A Model for Query Execution Over Heterogeneous Instances

Ziheng Wang
Stanford University
zihengw@stanford.edu

Emanuel Adamiak
Stanford University
adamiak@stanford.edu

Alex Aiken
Stanford University
aaiken@stanford.edu

ABSTRACT

Decreasing VM start-up times and recent trends in serverless computing on public clouds now allow users to spin up a dedicated cluster for an SQL query, in contrast to the longstanding paradigm of submitting queries to a fixed cluster. This new capability places additional responsibility on the query planner, which now must recommend the most cost-efficient cluster configuration for a given query. While the space of potential cluster configurations is immense, recent work has mostly focused on homogeneous clusters that consist of only one instance type. We argue that to truly leverage the flexibility of public clouds, we need to consider heterogeneous clusters consisting of multiple instance types. In this paper, we present a framework to gauge the optimality of cluster configurations, an intuitive model showcasing how heterogeneity leads to performance improvements for pipelined joins, and preliminary experimental evidence supporting the model.

1 INTRODUCTION

While a large volume of work has focused on optimizing distributed query execution on a fixed cluster configuration, we focus on the emerging problem of ***optimizing the cluster configuration for a particular query***. This issue is gaining relevance as the fast VM spin-up time on public clouds has made it practical to spin up a dedicated cluster for each query instead of sharing a fixed cluster among queries, which enables highly desirable performance and security isolation amongst different data processing jobs. While already viable for long-running queries prevalent in data pipelines, new developments in micro-VMs promise to unlock this potential even for interactive queries [3].

The performance and cost-efficiency of a distributed query engine on any particular query could vary drastically depending on the VM instance types on which it is executed. EMR, the hosted Hadoop service offering SparkSQL and Trino on AWS, supports more than 400 VM options. Instance types on cloud providers like AWS belong to different classes, some optimized for compute with a higher vCPU to memory ratio, while others are tailored for IO performance. Figure 1 displays the computed cost per GB of memory and cost per Gbps of network bandwidth of different instance types on AWS. There can be a 14.7x difference in dollar cost per GB of RAM and a 61.7x (!) difference in dollar cost per Gbps of bandwidth between instances. Even among the much smaller set of instance types typically used for Spark or Trino clusters in practice, we commonly observe 3x differences.

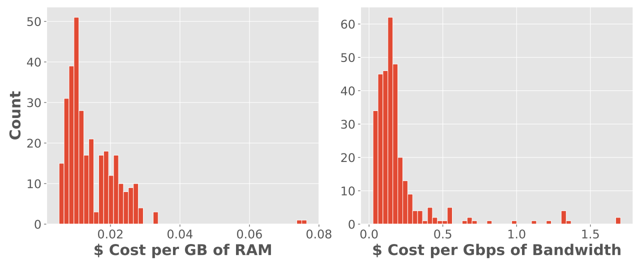

Figure 1: Histograms of memory and bandwidth costs of available instance types on AWS.

To the best of our knowledge, all current work on optimizing the cluster configuration for query processing has focused on *homogeneous clusters*, where the distributed query engine executes the entire query on a cluster consisting of a single instance type. Of particular interest is recent work by Leis and Kuschewski that proposes a mental model for optimizing instance type selection based on the Pareto optimal frontier of the query's total cost and the total execution time [13].

While an important first step, we believe homogeneous clusters do not fully exploit the resource flexibility offered on the public cloud today. This work studies the optimal cluster configuration problem for *heterogeneous clusters*, which might contain more than one instance type. We make the following contributions:

- We propose a Pareto optimal framework for evaluating cluster configurations in the context of heterogeneous clusters based on iso-cost curves, instead of the iso-instance curves proposed in [13].
- We apply the framework to left-deep join trees in pipelined query engines and provide intuition on why heterogeneous clusters are beneficial for cost efficiency.
- We evaluate heterogeneous clusters in a real pipelined query engine, showing performance benefits over homogeneous clusters on an example join.

2 WHAT IS OPTIMAL?

In this section, we propose a method to evaluate the optimality of cluster configuration given a fixed query. Let us assume we have a distributed query engine and a query we would like to execute on some fixed input data. The distributed engine could be run on a *homogeneous* cluster, consisting of one instance type, or a *heterogeneous* cluster, with multiple different instance types.

The key question is: **How do we know the cluster configuration we picked is the best one?**

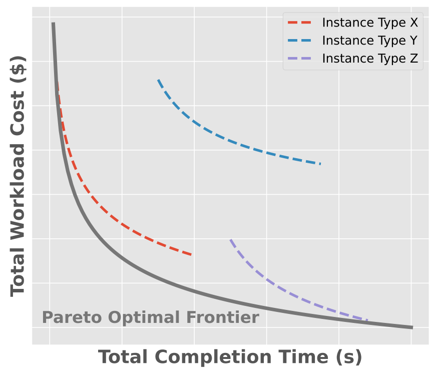

Figure 2: The model in [13] based on iso-instance curves.

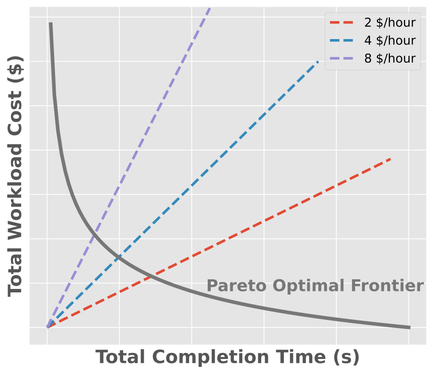

Figure 3: Our proposed framework based on iso-cost curves.

2.1 Pareto Optimality

To answer this question, we expand on the framework developed by Leis and Kuschewski [13], which only considered homogeneous clusters. The framework considers a graph with the x-axis representing the total execution time of the query and the y-axis representing the workload cost, as shown in Figure 2. Each point on the plot corresponds to a cluster configuration: if we assume that, given a cluster configuration, there is one way for the query engine to execute the query, then each configuration has exactly one fixed total completion time and workload cost.

The *Pareto optimal frontier* represents the lowest total workload cost for a fixed completion time. We should aim to pick cluster configurations that lie on this frontier. The Pareto frontier is parameterized by the query engine, the input data, and the query. Enhancing the engine's efficiency or simplifying the query shifts the curve to the left and vice versa.

Leis and Kuschewski propose a model that predicts how the total workload cost and completion time vary as a function of the number of VMs in a homogeneous cluster for a specific instance type. This approach results in an *iso-instance* curve in the graph for each instance type. An iso-instance curve could intersect the Pareto frontier at more than one point, as shown in Figure 2.

2.2 Iso-instance vs. Iso-cost Curves

If we consider heterogeneous clusters, there are a very large number of such curves to draw since the potential cluster configurations grow exponentially with the number of different instance types considered. An alternative approach is to consider cluster configurations with the same cost per hour instead of the same instance type. Cluster configurations with the same unit-cost can be described by a straight line from the origin as shown in Figure 3, with the slope equal to the cost per hour of the cluster configurations on that line. We will refer to these lines as *iso-cost curves* (alternatively iso-cost lines).

An iso-cost curve by itself is not tied to a specific query and is purely based on available cluster configurations by the cloud provider and instance costs. Given the query, the runtimes of the cluster configurations described by the curve will map to a set of points on this curve. However, the Pareto frontier is tied to a specific query, and it intersects these iso-cost curves at different points for different queries. We will show some real examples of iso-cost curves in Section 4.

An important benefit of thinking about cluster configurations in terms of these *iso-cost* curves is that, unlike iso-instance curves, each iso-cost curve intersects the Pareto frontier at exactly one point. In other words, among all possible cluster configurations with a particular cost per hour, there is only one cluster configuration that is optimal for a particular query.

In a heterogeneous cluster, a single cluster configuration could map to multiple points along the iso-cost curve if the query engine explores different *mapping strategies* of the query onto the processors, i.e. using different instance types for different stages[1]. Therefore, the optimal point on an iso-cost curve corresponds to the optimal mapping strategy of the optimal cluster configuration.

We can thus approach our key question using constrained optimization: **Among cluster configurations with the same cost per hour, which is the most efficient for the given query engine and query?**

2.3 Heterogeneity = More Choice

We hypothesize that the optimal point along an iso-cost curve is more likely to correspond to a heterogeneous cluster than a homogeneous cluster. To see why, let us introduce some formalism to describe a cluster configuration.

We assume our cluster is composed of instances from a set of N instance types, described by length-N vectors $\mathbf{c}$, $\mathbf{cpu}$, $\mathbf{mem}$, and $\mathbf{io}$. These vectors represent the per-hour cost, number of vCPUs, available RAM, and network bandwidth of each instance type. The cluster configuration can then be described by another vector $\mathbf{n}$ denoting how many of each instance type is in the cluster.

If we assume that the runtime for the optimal mapping strategy of a cluster configuration is a function H of the cluster's total resources, then we arrive at the following constrained optimization problem for the optimal configuration along an iso-cost curve:

[1]This observation applies to homogeneous clusters as well, though it is not considered in [13].

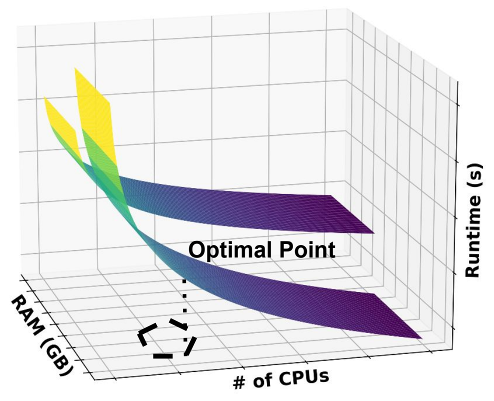

Figure 4: Optimizing H over the constraint polytope. For simplicity, the IO axis is ignored in the plot.

$$\begin{aligned} \underset{\mathbf{n}}{\text{minimize}} \quad & H(\mathbf{cpu}\cdot\mathbf{n}, \mathbf{io}\cdot\mathbf{n}, \mathbf{mem}\cdot\mathbf{n}) \\ \text{subject to} \quad & \mathbf{c}\cdot\mathbf{n} = C \end{aligned} \tag{1}$$

We define new variables for total resources, $cpu = \mathbf{cpu}\cdot\mathbf{n}$, and similarly for mem and io. If we also define the set of all possible combinations of these values given the total cost constraint to be P, then we can rewrite Equation 1 in terms of total resources:

$$\begin{aligned} \underset{cpu,\, mem,\, io}{\text{minimize}} \quad & H(cpu, mem, io) \\ \text{subject to} \quad & (cpu, mem, io) \in P \end{aligned} \tag{2}$$

The problem boils down to optimizing a potentially hard-to-evaluate function H over a constraint set P, as shown in Figure 4. We refer to the constraint set as the *resource constraint polytope*. The benefit of using heterogeneous clusters lies in greatly expanding the size of this polytope by leveraging the vastly different resource-per-unit-cost characteristics of cloud instances. Note as shown in Figure 4, H might not be continuous, but should be generally decreasing with increasing resources.

This model can be extended to incorporate resource types other than vCPU, IO, and total memory to account for different vCPU types like Gravitron or AMD. The constraint polytope would remain linear if the amount of this new resource increases linearly with the number of instances.

Figure 5 shows the significant resource flexibility achieved through heterogeneous instances, specifically in terms of total vCPU and RAM. The different colors represent constraint polytopes corresponding to different total cluster costs. For instance, the purple region in Figure 5 represents all possible combinations of total vCPU and RAM achievable for both (a) heterogeneous clusters and (b) homogeneous clusters with a total cost of \$4/hour, using three different instance types. Homogeneous clusters provide only three choices, where the maximum number of instances available for each type is selected within the cost constraint. In contrast, heterogeneous cluster configurations "connect" the vertices identified by the homogeneous clusters with the same cost.

Importantly, instead of allowing the users to obtain more of a particular resource type like total vCPUs for a particular price,

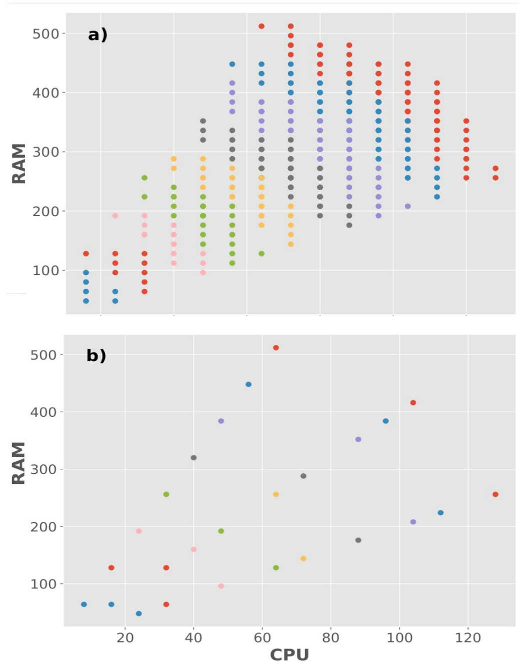

Figure 5: Available total resource combinations for fixed total cost for a) heterogeneous clusters and b) homogeneous clusters. Different colors correspond to different total cost per hour from \$.5 to \$5/hour in increments of \$0.5 from lower left to upper right.

heterogeneity offers more fine-grained *flexibility* in terms of trade-offs between different resource types.

3 WHY IS MORE CHOICE BENEFICIAL?

Increasing the space of potential total resource combinations does not guarantee enhanced query performance. For example, if the query engine is entirely CPU-bound on a query, the optimal cluster configuration should consist solely of the instance type with the lowest cost per vCPU per hour.

As illustrated in Figure 5, if the best performance is achieved by maximizing one resource type, homogeneous clusters would suffice. However, real query engines typically exhibit different resource demands when executing different parts of a query, which makes the increased flexibility in navigating the total resource space beneficial.

3.1 Query and Query Engine

To illustrate the potential benefits of this flexibility, we focus on the query execution of multi-stage joins in pipelined query engines, such as Snowflake, SingleStore, Trino, and DuckDB [6, 8, 12, 19]. These queries are ubiquitous in production data pipelines.

The query planner typically executes a multi-stage join with a left-deep join tree as shown in Figure 6, which executes a series of distributed hash joins in a pipeline. Typically, the largest table is selected as a *probe table*, and the other tables serve as *build tables*. The join is executed in two phases. In the *build* phase, build tables

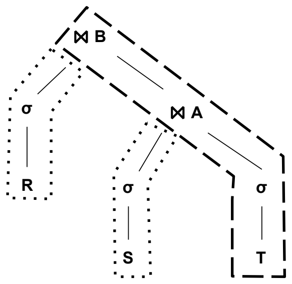

Figure 6: A left deep join tree.

R and S are read in parallel and hashed in memory with possible disk spilling. In the *probe* phase, probe table T is read and joined against the pre-built hash tables in a pipeline.

In a typical star-schema data warehouse, the build tables are a lot smaller than the probe table, which leads to the probe phase being the bottleneck. In this work, we will focus on the performance of the probe phase. Consequently, our key question becomes very specific: **among cluster configurations on a fixed iso-cost curve, which is the best for the probe phase pipeline in a multi-stage join?**

3.2 A Concrete Example

We have seen that different cluster configurations allow us to navigate within the resource constraint polytope to optimize H as given in Equation 2. However, up until this point we have avoided speculating on the shape of H. How do we relate the total amount of resources to the runtime of the query?

For simplicity, let's consider just a two-stage probe phase pipeline, for example, this SQL query on the TPC-H dataset.

Listing 1: Example join query

```
SELECT sum(l_quantity) as sum_qty,
       sum(l_extendedprice) as sum_base_price,
       sum(l_discount) as sum_disc,
       sum(l_tax) as sum_tax,
       max(l_shipdate) as max_shipdate,
       max(l_commitdate) as max_commitdate,
       max(l_receiptdate) as max_receiptdate,
       sum(o_totalprice) as sum_charge,
       max(o_orderdate) as max_orderdate
FROM lineitem, orders
WHERE l_shipmode in ('SHIP', 'MAIL')
  and l_orderkey = o_orderkey;
```

Assuming `lineitem` is the probe table, a pipelined query engine like Trino would execute the probe phase pipeline as a sequence of a scan stage followed by a join stage [12, 19]. Batches produced by tasks in the scan stage can (and should) be consumed immediately by tasks in the join stage to avoid materializing the intermediate results as much as possible. The two stages effectively proceed concurrently, sharing resources in the cluster.

To understand how the performance of these two stages varies with the amount of vCPU, memory, and IO assigned to each stage, we benchmark the performance of the scan stage and the join stage independently. We assign various combinations of vCPU, memory, and IO resources to each stage, using the query shown in Listing 1 on TPC-H SF-100 with input in Parquet format stored on AWS S3.

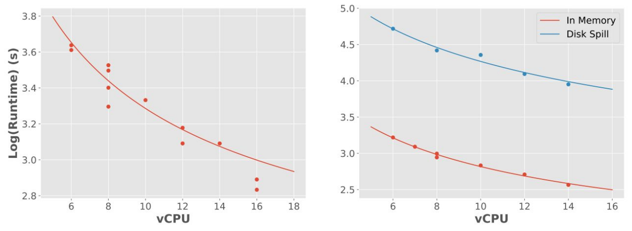

Figure 7: Performance for A) scan stage B) join stage as a function of assigned vCPU.

We use an open-source distributed query engine Quokka in our experiments [16]. Quokka's architecture resembles that of Trino, where tasks belonging to different stages in a pipeline are scheduled to executors on different machines by a centralized scheduler. On the TPC-H query benchmark, Quokka is able to achieve competitive results with Trino and SparkSQL. We select Quokka because it offers a simple interface to spin up heterogeneous clusters and it allows us to assign different resources to different query stages, a functionality missing in Trino or SparkSQL.

The benchmark results are shown in Figure 7. For the scan stage, we find that on the instance types we selected, the workload is entirely CPU-bound. While one might expect network bandwidth to be the bottleneck, Quokka relies on open-source Parquet readers, which cannot saturate the high available network bandwidth per core. High-performance query engines like Clickhouse and DuckDB have developed their own highly optimized Parquet readers to mitigate this issue [7, 18], but other query engines like SparkSQL and Trino share this problem. In practice, we observe no performance difference when using network-optimized AWS instances and regular instances on TPC-H-like workloads on Trino, SparkSQL, or Quokka. In terms of memory, a fixed amount of memory is required by the tasks to parse Parquet files. Additional memory does not lead to a speedup.

The join stage's performance is dominated by both the available memory and the number of vCPUs. When there is not enough memory, Quokka has to spill the build side to disk, leading to substantial performance degradation. For both disk-based and in-memory joins, increasing the number of vCPUs results in increased performance. Like other open-source query engines, Quokka does not support graceful performance degradation.[2] Once the necessary amount of RAM is available to perform the in-memory join, additional RAM does not improve performance. Similar to the scan stage, we did not observe much impact on performance from the network bandwidth.

These results suggest that the shape of H resembles the surface in Figure 4: there is a qualitative shift in the shape of the function when there is enough RAM to perform an in-memory join. This analysis suggests a simple heuristic to select the optimal cluster configuration:

- If the largest total memory in the constraint polytope is less than what is needed for an in-memory join, use the maximum

[2]Trino would typically run out of memory while SparkSQL defaults to a more expensive disk-based sort-merge join.

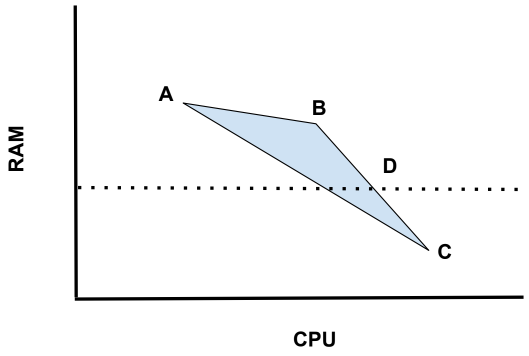

Figure 8: Illustration of the best cluster configurations achieved by heterogeneous clusters (D) vs homogeneous clusters (B). D has more CPU than B at the same cost per hour.

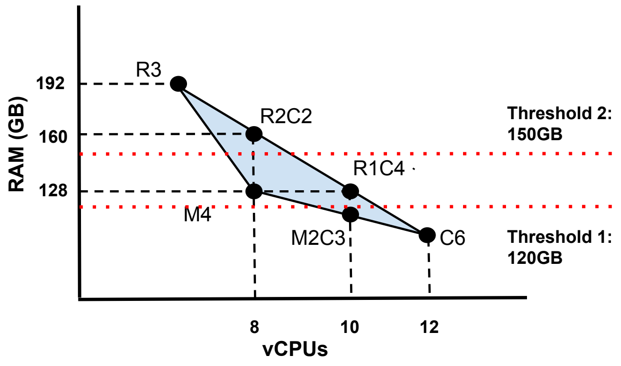

Figure 10: Illustration of the polytope depicted in Figure 8 for r6id.2xlarge, m6id.2xlarge and c6gd.2xlarge instance types with a total cost of around $1.8 an hour on demand. Valid cluster configurations are shown by solid dots.

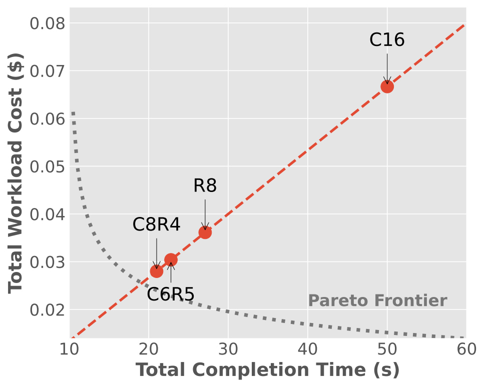

Figure 9: Speedups achieved by using heterogeneous clusters along the iso-cost curve with a unit cost of $4.8/hour. C8R4 denotes cluster configuration with eight c6gd.2xlarge and four r6id.2xlarge.

number of vCPUs. Since the query is bottlenecked by one resource (vCPU), heterogeneous clusters will not help.

- If the polytope contains configurations with enough memory to perform an in-memory join, then use the configuration that has exactly this much memory and as many vCPUs as possible, since additional memory does not contribute to higher performance. Heterogeneous clusters are helpful in this case.

Figure 8 illustrates why heterogeneous clusters are useful in the latter case. We have denoted the polytope in blue. As we have shown before in Figure 5, homogeneous clusters only hit the vertices. However, the most efficient total resource combination occurs on the line that connects vertices B and C, which can be approached more accurately with a heterogeneous cluster. To construct cluster configurations that connect B and C, one simply has to combine the instance types that would have made up the homogeneous clusters at B and C and mix the two in different ratios.

4 EXPERIMENTS

We take the query shown in Listing 1 and explore whether using heterogeneous clusters can lead to real speedups in the pipelined probe phase. We consider heterogeneous clusters consisting of only two instance types, r6id.2xlarge and c6gd.2xlarge.

In this study, we do not consider different sizes of each instance (i.e. 4xlarge, 8xlarge, etc.), as distributed database offerings typically rely on a fixed instance size. The 2xlarge setting (8 vCPUs) appears to be what Snowflake uses in practice [2] and what Databricks used for the TPC-DS benchmark [1].

The hourly on-demand pricing for the r6id.2xlarge instance is approximately double that of the c6gd.2xlarge instance. The two have the same number of vCPUs while the former has 4x the RAM per instance, which makes the former more cost-efficient for RAM and the latter more cost-efficient for vCPUs.

We consider a cost budget of using eight r6id.2xlarge instances or sixteen c6gd.2xlarge instances. We could trade off one r6id.2xlarge instance for two c6gd.2xlarge instances in our cluster while maintaining the same cost per hour, which in terms of total resources amounts to moving on the line between points B and C in Figure 8.

Figure 9 shows the concrete performance tradeoffs made as we explore heterogeneous cluster configurations that lie on that line. We start from eight r6id.2xlarge instances (R8)—offering the most RAM but the fewest CPUs—which allows us to do an in-memory join with a total completion time of 27s.

As we use more c6gd.2xlarge instances in our cluster, we move to configurations such as C8R4 (eight c6gd.2xlarge and four r6id.2xlarge instances) and C6R5. These configurations have less RAM but still enough to execute the hash join purely in memory. However, these configurations have more CPUs, leading to a much faster total runtime of 20s and 21s, amounting to a 35% speedup over the R8 configuration.

As we move purely to c6gd.2xlarge instances we have much less RAM and cannot execute the join in memory, which forces the query engine to use disk-spilling and severely degrades performance, leading to a runtime of 50s, more than two times slower than the C8R4 configuration.

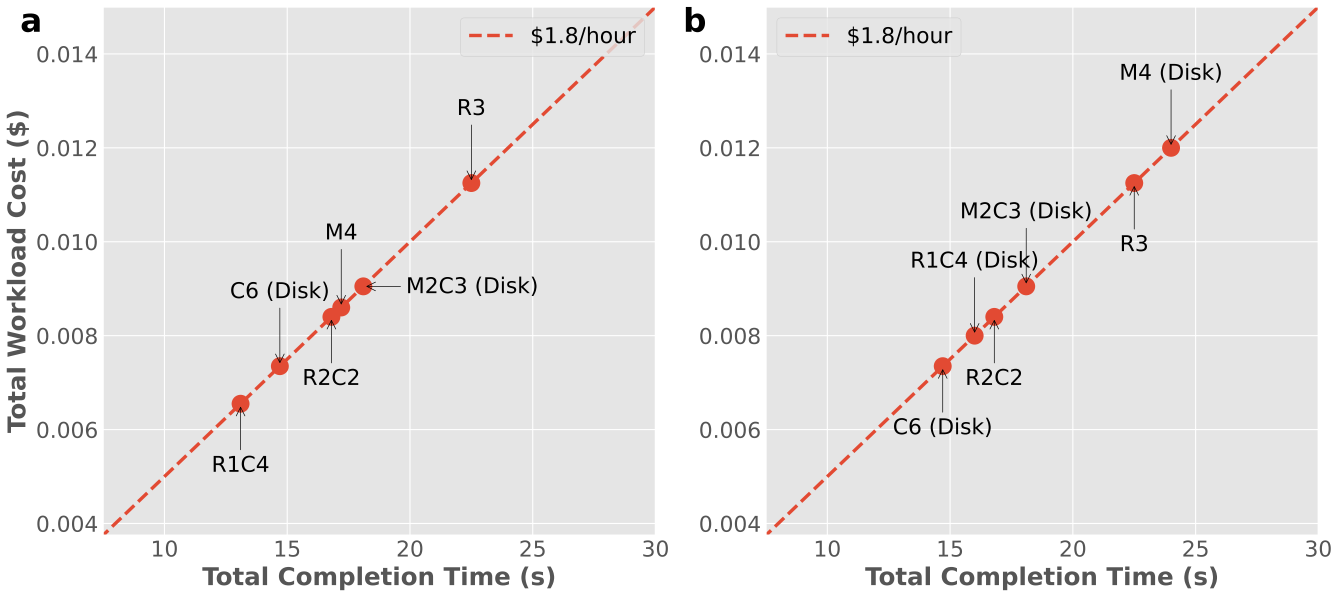

Figure 11: Runtimes of cluster configurations in Figure 10 assuming a disk-spilling threshold of a) 120GB and b) 150GB. The best performing cluster configuration is on the lower left.

The evaluation illustrated in Figure 9 uses the framework laid out in Section 2, where we move along an iso-cost curve. We see that the heterogeneous cluster configurations bring us closer to the hypothetical Pareto optimal frontier to the lower left. Since we are only exploring a very limited subspace of cluster configurations on this iso-cost curve, we cannot definitively state that any of our sampled configurations are "Pareto optimal". However, we see that the heterogeneous configurations C8R4 and C6R5 perform strictly better than the homogeneous configurations R8 and C16.

4.1 A More Complex Example

We now consider a more complex example with three different instance types, r6id.2xlarge, c6gd.2xlarge and m6id.2xlarge. For about $1.8/hour, we can have a cluster of three r6id.2xlarge instances, or four m6id.2xlarge instances or six c6gd.2xlarge instances. We consider a more complicated join query involving three tables based on TPC-H 3, shown in Listing 2:

Listing 2: Multi-stage join query

```
SELECT l_orderkey, o_orderdate, o_shippriority
        sum(l_extendedprice * (1 - l_discount))
            as revenue,
FROM customer, orders, lineitem
where c_custkey = o_custkey
      and l_orderkey = o_orderkey
      and o_orderdate < date '1995-03-15'
      and l_shipdate > date '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
```

In Figure 10 we show what the constraint polytope shown abstractly in Figure 8 looks like for these three instance types. Besides the homogenous cluster configurations, we also consider three heterogeneous cluster configurations: R2C2 (two r6id.2xlarge instances and two c6gd.2xlarge instances), R1C4 and M2C3.

We consider two different RAM thresholds to force disk-spilling of the joins at 120GB and 150GB. The performance of each cluster configuration under these two settings is shown in Figure 11 along the iso-cost curve with slope $1.8/hour. We mark what cluster configurations are forced to disk-spill for the join in each case.

We briefly discuss how to interpret the results. For Figure 11a where the threshold is 120GB, we can see in Figure 10 that R1C4, the cluster configuration that performs the best, is directly above the cutoff, meaning it has the most vCPUs out of all cluster configurations that do not have to disk spill.

However, for a threshold of 150GB, the analogous cluster configuration R2C2 is outperformed by both R1C4 and C6, which use disk spilling. The query in Listing 2 is impacted by disk spilling less than the query in Listing 1, and the extra vCPUs overcome the benefits of avoiding disk spills. This example suggests that for some queries, there is a need for more accurate performance modeling than our simple heuristic to obtain the most CPUs without disk spilling.

Another observation is that cluster configurations involving the m6id.2xlarge (M) instance type are not among the top-performing configurations in either case—see Figure 10. Note that for any valid cluster configuration that includes an M instance, we can find a configuration consisting of only R and C instances by finding its horizontal or vertical projection onto the line connecting R3 and C6. Examples are shown for the cluster configuration M4 and M2C3, which suggests that the m6id.2xlarge instance type is not as cost effective as the r6id.2xlarge and c6gd.2xlarge instance types in terms of RAM or CPU.

5 RELATED WORK

Optimal VM Selection. Previous research has focused on optimizing VM cluster selection for analytical workloads to balance cost reduction with performance requirements [4, 5, 9, 13, 21]. These systems vary in terms of the approach adopted to determine the optimal configuration (e.g. using Bayesian Optimization [4, 9]), the execution of this approach (i.e., online vs. offline modeling), and the type of data integrated into the model (e.g. coarse-grained instance-level information, low-level metrics, etc.). These frameworks, however, ultimately consider only clusters of one specific instance type.

Heterogeneous Clusters. FineQuery and Selecta also consider heterogeneous hardware for SQL queries. However, their focus is on different storage devices or CPU-GPU, not different VM instance types [11, 20]. Previous works, like KAIROS and OptimusCloud, have also explored optimizing heterogeneous cloud resources for non-SQL workloads like machine learning inference and key-value stores [14, 15]. A related body of work has also explored using heterogeneous clusters to exploit model parallelism in deep learning, where different layers in a deep learning model have different resource requirements [10, 17]. Our work draws inspiration from the findings in these systems.

6 CONCLUSION AND VISION

In this work, we show that using different instance types in the same cluster can speed up query execution. We introduce iso-cost curves, which offer a simple method to reason about the optimality of a cluster configuration when optimizing query runtime with constraints on overall resources. We demonstrate that the key benefit of cluster heterogeneity is expanding the space of total resources available, leading to concrete performance benefits for the probe phase of a pipelined join. We note three promising directions to pursue for future work:

Different mapping strategies. While this paper focused on optimizing different cluster configurations, pipelined query engines support different ways to execute a query given a fixed set of instances. [3] Being able to quickly decide on the strategy given a query, a query engine, and a cluster configuration is far from trivial. Techniques employed in prior work such as Selecta and FineQuery could be applied to this problem [11, 20].

Model-guided cluster configuration optimization. This paper presented a heuristic for a given query pattern based on performance benchmarking. It is impractical for practitioners to benchmark every query they encounter. A solution could be extensive offline benchmarking of different query execution stages coupled with a fast online cost-function-guided search for a new query. We believe this approach, pioneered by FlexFlow for deep learning problems, holds great promise in making heterogeneous clusters practical for real query workloads [10].

[3]Assume we have a single join stage following a scan stage and two worker machines with two executor slots per machine. The engine could assign one executor slot per machine to each stage, the default strategy used in experiments in this paper, or assign one machine completely to each stage.

Virtual clusters. In cases where a cluster manager like Kubernetes is used, the question then becomes how to properly configure the resource requirements of different pods used in a data processing job. The per unit resource cost of a pod is much harder to reason about than VM on-demand pricing, as it may involve business costs of preempting other jobs. Heterogeneity is still interesting, as e.g. using a mix of high and low-memory pods for a job might be easier to schedule than uniformly mid-memory pods.

Vision. With the rise of serverless computing, we envision a future where each query is executed on its own dedicated ephemeral cluster. This scenario is not only plausible, but also appealing due to its performance and security isolation properties. In such a future, where clusters of varying resource configurations can be quickly ordered on demand, the query engine must be able to express a preference on what this cluster should look like for each query. Being able to leverage heterogeneous clusters is necessary to take full advantage of the flexibility public clouds provide, and is a key step towards this goal.

REFERENCES

[1] [n. d.]. Databricks TPC-DS. https://www.tpc.org/results/fdr/tpcds/databricks~tpcds~100000~databricks_sql_8.3~fdr~2021-11-02~v01.pdf. Accessed on May 29, 2023.

[2] [n. d.]. Snowflake Debugging Info. https://stackoverflow.com/questions/58973007/what-are-the-specifications-of-a-snowflake-server. Accessed on May 29, 2023.

[3] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*. 419–434.

[4] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 469–482.

[5] Muhammad Bilal, Marco Canini, and Rodrigo Rodrigues. 2020. Finding the right cloud configuration for analytics clusters. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 208–222.

[6] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL Query Optimizer: A modern optimizer for real-time analytics in a distributed database. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1401–1412.

[7] Clickhouse. 2023. Clickhouse. https://github.com/ClickHouse/ClickHouse. Accessed on July 10, 2023.

[8] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. 2016. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*. 215–226.

[9] Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. 2018. Arrow: Low-level augmented bayesian optimization for finding the best cloud vm. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 660–670.

[10] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. *Proceedings of Machine Learning and Systems* 1 (2019), 1–13.

[11] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2018. Selecta: Heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 759–773.

[12] Viktor Leis, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2014. Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 743–754.

[13] Viktor Leis and Maximilian Kuschewski. 2021. Towards cost-optimal query processing in the cloud. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1606–1612.

[14] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. Building Heterogeneous Cloud System for Machine Learning Inference. *arXiv preprint arXiv:2210.05889* (2022).

[15] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. {OPTIMUSCLOUD}:

Heterogeneous configuration optimization for distributed databases in the cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 189–203.

[16] marsupialtail. 2023. Quokka. https://github.com/marsupialtail/quokka. Accessed on July 10, 2023.

[17] Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jaesik Choi, Sam H Noh, and Young-ri Choi. 2020. {HetPipe}: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 307–321.

[18] Mark Raasveldt and Hannes Mühleisen. 2019. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*. 1981–1984.

[19] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, et al. 2019. Presto: SQL on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1802–1813.

[20] Dalin Wang, Feng Zhang, Weitao Wan, Hourun Li, and Xiaoyong Du. 2021. FineQuery: Fine-grained query processing on CPU-GPU integrated architectures. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 355–365.

[21] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. 2017. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*. 452–465.