Minh Chung[3,5§], Katarzyna Rycerz[1,2], Laura Schulz[3], Martin Schulz[3,4], Dieter Kranzlmüller[3,5]
[1]AGH University of Krakow, Institute of Computer Science, Krakow, Poland
[2]Academic Computer Center CYFRONET AGH, Krakow, Poland
[3]Leibniz Supercomputing Centre (LRZ), Garching bei München, Germany
[4]Chair for Computer Architecture and Parallel Systems, Technical University of Munich (TUM), Germany
[5]MNM-Team, Ludwig-Maximilians-Universität München (LMU), Germany
{jzawalska, kzajac}@agh.edu.pl, {minh.chung, laura.schulz, martin.schulz, dieter.kranzlmueller}@lrz.de

***Abstract*—Load imbalance is a challenge for parallel applications in High Performance Computing (HPC). It is caused by processes having different execution times or load values, leading to idle or wait times at synchronization points, where faster processes must wait for the slowest process to catch up. To mitigate this issue, applications can employ load balancing (LB) strategies, which migrate load between processes to even out load. This is often referred to as the Load Rebalancing Problem (LRP). While many approaches solving the LRP exist, they can only be heuristics and hence further optimization potential exists. In our work, we turn to a novel approach by using hybrid classical-quantum approaches and present two versions of the constrained quadratic model for solving the LRP; the two differ in how they balance the number of qubits required with the types of applied constraints. We compare the quantum-based methods with classical methods using heuristic algorithms Greedy, Karmarkar–Karp, and ProactLB. We evaluate our approaches using imbalance ratio and speedup as metrics, as well as the number of migrated tasks to indicate overhead caused by migrations. Our results show that the quantum-based methods outperform the classic methods. For example, we need only $1/4$ of the number of migrated tasks in a realistic use case compared with classical methods, particularly Greedy and KK, to balance the load.**

***Index Terms*—HPC, Quantum Computing, HPCQC integration, Load Rebalancing, CQM, task migration**

I. Introduction

With the ever-increasing demand for computational power in scientific research, High Performance Computing (HPC) plays a vital role in the field of computational science. Nearly all applications in HPC employ some form of task parallelism, where many tasks are executed concurrently on shared and/or distributed memory machines [1]. In such scenarios, load imbalance poses a critical challenge, especially in task-based parallel applications [2], leading to unwanted idle or wait times. These issues can be mitigated with load balancing (LB) approaches, which redistribute tasks to achieve a more balanced execution thereby reducing idle times. LB methods are categorized as static or dynamic, meaning they can be applied before or during task execution [3]. We focus on the Load Rebalancing Problem (LRP) [4], which assumes that tasks are already distributed on different machines using pre-partitioning algorithms before execution. If the model used to predict task execution times is not sufficiently accurate or task lengths are unpredictable, imbalances can occur in the form of load differences across machines. In such scenarios, migrating tasks at runtime can help rebalance the load.

Aggarwal et al. [4] define the LRP as a problem where N jobs (referred to as tasks) of different workloads are assigned to M processors. Suppose the workloads per task are $w_1, w_2, ..., w_N$. The load on a processor is the sum of all w values of its assigned tasks. Differences in these sums between processes indicate load imbalance. Load rebalancing aims to optimize overall performance by relocating tasks among processors. An appropriate method guiding task migration is the key to solving the LRP. With the constraint of migration costs, LRP is formulated as an NP-hard problem that minimizes imbalance with an additional limit on the number of migrated tasks of $\leq k$. In terms of optimization problems, LRP can be turned to multi-way number partitioning [5]; N tasks with the corresponding workloads are interpreted as numbers, M processes as partitions. Traditionally, many classical algorithms/methods[1] are proposed for number partitioning problems. The most common algorithm is Greedy, a variant of Graham's algorithm [6].

In this work, we approach the Load Rebalancing Problem Problem (LRP) with a novel method using quantum computing, which is known to be suitable for optimization challenges. First, we formulate the LRP as a constrained quadratic model (CQM)[2]. This CQM then serves as the input for the D-Wave Leap hybrid solver, which is based on quantum annealing. The solver's output is a solution that guides task migration, meaning it specifies how many tasks should be migrated and how they are distributed among the processes involved.

Specifically, our paper offers the following contributions:

- We transform the LRP into a classical-quantum problem that can be solved with quantum annealing.

[§]These authors contributed equally.

[1]Algorithms and methods might be used interchangeably, while approaches imply a scheme or a broader strategy in this paper.

[2]https://docs.ocean.dwavesys.com/en/stable/concepts/cqm.html#cqm-sdk

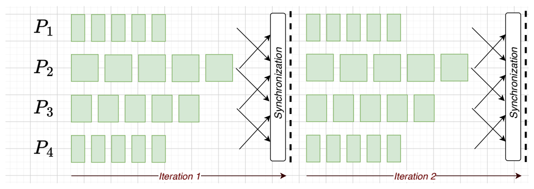

Fig. 1. An illustration of the BSP model as iterative applications in HPC.

- We implement and execute our approach on both a synthetic benchmark and a real-world application, the Sam(oa)^2 Tsunami simulation.
- We assess the influence of different LRP formulations on the results achieved using a quantum device.
- We compare the proposed methods with classical counterparts, i.e., Greedy, Karmarkar–Karp [7], ProactLB [8].

Our results show that, for the real-world benchmark, we are able to achieve load balance by using a quarter of the needed task migrations, substantially reducing overhead.

The remainder of the paper is organized as follows: Section II introduces the problem statement and motivation. Section III reviews related work. Section IV explains the quantum transformation of LRP. In Section V, we discuss the experiments and results of the proposed methods. Section VI provides our analysis of the limits and challenges we face with hybrid classical-quantum schemes, and suggests future work. Lastly, Section VII concludes the paper.

II. Problem Statement and Motivation

We formulate the Load Rebalancing Problem (LRP) similar to Aggarwal et al, [4], but with adjustments tailored for task-based parallel applications. Specifically, we use:

- N tasks in a task-based parallel application are assigned to M processes, assuming one process per node (i.e., M indicates the number of compute nodes).
- Processes are indexed with the following notation: $\{P_1, P_2, ..., P_M\}$.
- The execution time of a task t indicates its length or so-called load value, denoted by w_t.
- The sum of all load values of all tasks assigned to one process P_i indicates its total load value, L_i.
- The maximum and average load values across all processes are denoted by L_{max} and L_{avg}.
- The imbalance level is calculated by a ratio, $R_{imb} = \frac{L_{max} - L_{avg}}{L_{avg}}$ [9].

Using these definitions, the LPR is an optimization problem aimed at reducing L_{max} and R_{imb}, which can be achieved by migrating tasks between processes. However, the migration induces communication and with that overhead, in turn leading to longer process load values. We, therefore, must consider between an optimal R_{imb} and task migration overhead when using LPR in HPC systems to achieve efficient execution.

One of the most commonly seen use cases for LRP can be found in applications relying on the bulk synchronous parallel (BSP) model [10]. BSP is a common parallel execution model used in many scientific applications. In this model, the execution behavior is iterative (typically in the form of an outer time step loop) and each iteration consists of two phases: computation followed by synchronization/communication. The computation phase consists of task execution, followed by the result synchronization before a new iteration starts. Figure 1 shows an example of BSP execution: 4 processes (P_1 to P_4) are each assigned 5 tasks ($N = 20$ tasks in total). Each green box indicates a task, and its length indicates the task's load value. All processes are synchronized in the communication phase, leading to idle/wait times in which processes with smaller load values have to wait for the process with L_{max} to catch up. The goal of LRP is to identify a solution for how tasks should be migrated, i.e., which task should be moved from one process to another.

In this paper, we study this problem using two applications. One is a synthetic test case implementing a matrix multiplication (MxM); the second is sam(oa)^2 [11] [12], an Adaptive Mesh Refinement (AMR)-based code for Tsunami simulation. From both, we gather information on the number of assigned tasks in each process and their lengths in the imbalanced case and apply this as input for both the classical baselines and our novel quantum-hybrid solutions. The implementation of MxM and sam(oa)^2 is ported to task-based parallel applications by using Chameleon [13], a library for task-parallel MPI+X applications (more details in Section V).

For MxM, we decompose the problem into individual MxM kernels and use them as tasks. The load of each task then depends on the number of kernels used and the size of the original matrices. As Figure 1 shows, we can generate different test cases demonstrating load imbalance. The green boxes can represent MxM tasks with uniform load per process, but different processes might have different task lengths.

For sam(oa)^2, the framework supports the implementation of numerical schemes, like partial differential equation (PDE) systems, on dynamically adaptive, tree-structured triangular meshes. Running on shared and distributed-memory systems, sam(oa)^2 partitions the mesh into sections, which define computation tasks. The cells in a section are contiguous along a Sierpinski space-filling-curve (so-called "Sierpinski" sections). We use sam(oa)^2 with the model for solving 2D shallow water equations applied for oscillating lake and tsunami simulations. Sam(oa)^2 provides a finite volume solver as well as a high-order discontinuous Galerkin scheme with a-posteriori limiting (ADER-DG [11]). In ADER-DG, variations in computational cost per element can be caused by the limiting procedure that can lead to load imbalance. For this reason, sam(oa)^2 has its own load balancer based on an application specific cost predictor; however, we assume an incorrect cost model to generate imbalance use cases. We use an extension of sam(oa)^2, which supports distributed tasking with the Chameleon library [13]. Each compute node launches an MPI process with multiple OpenMP threads to execute tasks, where we spawn a section traversal in the time-stepping phase (ADER-DG + Finite Volumes) as a task in Chameleon.

The LRP solution is important not only regarding where tasks are moved, but also the total number of migrated tasks. Due to communication overhead, migrating too many tasks can negatively impact performance. When applying classical algorithms like Greedy or KK, task migration is purely considered a partitioning solution focused on achieving the correct number of migrated tasks without considering the overheads involved. This paper introduced and motivated a new solution based on a hybrid classical-quantum approach, which aims to reduce the number of task migrations and with that overhead.

III. Related Work

Load balancing (LB) aims to assign each process (node) an equal share of the total load [14]. LB can be solved by static or dynamic approaches. A static approach usually assumes prior knowledge, such as the number of tasks, load values, and involved processes [15], while a dynamic approach (DLB) mainly works for on-the-fly load balancing decisions [16] in which prior information is limited, such as only current queue status. A common method for DLB is work stealing [17], where idle processes steal tasks from busier ones at runtime. In HPC systems, work stealing might be delayed due to communication overhead [18]. To improve this, Samfass et al. [19] propose a reactive task offloading method. The reactive method is based on the most current execution status among processes to make speculative and reactive decisions on task migration. A further improvement is from [8], introducing proactive load balancing (ProactLB), which integrates a load predictor at runtime and guides task migration with an appropriate number of tasks as well as appropriate target processes.

In principle, static load balancing makes decisions once before execution, while dynamic load balancing does so periodically during execution. However, similar questions remain, such as which process should migrate tasks, to where, and with how many tasks. When this problem is approached as a classical multi-way number partitioning problem [5], we consider the task load values and their current assignment to processes as the inputs. Further, we constrain the cost of migrating tasks to a constraint, where the number of migrated tasks should be $\leq$ a threshold (called the relocating cost [4]). The applied classical algorithms can be divided into approximation and optimal algorithms [20]. Approximation algorithms, like Greed and KK, gain benefits in complexity and provide an upper bound for optimal algorithms to help prune the search space. Greedy first sorts the input (numbers) and then considers placing them into the subset with the smallest cumulative sum [21]. KK also first sorts the input, but then iteratively replaces the largest two numbers of the input with their difference [7]. In contrast, the algorithm in ProactLB, as mentioned earlier, takes a more distributed view by sorting the total load values of processes and then relying on the difference in total load to search for enough tasks to migrate. This approach aims at limiting migration cost.

LRP, being a combinatorial optimization problem, is well-suited for exploration with quantum computing methods. Rathore et al. [22] investigate quantum annealing (QA) for HPC workload allocation, focusing on adaptive mesh refinement and smoothed particle hydrodynamics. The study utilizes a recursive number partitioning to distribute tasks across a power-of-2 number of processors and employs multi-objective optimization to address the associated complexities. In contrast, our study focuses on rebalancing tasks already assigned with a different problem formulation tailored for quantum devices. To our knowledge, this is the first application of hybrid classical-quantum computing methods for the LRP.

IV. Transformation of LRP for Hybrid Classical-Quantum Solvers

From the problem statement in Section II, we consider the given input: N tasks distributed on M processes. We assume each process is assigned an equal amount of n tasks, $n \times M = N$. Also, in our scope, all n tasks of a process have (initially) uniform execution times. Let w_i be the execution time of the task originally assigned to process P_i, $i \in \{1, \ldots, M\}$. The sum of all task execution time values assigned to a process indicates its total load value, L_i. The maximum load is $L_{max} = max(L_i)$, which is the greatest load among all involved processes. The average load is calculated by $L_{avg} = \frac{1}{M}\sum_i L_i$.

Before rebalancing, the load on process P_i is $L_i = w_i \cdot n$. After rebalancing, the new load is given by $L'_i = \sum_{j=1}^{M} w_j \cdot x_{i,j}$, where $x_{i,j} \in \mathbb{N}_0$ represents the number of tasks moved to process P_i from process P_j (for $i = j$ it denotes the number of tasks that are not relocated). To express this as a binary quadratic model, which can be translated into CQM for the hybrid classical-quantum solver, we need to convert the values of x from non-negative integers to binary variables. For this purpose, we utilize a non-standard binary representation:
$C \in \left\{2^{l-1} \mid l = 1, 2, \ldots, \lfloor \log_2(n) \rfloor\right\} \cup \left\{n - 2^{\lfloor \log_2(n) \rfloor} + 1\right\}$.
In this formulation, instead of representing numbers using default binary coefficients, we use a set of coefficients that precisely sum up to the number we want to represent. For example, to express 13_{10}, the coefficients are $\{2^0, 2^1, 2^2, 6\}$, allowing it to be represented as 1111_C. The advantage of this method for encoding task distribution among processes is that if all coefficients are used and each process will have a total number of tasks (both migrated and original) that adds up to exactly n. Without any additional constraints this ensures the solution's correctness, as all tasks must either be migrated or remain in their original process. The decision variable $x_{i,j,l}$ is defined as a binary variable, taking the value of 1 if and only if the amount of c_l tasks should be moved to process i from process j; otherwise $x_{i,j,l} = 0$.

Objective function: Our objective is to minimize the maximum load, L_{max}. In a scenario where the computational loads are well balanced across all processes, each process's load closely approximates L_{avg}. Consequently, the difference between each individual load L_i and L_{avg} can serve as a metric to assess the imbalance ratio. We utilize the square of load differences to capture the contribution of both underloaded and overloaded processes to this imbalance.

$$min \sum_{i=1}^{M} \left(\left(\sum_{j=1}^{M} w_j \cdot \sum_{l=1}^{|C|} x_{i,j,l} \cdot c_l \right) - L_{avg} \right)^2$$

Constraints: LRP has several constraints: First, for each process that contributed to the migrated load, the sum of migrated and retained tasks must equal $n \rightarrow$ no task is lost.

$$\sum_{i=1}^{M} \sum_{l=1}^{|C|} x_{i,j,l} \cdot c_l = n, \quad \forall j \in \{1, \ldots, M\}$$

The second constraint is that on each process the new load should not exceed the maximum load.

$$\sum_{j=1}^{M} w_j \sum_{l=1}^{|C|} x_{i,j,l} \cdot c_l \leq L_{max}, \quad \forall i \in \{1, \ldots, M\}$$

The final constraint is that no more than k tasks can be moved in total, ensuring that LRP differs from task pre-partitioning or task assignment. This is intended to cap and/or save the migration costs on real HPC systems.

$$\sum_{i=1}^{M} \sum_{j=1, j \neq i}^{M} \sum_{l=1}^{|C|} x_{i,j,l} \cdot c_l \leq k$$

Using the formulation of the objective function and the constraints described above, the required number of binary variables for constructing CQM is $M^2 \cdot (\lfloor \log_2(n) \rfloor + 1)$. This formulation involves M equality constraints and $M + 1$ inequality constraints. However, the number of binary variables can be reduced to $(M - 1)^2 \cdot (\lfloor \log_2(n) \rfloor + 1)$ by inferring the values of tasks that are not moved from process j based on the number of tasks that are moved. Specifically, for each process j, $x_{j,j} = n - \sum_{i=1, i \neq j}^{M} \sum_{l=1}^{|C|} x_{i,j,l} \cdot c_l$. By adopting this approach, the CQM can be reformulated to use fewer variables, while maintaining the same number of constraints, but all of the constraints will be the inequality constraints. For the following experiments, we use two variants of the quantum CQM formulation to indicate:

- Q_CQM1: the formulation with the reduction in the number of qubits, $(M - 1)^2 \cdot (\lfloor \log_2(n) \rfloor + 1)$.
- Q_CQM2: the formulation without reduction in the number of qubits, $M^2 \cdot (\lfloor \log_2(n) \rfloor + 1)$.

We emphasize the significance of the number of binary variables in the CQM, as it directly correlates with the number of qubits required. Assuming that representing inequality constraints does not require additional ancillary qubits, the number of logical qubits needed is equivalent to the number of binary variables. The CQM can be converted into a quadratic unconstrained binary optimization (QUBO) problem outlined by Glover et al, [23], who incorporates constraints into the objective function using penalty coefficients. The number of logical qubits for inequality constraints can remain unchanged using unbalanced penalization [24]. However, this study focuses exclusively on applying the CQM formulation.

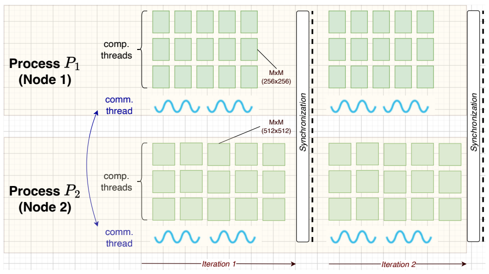

Fig. 2. An overview of Chameleon execution flow with task-based parallel applications, featuring an example of MxM tasks.

V. Experiments

A. Experimental Setup

The imbalance inputs are from experiments of matrix multiplication (MxM) and sam(oa)^2, which are performed on the CoolMUC2 cluster at Leibniz Supercomputing Centre (LRZ) featuring 28-way Intel Haswell-based nodes; 812 compute nodes, 28 cores per node. MxM is a synthetic example and Sam(oa)^2 is a realistic HPC application for numerical simulations. The experiments using the hybrid classical-quantum approach are conducted on D-Wave Cloud services using Leap's Hybrid CQM Solver, with access facilitated by Cyfronet AGH.

For MxM, the compute kernel $A = B \times C$ defines a task, where the matrix size affects the execution time of tasks. We can vary the task lengths by varying matrix sizes. Also, the number of tasks assigned to processes can be varied. In Subsection V-B, we show three groups of experiments:

- varying the imbalance ratios;
- varying the number of involved processes;
- varying the number of tasks assigned to each process.

The applied rebalancing methods are compared in speedup calculated by the fraction of the maximum load values (L_{max}) between baseline (no rebalancing) and rebalancing, imbalance ratio, the total number of migrated tasks, the average number of migrated tasks per process, and runtime overhead.

For sam(oa)^2, we run the simulation model of oscillating lake. Its implementation is based on the concept of adaptive mesh refinement. The simulated problem is simplified as a mesh, which is sub-partitioned into sections of cells. Cell traversal is a computation unit defining a task. Both sam(oa)^2 and MxM are ported into task-based applications using Chameleon. Conventionally, sam(oa)^2 already has partitioning and balancing algorithms. However, we assume an incorrect performance model at runtime, leading to an imbalance during execution. Therefore, rebalancing is essential, and task migration is mainly focused on rebalancing.

Chameleon: is a task-based parallel programming framework for shared and distributed systems [13], which is implemented as an extended library in C++. Parallel applications, which follow a bulk-synchronous paradigm, can be supported by Chameleon to enable overlapping computation and communication phases. Chameleon uses hybrid MPI+OpenMP,

TABLE I
AN OVERVIEW OF GENERAL COMPLEXITY AND LOGICAL QUBITS.

Algorithm	Complexity	Logical Qubits
Greedy	$O(N \log N)$ - $O(2^N)$	
KK	$O(N \log N)$ - $O(2^N)$	
ProactLB	$O(M^2K)$	
Q_CQM1_k1, _k2		$(M-1)^2(\lfloor \log_2(\frac{M}{N}) \rfloor + 1)$
Q_CQM2_k1, _k2		$M^2(\lfloor \log_2(\frac{M}{N}) \rfloor + 1)$

where compute-bound tasks express computation units. We can define independent tasks without side effects, such as access to global variables by multiple tasks. In Figure 2, the y-axis indicates one MPI process per node; a process can spawn multiple threads to execute tasks (`comp.threads`), where one thread is dedicated to communication (`comm.threads`). The x-axis shows the iterative execution direction, where each green box indicates a task; here, we denote an example of MxM tasks. With the dedicated threads, we can perform task migration overlapping with computation.

B. Complexity and Algorithmic Output

Table I shows the overview of the classical methods compared to the quantum-based methods. Greedy and Karmarkar-Karp (KK) are analyzed in time $O(N \log N)$ up to $O(2^N)$ for the worst-case time complexity [20]. In our experiments, the applied Greedy and KK purely focus on an optimal solution for partitioning tasks without considering migration cost. Furthermore, "partitioning" emphasizes that Greedy and KK consider imbalance inputs like multi-way number partitioning. ProactLB proposed by Chung et al [8] aims at the distributed view of load rebalancing that is more relevant to distributed memory systems; therefore, the number of migrated tasks and migration costs are taken into account (# migrated tasks in ProactLB is less than in Greedy and KK). The complexity of ProactLB depends on the number of involved processes (M), and K is supposed to be $\leq$ the number of tasks assigned to a process. K also means the search space for selecting how many tasks in a process should be migrated.

Regarding the hybrid classical-quantum methods, we explore two variants: Q_CQM1 (with qubit reduction) and Q_CQM2 (without qubit reduction). The parameter k represents the constraint on the number of migrated tasks in the CQM formulation, limiting the migration to $\leq k$ tasks. Finding an appropriate value for k is essential. In this study, we apply two values, $k1$ and $k2$, based on the number of migrated tasks determined by classical algorithms, which are run first to guide the hybrid experiments. Specifically, $k1$ corresponds to the tasks migrated using ProactLB, while $k2$ reflects the count from Greedy and KK. The experiments distinguish between these cases using $k1$ and $k2$ as postfixes. If using only the classical-quantum approach, a parameter study could be conducted by testing multiple values of k, as it is a discrete, bounded parameter. However, this requires the use of a large amount of hybrid resources.

1) Varying imbalance levels: This group of experiments includes different imbalance levels. We generate the tests on eight compute nodes with one MPI process per node. Each node is assigned 50 tasks, and tasks assigned to a node have a uniform load. The matrix size of tasks on different nodes is different, which we exploit to generate different imbalance cases. The matrix sizes of tasks are in the range of $\{128, 192, 256, \ldots, 512\}$.

TABLE II
COMPARISON OF THE AVERAGE NUMBERS OF TOTAL TASKS MIGRATED, TASKS MIGRATED PER PROCESS, AND RUNTIME OVERHEAD.

Algorithm	# total mig. tasks (avg)	# mig. tasks per process (avg)	Runtime (ms)
Greedy	351.8	43.98	0.5012
KK	351.4	43.93	2.5876
ProactLB	60.4	7.55	0.4490
Q_CQM*_k1	60.4	7.55	5231.02
Q_CQM*_k2	316.0	39.50	5246.67

Figure 3 shows the imbalance and speedup results after applying the rebalancing methods, where the x-axis denotes imbalance cases increasing from `Imb.0` to `Imb.4`, the y-axis of the left sub-figure shows imbalance ratios (R_{imb}), while the right sub-figure shows speedup values. R_{imb} and speedup are calculated from the rebalancing solution of the applied methods, which tasks should be migrated to which processes. `Imb.0` represents no imbalance, allowing for the assessment of different methods on whether or not to make task migration decisions. Overall, all methods help to reduce the imbalance close to 0. Q_CQM1_k2 reaches a balance similar to Greedy and KK. For speedup, all quantum-based methods also obtain results equivalent to those of classical methods.

In Table II, we show in detail the total number of migrated tasks (# total mig. tasks) and the number of migrated tasks per process (# mig. tasks per process) in average (avg) corresponding to the results of 5 imbalance cases shown in Figure 3. Column "Runtime" shows the average runtime when performing each algorithm. Although Greedy and KK obtain the optimal results, we have to move many tasks. In contrast, the number of migrated tasks in ProactLB is reduced because it takes migration overhead into account. With the quantum-based methods, Q_CQM*_k1 points to the number of migrated tasks and runtime on average for both Q_CQM1_k1 and Q_CQM2_k1 because they have the same problem size and the constraint of $k1$. Their runtime values are not much different. Similarly, Q_CQM*_k2 represents both Q_CQM1_k2 and Q_CQM2_k2. Interestingly, with $\leq k1$ migrated tasks, where $k1$ refers to the total number of migrated tasks in ProactLB, the performance is equal and even slightly better than the classical methods. In the case of $k2$, the constraint of migrated tasks is relaxed as $k2 > k1$. Thereby, the performance is improved while the number of migrated tasks is still less than Greedy, KK. For the runtime measurement of the classical methods, we simply set a timer before and after the algorithm is finished. In Q_CQM*_k1 and Q_CQM*_k2, the runtime indicates both CPU and QPU time, which charges for the time that the quantum annealing-based solvers run our problems. The runtime is quite large compared to the classical algorithms.

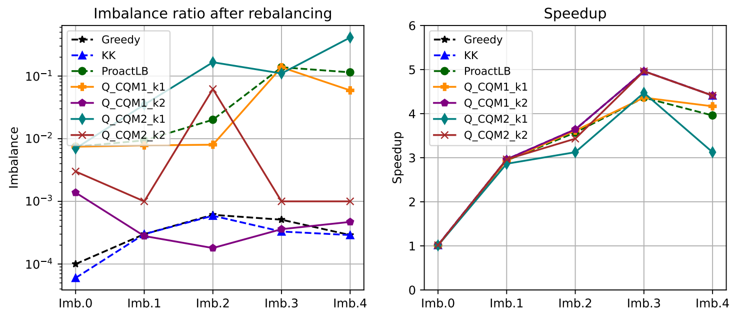

Fig. 3. A comparison of imbalance ratio and speedup in various imbalance levels.

TABLE III
THE TOTAL NUMBER OF MIGRATED TASKS IN VARYING NODE SCALES.

Algorithm	**4 nodes**	**8 nodes**	**16 nodes**	**32 nodes**	**64 nodes**
Greedy	300	700	1499	3105	6302
KK	300	700	1501	3098	6302
ProactLB	90	163	350	644	2353
Q_CQM1_k1	89	163	350	644	2353
Q_CQM1_k2	285	681	1482	3053	6298
Q_CQM2_k1	79	163	338	644	2353
Q_CQM2_k2	284	634	1434	3084	6300

TABLE IV
THE TOTAL NUMBER OF MIGRATED TASKS IN VARYING # TASKS.

Algorithm	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Greedy	56	112	224	448	896	1792	3584	7168	14336
KK	56	112	224	448	896	1792	3584	7168	14336
ProactLB	11	53	43	87	196	349	696	1407	2800
Q_CQM1_k1	11	53	43	87	196	349	696	1407	2800
Q_CQM1_k2	54	102	211	447	855	1781	3501	7049	14248
Q_CQM2_k1	11	51	43	76	194	333	694	1405	2758
Q_CQM2_k2	54	107	206	414	809	1584	3365	6657	11473

2) Varying the number of compute nodes: This group of experiments is set up by varying the number of compute nodes. Each node is assigned the same number of 100 uniform tasks. The different matrix sizes are again used to give the total load value of processes and imbalance levels.

Figure 4 depicts the imbalance and speedup results based on the outputs from the applied methods. Q_CQM1_k2 achieves results better than or equal to classical algorithms. The speedup values of Q_CQM1_k2 are competitive in the cases scaling up to 32 and 64 nodes. Q_CQM1_k1 and Q_CQM2_k2 also have positive results, while Q_CQM2_k1 is the worst. It uses more qubits (without qubit reduction), and $k1$ refers to the number of migrated tasks based on ProactLB.

Table III shows the total number of migrated tasks of each scale. Keeping the same number of tasks per node but increasing the number of processes affects the applied methods. For instance, the total number of migrated tasks is increased from 300 to over 6000 in the case of applying Greedy, KK. When setting the constraints $k1$, $k2$, the number of tasks migrated in the quantum-based methods is less than or equal. Furthermore, they perform better, especially in Q_CQM1_k2, as Figure 4 shows. This shows that the decision of the quantum-based methods on task selection can be clever.

3) Varying the number of tasks per node: In this group of experiments, we increase the number of tasks per node. We keep the experiments with the number of 8 nodes, while the number of assigned tasks per each varies from 8 to 2048.

Figure 5 also shows the imbalance ratio and speedup results calculated by the applied method's outputs. Q_CQM1_k1 (qubit reduction) slightly fluctuates when the number of assigned tasks is increased. Q_CQM1_k2 obtains good results close to KK and Greedy, especially in the cases 512, 1024, 2048 tasks. Q_CQM2_k1 (without qubit reduction) is the worst, while the same variant with $k2$ keeps positive performance. The quantum-based methods with more qubits introduce instability. The speedup values of all methods look approximately equal, except for Q_CQM2_k1. In Table IV, we show the total number of migrated tasks corresponding to each case. The constraint of $k1$ produces the same output as ProactLB. However, for $k2$, although the limit is relaxed, the final output of Q_CQM1_k2 and Q_CQM2_k2 is slightly better than KK and Greedy in terms of migration cost.

TABLE V
EXPERIMENTAL RESULTS FROM THE IMBALANCE USE CASE OF SAM(OA)^2.

Algorithm	R_{imb}	**Speedup**	**# mig. tasks**	**Runtime (ms)**	
				CPU	QPU
Baseline	4.19940	1.0			
Greedy	0.00007	5.19905	6447	8.14	
KK	0.00001	5.19938	6447	109.63	
ProactLB	0.00944	5.15076	1568	1.32	
Q_CQM1_k1	0.0001	5.1990	1567	19349.3	32.1
Q_CQM1_k2	0.0001	5.1990	6418	19359.1	32.0
Q_CQM2_k1	2.3192	1.5664	1550	19965.8	32.0
Q_CQM2_k2	0.0001	5.1989	6440	19372.3	32.1

C. Application in Realistic Use Case

This experiment uses the imbalance input from a realistic use case, sam(oa)^2, to simulate an oscillating lake on 32 compute nodes. The problem size of this use case is configured by default, and it sub-partitions into 16 sections. The number of tasks on each node is 208 with uniform load. The application itself maintains the imbalanced nature of this use case.

Table V shows the summarized results, also including the imbalance ratio (R_{imb}) and speedup calculated from the output of the applied methods. Column "# mig. task" denotes the total number of migrated tasks. Column "Runtime" shows the

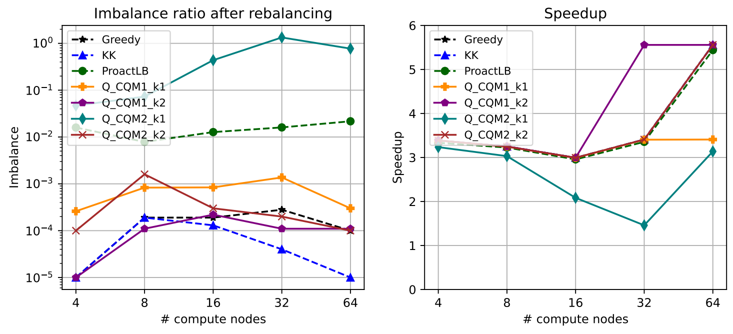

Fig. 4. A comparison of imbalance and speedup when varying the number of processes.

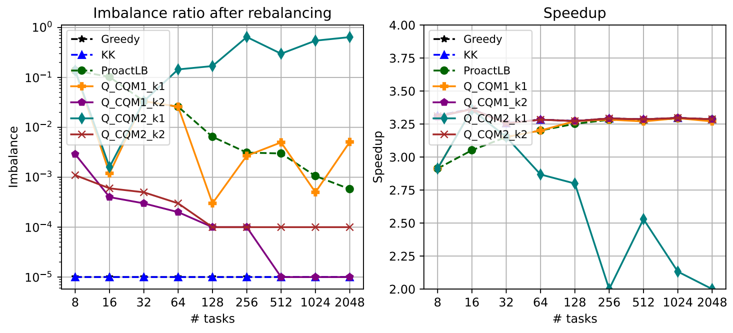

Fig. 5. A comparison of imbalance and speedup when increasing the number of tasks per node.

elapsed time in milliseconds (ms) as the runtime overhead of each method. The runtime values for Q_CQM1_k1, _k2 and Q_CQM2_k1, _k2 are divided into CPU and QPU measurements. The significant CPU runtimes values highlight a considerable overhead and latency compared to classical methods, with a portion of this time dedicated to communication with D-Wave's Leap quantum cloud service.

VI. Discussion

Based on the results from our experiments, the quantum-based methods show interesting solutions, especially in solving the LRP applied to task-parallel applications. Q_CQM1_k1 returns equivalent or even better solutions for LRP, in which the number of migrated tasks is reduced. Interesting to note is that the results obtained using the CQM solver with fewer qubits, but a higher number of inequality constraints (the total number of constraints is the same) generally outperform those stemming from more qubits and fewer inequality constraints. This observation is notable given the challenge of representing inequality constraints effectively in quantum formulations. However, the superior performance with fewer qubits is expected, as it corresponds to a smaller search space. Future work will explore the impact of the upper bound k of migrated tasks and different problem formulations, as well as the challenges related to runtime costs.

Regarding scalability, the number of qubits currently available in real devices is a challenge. In our formulation, the number of required logical qubits is $(M-1)^2 \times (\lfloor \log_2(n) \rfloor + 1)$, where M is the number of processes and n is the number of tasks per node. This challenge becomes more significant when the number of tasks reaches up to millions or billions. Further, noise and error mitigation models must also be considered as we increase the problem size.

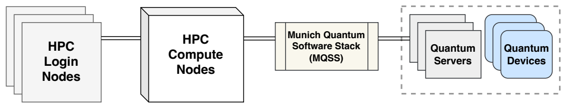

Fig. 6. Overview of HPCQC integration system at LRZ.

Finally, the design of HPC-Quantum Computing (HPCQC) integration systems is a major point for further investigation. Figure 6 shows our current overview of HPCQC integration at LRZ. The HPC clusters have connections to quantum servers, which are hosting quantum backend devices through a quantum software stack known as the Munich Quantum Software Stack (MQSS) [25]. This stack connects domain end users to a variety of quantum devices and supports quantum circuit optimization, compilation, and hybrid execution workflows. The integration motivates a hybrid execution model of the proposed quantum-based method in this paper. Concretely, quantum devices are considered accelerators alongside HPC systems. The hybrid model of our Q_CQM* methods can be extended to use gate-based quantum solvers.

VII. Conclusion

In this paper, we presented a formulation for load rebalancing in HPC systems that is specifically designed for leveraging quantum devices. Load rebalancing is an NP-hard problem that can be modeled as a combinatorial optimization task, similar to number partitioning. Our methods produced solutions that are particularly valuable in guiding task migration strategies that match or even surpass those from classical approaches like Greedy, KK, and ProactLB. When task migration costs are a key constraint, represented in our study by an upper bound of k migrated tasks, we showed that hybrid classical-quantum methods offered better guidance, resulting in more efficient task migration. Overall, we highlighted the promise of hybrid classical-quantum approaches and outlined the challenges and future work toward advancing these models.

Acknowledgements

At Cyfronet, we gratefully acknowledge the Polish high-performance computing infrastructure PLGrid (HPC Centers: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017426. At AGH, the research project was partly supported by program "Excellence initiative – research university" for the AGH University of Krakow. At LRZ, this work is funded by the German Federal Ministry for Education and Research through the MUNIQC-Atoms project, and the Bavarian State Ministry of Science and the Arts as part of Munich Quantum Valley (MQV) and the Bavarian High-Tech agenda. The authors acknowledge LRZ for providing computation time on HPC systems.

References

[1] H. Jin, D. Jespersen, P. Mehrotra *et al.*, "High performance computing using mpi and openmp on multi-core parallel systems," *Parallel Computing*, vol. 37, no. 9, pp. 562–575, 2011. [Online]. Available: https://doi.org/10.1016/j.parco.2011.02.002

[2] P. Thoman, K. Dichev *et al.*, "A taxonomy of task-based parallel programming technologies for high-performance computing," *The Journal of Supercomputing*, vol. 74, no. 4, pp. 1422–1434, 2018. [Online]. Available: https://doi.org/10.1007/s11227-018-2238-4

[3] M. Garcia-Gasulla, G. Houzeaux *et al.*, "MPI+X: task-based parallelisation and dynamic load balance of finite element assembly," *International Journal of Computational Fluid Dynamics*, vol. 33, no. 3, pp. 115–136, 2019. [Online]. Available: https://doi.org/10.1080/10618562.2019.1617856

[4] G. Aggarwal, R. Motwani, and A. Zhu, "The load rebalancing problem," *Journal of Algorithms*, vol. 60, no. 1, pp. 42–59, 2006. [Online]. Available: https://doi.org/10.1016/j.jalgor.2004.10.002

[5] S. Mertens, "The Easiest Hard Problem: Number Partitioning," in *Computational Complexity and Statistical Physics*, A. Percus, G. Istrate, and C. Moore, Eds. New York: Oxford University Press, 2005. [Online]. Available: https://doi.org/10.1093/oso/9780195177374.003.0012

[6] R. E. Korf, "A complete anytime algorithm for number partitioning," *Artificial Intelligence*, vol. 106, no. 2, pp. 181–203, 1998. [Online]. Available: https://doi.org/10.1016/S0004-3702(98)00086-1

[7] N. Karmarker and R. M. Karp, "The Differencing Method of Set Partitioning," UC Berkeley, USA, Tech. Rep. UCB/CSD-83-113, 1983. [Online]. Available: https://dl.acm.org/doi/10.5555/894426

[8] M. T. Chung, J. Weidendorfer, K. Fürlinger, and D. Kranzlmüller, "From reactive to proactive load balancing for task-based parallel applications in distributed memory machines," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 24, p. e7828, 2023. [Online]. Available: https://doi.org/10.1002/cpe.7828

[9] H. Menon and L. Kalé, "A distributed dynamic load balancer for iterative applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, November 2013, pp. 1–11. [Online]. Available: https://doi.org/10.1145/2503210.2503284

[10] A. Tikin, *Bulk-Synchronous Parallelism: An Emerging Paradigm of High-Performance Computing*. New York, United States: John Wiley & Sons, Ltd, 2005, ch. 4, pp. 69–80. [Online]. Available: https://doi.org/10.1002/0471732710.ch4

[11] L. Rannabauer *et al.*, "ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework," *Computers & Fluids*, vol. 173, pp. 299–306, 2018. [Online]. Available: https://doi.org/10.1016/j.compfluid.2018.01.031

[12] O. Meister, K. Rahnema, and M. Bader, "Parallel Memory-Efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells," *ACM Trans. Math. Softw.*, vol. 43, no. 3, pp. 1–27, 2016. [Online]. Available: https://doi.org/10.1145/2947668

[13] J. Klinkenberg, P. Samfass *et al.*, "CHAMELEON: Reactive Load Balancing for Hybrid MPI+OpenMP Task-Parallel Applications," *Journal of Parallel and Distributed Computing*, vol. 138, pp. 55–64, 2020. [Online]. Available: https://doi.org/10.1016/j.jpdc.2019.12.005

[14] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, pp. 279–301, 1989. [Online]. Available: https://doi.org/10.1016/0743-7315(89)90021-X

[15] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *Journal of Parallel and Distributed Computing*, vol. 32, no. 2, p. 445–465, 1985. [Online]. Available: https://doi.org/10.1145/3149.3156

[16] G. Menon and H. Menon, "Adaptive load balancing for HPC applications," Ph.D. dissertation, UIUC, USA, October 2016. [Online]. Available: https://hdl.handle.net/2142/95292

[17] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 720–748, 1999. [Online]. Available: https://doi.org/10.1145/324133.324234

[18] S. Li, J. Hu, X. Cheng, and C. Zhao, "Asynchronous Work Stealing on Distributed Memory Systems," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Belfast, UK, February 2013, pp. 198–202. [Online]. Available: https://doi.org/10.1145/2503210.2503284

[19] P. Samfass, J. Klinkenberg *et al.*, "Predictive, reactive and replication-based load balancing of tasks in Chameleon and sam(oa)2," in *Proceedings of the PASC Conference*, Geneva, Switzerland, July 2021, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3468267.3470574

[20] E. L. Schreiber, R. E. Korf, and M. D. Moffitt, "Optimal Multi-Way Number Partitioning," *J. ACM*, vol. 65, no. 4, 2018. [Online]. Available: https://doi.org/10.1145/3184400

[21] R. L. Graham, "Bounds for certain multiprocessing anomalies," *The Bell System Technical Journal*, vol. 45, no. 9, pp. 1563–1581, 1966. [Online]. Available: https://doi.org/10.1002/j.1538-7305.1966.tb01709.x

[22] O. Rathore, A. Basden *et al.*, "Load Balancing For High Performance Computing Using Quantum Annealing," arXiv, March 2024. [Online]. Available: https://arxiv.org/abs/2403.05278

[23] F. Glover, G. Kochenberger, R. Hennig, and Y. Du, "Quantum bridge analytics I: a tutorial on formulating and using QUBO models," *Annals of Operations Research*, vol. 314, no. 1, pp. 141–183, 2022. [Online]. Available: https://doi.org/10.1007/s10479-022-04634-2

[24] J. A. Montañez-Barrera, D. Willsch *et al.*, "Unbalanced penalization: a new approach to encode inequality constraints of combinatorial problems for quantum optimization algorithms," *Quantum Science and Technology*, vol. 9, no. 2, p. 025022, 2024. [Online]. Available: http://dx.doi.org/10.1088/2058-9565/ad35e4

[25] M. Schulz, L. Schulz, M. Ruefenacht, and R. Wille, "Towards the munich quantum software stack: Enabling efficient access and tool support for quantum computers," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Bellevue, WA, USA, November 2023, pp. 399–400. [Online]. Available: https://doi.org/10.1109/QCE57702.2023.10301

Appendix

A. Artifact Description/Artifact Evaluation

In this paper, we present a formulation of the Load Rebalancing Problem (LRP) that can be solved in a hybrid classical-quantum solver. The formulation can be extended to work with gate-based quantum solvers, as mentioned in Section VI - Discussion of the paper. LRP revolves around the input: N tasks in a task-based parallel application already assigned to M processes running on distributed memory systems. The case studies demonstrate:

- Each process is assigned an equal number of tasks, n, where $n \times M = N$;
- the execution time value of a task defines load;
- Assuming that tasks in a process have a uniform load but the task load values from different processes are different, this leads to an imbalance.

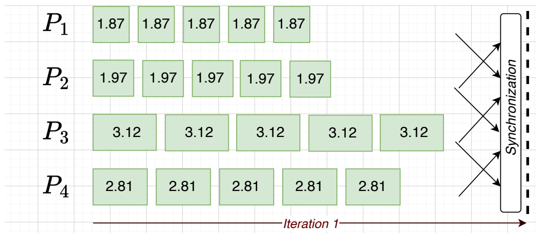

Fig. 7. An example of load imbalance in the case of task-based parallel applications.

For example, Figure 7 shows an imbalance case; note that the load values per task of P_1 and P_2 are reused as an illustration for the input format in Table VI in the next Subsection B. Figure 7 does not map directly to Table VI to avoid confusion. Generally, Figure VI is simply intended to illustrate a real-world example of load imbalance at runtime. There are 4 MPI processes; each is assigned 5 tasks with a uniform load. The load per task in process 1 is assumed 1.87 (ms), and 1.97, 3.12, 2.81 for processes 2, 3, 4 respectively. In overall, the total load values are 9.35, 9.85, 15.6, 14.05, where process 3 has the maximum load. When tasks are migrated appropriately, the maximum load value is reduced, and we can obtain the balance. This is a simple example to show the target of the LRP and how we can approach to solve it.

We use two applications to generate the imbalance inputs: matrix multiplication (MxM) and sam(oa)^2. They are ported to task-based parallel applications by using Chameleon, https://github.com/chameleon-hpc/chameleon. The number of tasks and task lengths in MxM can be reproduced, following the example repository https://github.com/chameleon-hpc/chameleon-apps/tree/master/applications. For sam(oa)^2, we refer to the instruction at https://github.com/chameleon-hpc/samoa-chameleon.git. In this paper, the logs, as well as imbalance inputs, are already synthesized. The code repository associated with this work is available at https://github.com/ctminh/qulrb. Please note that the repository is actively maintained and might be updated or improved over time as new insights are gained and additional features are developed. The repository is structured as follows:

- `docs/`: includes related documents and references.
- `experiments/`: includes log files from the experiments of MxM and sam(oa)^2 with Chameleon. In which, there are different sub-folders corresponding to each experiment group to show the synthesized inputs and outputs for each test case in the paper. Furthermore, the scripts for extracting input data, output data, and plotting are also included.
- `qubo/`: includes our explanation and example for the current CQM formulation of LRP. Also, this folder is used for the working-in-progress QUBO formulation as future work.
- `src/`: include the source code of classical algorithms and the source of porting LRP to be run on a hybrid classical-quantum solver.
- `utils/`: include related scripts for testing.

B. Input/Output Format

Each imbalance input is synthesized to a table and formatted as a `.csv` file. Concretely, Table VI shows an input example that is the content in the corresponding `.csv` file. Here, we use the case of MxM with 100 tasks assigned per process, where one process is spawned in a compute node. The columns and rows labelled $P1$, $P2$, $P3$, $P4$ form a symmetric matrix. Its diagonal indicates the original number of tasks in a process. When tasks are migrated, these numbers are changed. Accordingly, the total load values on column “L” will also be changed. Alongside, column “w” denotes the load value of each task per process.

TABLE VI
The format of imbalance input.

Process	**P1**	**P2**	**P3**	**P4**	**w**	**L**
P1	100	0	0	0	1.87	187.59
P2	0	100	0	0	1.97	196.68
P3	0	0	100	0	14.86	1485.99
P4	0	0	0	100	103.23	10322.68

For the output data, assuming we apply Greedy to solve the input, the output is shown in Table VII. We keep the table similar to the input format; the diagonal now indicates the remaining number of original tasks on each process. For example, at column “**P1**”, there are 25 remained tasks, and P1 migrates 25 tasks to P2, 25 to P3, and 25 to P4. The columns “num_total”, “num_local”, “num_remote” are used for cross-checking the total number of tasks, local tasks (original tasks), and remote tasks (migrated tasks or received tasks from the other processes). Column “L” shows the new total load value on each process.

According to the experiment groups in Section V of the paper, there are 4 corresponding sub-folders at `experiments/` that further consist of imbalance log files obtained from the experiments with Chameleon, input/output data, and related

TABLE VII
THE OUTPUT FORMAT AFTER REBALANCING, WHERE THIS CASE APPLIES THE GREEDY ALGORITHM.

Process	**P1**	**P2**	**P3**	**P4**	**num _total**	**num _local**	**num _remote**	**L**
P1	**25**	25	25	25	100	25	75	3048.24
P2	25	**25**	25	25	100	25	75	3048.24
P3	25	25	**25**	25	100	25	75	3048.24
P4	25	25	25	**25**	100	25	75	3048.24

Python scripts. In details, the 4 corresponding sub-folders include:

- `real_usecase_samoa/` points to the experiments in Section V.C.
- `varied_imb_ratios/` points to the experiments in Section V.B.1)
- `varied_num_procs/` points to the experiments in Section V.B.2)
- `varied_num_tasks/` points to the experiments in Section V.B.3)

In each experiment group, the folders are structured by `cham_logs/` that contains the Chameleon logs where we get the imbalance inputs based on the execution of MxM or sam(oa)^2 on HPC systems. `input_lrp/` contains the formatted inputs as `.csv` files, where the input values are extracted from the Chameleon logs (by using the Python script, `cham_log_parser.py`). `output_lrp/` contains the formatted outputs after running the rebalancing algorithms. After applying the rebalancing algorithms, the summary table of all outputs can be collected using `extract_rimb_speedup.py`.

C. Reproducibility of Experiments

The provided source code for classical algorithms and quantum-based algorithms can be found in folder `src/`. We keep the classical algorithms in sub-folder `classical_algorithms/`, and the quantum-based algorithms in sub-folder `hybrid_quantum_algorithms/`.

1) In `classical_algorithms/`, three classical algorithms are provided by Python. They are invoked in the `main.py` file. Corresponding to the experimental use cases at `experiments/`, there are different bash scripts to reproduce the results, including
   - `run_real_usecase_samoa.sh` applies to the imbalance input of sam(oa)^2 (at `real_usecase_samoa/input_lrp`) by running on 32 processes, 208 tasks per process, and the baseline of imbalance ratio is 4.1994. All classical algorithms will be called in this bash script. The results output at `real_usecase_samoa/output_lrp`
   - `run_varied_imb_levels.sh` applies to the imbalance inputs at `varied_imb_ratios/input_lrp`. Accordingly, the results output at `varied_imb_ratios/output_lrp`.
   - `run_varied_num_procs.sh` is called and outputs the results similar to the above scripts.
   - `run_varied_num_tasks.sh` is called and outputs the results similar to the above scripts.
2) In `hybrid_quantum_algorithms/`, we provide a notebook file with instructions and steps to connect and run the problem on the D-Wave CQM solver. However, the access and computation time should registered at https://docs.dwavesys.com/docs/latest/handbook_hybrid.html. The paper's results from CQM solvers are put in the sub-folder `poc/` of `hybrid_quantum_algorithms/`. There are two outputs from the two versions of our quantum CQM formulation, with qubit reduction and without qubit reduction.

Please note that the results of using D-Wave's CQM solver to solve our LRP formulation are not deterministic. We ran each experiment with the CQM solver at least three times. While there is some variation from run to run, the results are not significantly skewed. Therefore, in this paper, we select the best results for the corresponding experiments.