Identifying Iso-Cost Sweet Spot Configurations for Cloud OLAP Queries

Weitao Wan[1][0009−0009−8124−7059], Huanchen Zhang[1,2][0009−0001−4821−1558], and Mingyu Gao[1,2][0000−0001−8433−7281]✉

[1] Tsinghua University, Beijing, China
[2] Shanghai Qi Zhi Institute, Shanghai, China

wwt22@mails.tsinghua.edu.cn, huanchen@tsinghua.edu.cn, gaomy@tsinghua.edu.cn

Abstract. Cloud-native OLAP systems run queries on virtual machine clusters. While pricing scales linearly with the virtual core count, query performance does *not*. Therefore it is critical to identify the best performing sweet spot among many iso-cost configurations, e.g., 4 nodes of 32 cores each vs. 64 nodes of 2 cores each. In this work, we aim to comprehensively understand what are the key factors of the underlying hardware and the executing query that determine the performance trade-off, and use them to help identify the iso-cost sweet spot configuration. We conduct extensive experiments using Presto, running dozens of queries across five AWS EC2 instance families. We empirically find that mid-range instances usually outperform both extremes. But the exact profile is highly dependent on instance families, determined by the hardware chiplet organization as well as the available network bandwidth. Furthermore, certain query-level characteristics, including data skewness, plan complexity, and shuffle intensity, can shift the sweet spots. Finally, we distill these findings into insightful lessons for future query engine design, including chiplet-aware parallelism planning, cluster sizing as a skew mitigation lever, and structural isolation of straggler propagation.

Keywords: cloud computing · OLAP · resource configuration · performance analysis

1 Introduction

Cloud-native online analytical query processing (OLAP) systems have recently emerged as a cost-efficient way to host database applications [5,8,17]. The query engine can leverage parallel computing resources in the form of virtual machine (VM) clusters, while data reside in object stores that are disaggregated from the computing [14,22]. This approach offers many unique advantages including high elasticity, high availability, on-demand cost, etc. when compared to traditional database infrastructures.

On current cloud platforms such as AWS EC2, the per-hour VM pricing scales linearly with the virtual CPU (vCPU) count. For example, in the AWS `r7i` family, $4 \times$ `r7i.8xlarge` (32 vCPUs each) costs the same per hour as $64 \times$ `r7i.large`

(2 vCPUs each); both configurations hold 128 total vCPUs and identical total memory. We call such VM clusters *iso-cost configurations*. As a result, spending the same budget can purchase VM clusters of very different configurations, among which some run substantially faster than the others for specific OLAP queries. Choosing the wrong configurations would waste substantial budget and also risk missing strict user-facing latency targets [27].

It thus remains an important question *how to choose the most appropriate cluster for a given query among the iso-cost configurations*. This decision is non-trivial because no single choice fits all. The performance profile is highly dependent on specific query behaviors as well as hardware types; any fixed policy would likely leave performance on the table for a certain portion of thousands of daily queries. Existing approaches either predict good configurations without explaining why they work [1, 7, 12], or characterize hardware effects in isolation without connecting them to query-level configuration choices [10, 15]. What is missing is a comprehensive study about *which underlying mechanisms of the query and the hardware* actually govern the performance trade-off.

In this paper, we conduct such a study using a state-of-the-art engine Presto [22], running TPC-H [23] and TPC-DS [24] across five AWS EC2 instance families. We draw several insightful findings (Section 3). First, we observe a general pattern: for most queries, mid-range instances usually outperform both extremes. We call this optimum the *iso-cost sweet spot*. Second, the exact location of the sweet spot is dependent on hardware instance families. This is mainly due to *different microarchitecture designs, particularly the chiplet hierarchy within each processor*. On AMD instance families like `r7a` and `r6a`, a sharp per-core bandwidth drop at the chiplet boundary pushes the sweet spot to smaller instances, while Intel's tiled design (e.g., `r7i`) produces an architecturally milder boundary, and the iso-cost sweet spot profile resembles monolithic processors like `r6i`. Besides, network-enhanced instances like `r6in` differ from the general pattern: large instances are the fastest, while small instances exhibit elevated instability. Third, when fixing the instance family and executing different queries, some query-level characteristics may also shift the sweet spots: *data skewness and plan complexity* favor larger instances, while *shuffle intensity* favors smaller ones.

We distill the above findings into lessons for future query engine design guidelines and cluster-sizing tools (Section 4). We argue that, the chiplet topology should be considered to inform parallelism decisions beyond the raw vCPU count; cluster sizing should be treated as a first-class skew mitigation lever; and engine-level design changes can reduce the impact of straggler cascades. The indicators identified in our findings can further serve as extra decisive features for machine-learning-based configuration advisor methods.

2 Experimental Methodology

Our experiments use Presto [22], a representative SQL query engine, with the Prestissimo/Velox C++ backend [19], on AWS EC2 cloud. We use five memory-optimized instance families, covering two AMD chiplet architectures (AMD Genoa

Table 1: AWS EC2 `r6in` iso-cost configurations at 128 total vCPUs.

Instance Type	vCPUs	Memory	Baseline BW	Burst BW	Nodes
`r6in.large`	2	16 GiB	3.125 Gbps	Up to 25 Gbps	64
`r6in.xlarge`	4	32 GiB	6.25 Gbps	Up to 30 Gbps	32
`r6in.2xlarge`	8	64 GiB	12.5 Gbps	Up to 40 Gbps	16
`r6in.4xlarge`	16	128 GiB	25 Gbps	Up to 50 Gbps	8
`r6in.8xlarge`	32	256 GiB	50 Gbps	50 Gbps	4

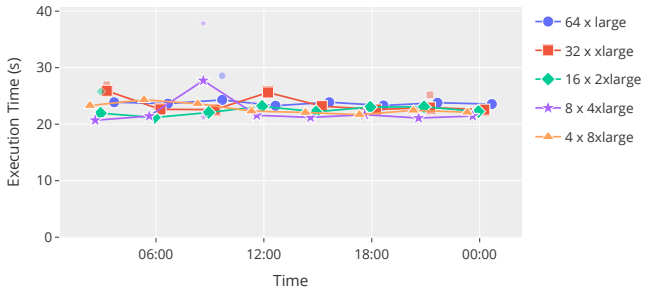


Fig. 1: Execution time of TPC-DS Q23-pruned on `r7i` with multiple executions over 24 hours. Each point is one measured run, grouped by configurations.

`r7a`, AMD Milan `r6a`), an Intel tiled architecture (Sapphire Rapids `r7i`), an Intel monolithic architecture (Ice Lake `r6i`), and a network-enhanced monolithic variant (`r6in`). Within each family we build five iso-cost clusters at 128 total vCPUs, from 64 nodes of `large` to 4 nodes of `8xlarge`. Table 1 shows the `r6in` family as a representative example; other families have similar specifications at roughly half the absolute network bandwidth.

We run TPC-H [23] and TPC-DS [24], at the scale factor 1000 (∼295 GB in the Parquet file format) with data from Amazon S3. We restrict our analysis to queries whose physical plans remain identical across all the five iso-cost configurations in each family. Depending on resources such as per-node memory, Presto's optimizer may choose different join strategies (e.g., broadcast vs. partitioned) across configurations. By keeping the plan fixed, we focus on the hardware and query factors that determine performance. Comparing different configurations, each with its own best plan, is outside our scope. After filtering, 12 of 22 TPC-H queries and 29 of 60 TPC-DS queries are kept.

We perform parameter tuning on the Presto engine to avoid several performance pitfalls when it runs on our used instances. We find three engine parameters need adjustments. First, we reduce the output batch size of inter-worker data exchange from 10 MB to 2 MB. Smaller batches distribute received data across more threads and reduce per-batch initialization overhead. Second, we reduce the maximum data partition (split) size from 256 MB to 32 MB. The

finer granularity reduces data-dependent load imbalance across workers. Third, we enable task-level split scheduling to replace the default node-level accounting. Under the default policy, a large scan stage occupies the node's global split counter and starves other stages of the same query that share the node; task-level scheduling eliminates this interference.

We verify the tuned baseline with TPC-DS Q23-pruned (Appendix A), a hand-simplified probe query with balanced data, low shuffle volume, and simple plan. These nice characteristics should produce nearly identical performance across our iso-cost configurations. To account for temporal variation, we launch fresh clusters every few hours over a 24-hour window, perform warm-up runs, and record at least five measured runs per session. Figure 1 shows the result on `r7i`: each point is one measured run, grouped by configurations. Medians across the five configurations fall within ∼10–15% of each other, and no configuration is systematically fastest or slowest, confirming that the tuned engine does not inherently favor any particular configuration. We observe a similar trend on `r7a` and `r6i`. Nevertheless, we still observe several straggler spikes from the experiments, where an individual run occasionally has much longer time than the median. We suspect this is due to contention from co-located tenants or unfavorable task placement on physical servers. When a single worker degrades, the penalty does not stay local: Presto's shared `PartitionedOutput` buffer backpressures all producers uniformly, so the degradation cascades into a cluster-wide stall. Shuffle-heavy queries exhibit a second source of instability, network burst credit depletion, analyzed later in Section 3.3. As a result, a single run may produce misleading results. All reported numbers throughout the paper are medians across sessions with interquartile range (IQR) error bars.

3 Observation and Analysis of Iso-Cost Sweet Spot

In this section, we describe the key observations regarding the sweet-spot configurations across the 41 queries running on the well-tuned query engine from Section 2, using multiple hardware instance families at iso cost. We first summarize a general observation in Section 3.1. Then we compare different instance families (Section 3.2), as well as different queries on the same instance family (Section 3.3), and analyze their deviations from the general pattern.

3.1 General Pattern

Figure 2 shows the normalized execution time for the 12 TPC-H queries on `r7i`; the 29 TPC-DS queries exhibit the same pattern with similar magnitudes, and are omitted due to space limit. Every query exhibits measurable configuration sensitivity. Even Q04, the least sensitive query here, runs 11% slower on its worst configuration than on its best. For 10 of the 12 TPC-H queries, the optimum sits at $16 \times$ `2xlarge` or $32 \times$ `xlarge`, while the two extremes are consistently the slowest. Therefore the iso-cost sweet spot is clear: *both the small-node and large-node extremes are inferior to the middle, and the advantage of the mid-range holds across the full query set rather than appearing on one or two queries only.*

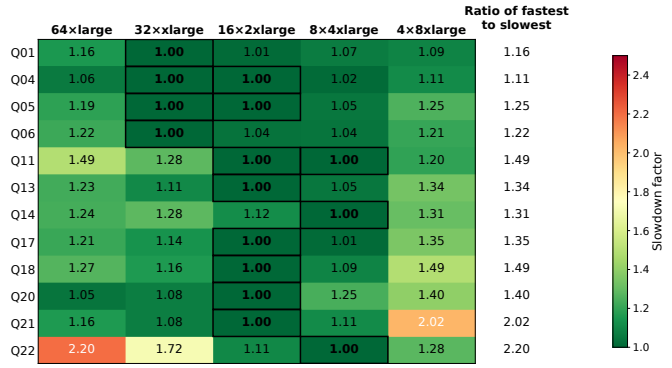


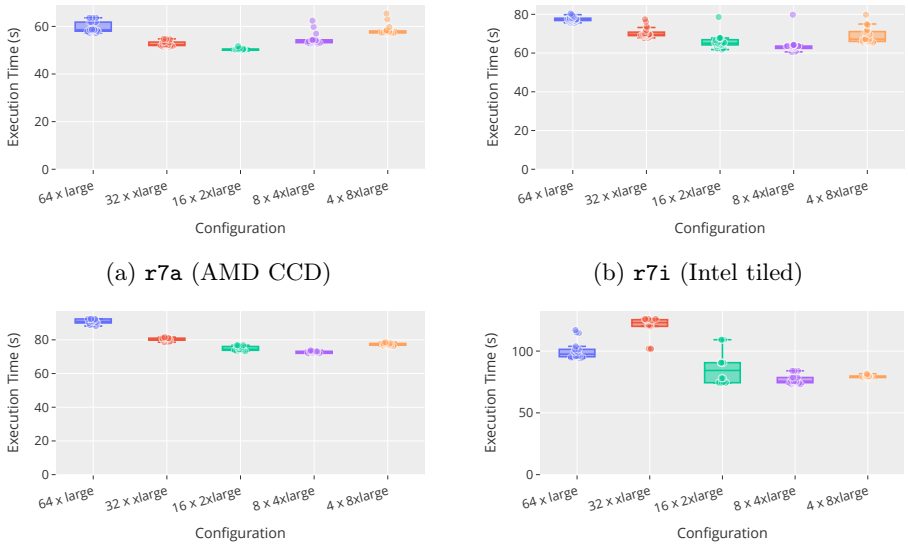
Fig. 2: Normalized execution time of TPC-H queries on `r7i`. Each row is a query. The fastest configuration for each query is marked as 1.00.

3.2 Differences among Instance Families

Although the sweet spot pattern appears on every family we tested, its shape and the sweet-spot location differ widely; i.e., *the sweet-spot profile is family-specific.* In Figure 3, we vary the instance family used, while keeping the same query of Q23-full (a complex multi-stage TPC-DS query). On AMD `r7a`, the optimum is $16 \times$ `2xlarge`, and larger instances become worse. On Intel `r7i` and `r6i`, the optimum shifts to $8 \times$ `4xlarge` with fewer but larger instances. Network-enhanced `r6in` breaks this pattern entirely.

We attribute these variations among instance families to their microarchitecture differences, particularly the *chiplet boundaries*. Table 2 summarizes the architecture for each family. Specifically, on AMD `r7a`, the individual cores within a single physical socket are distributed into multiple chiplets, a.k.a., core complex dies (CCDs) in AMD's terminology. Each CCD has a fixed link to the off-chip memory controllers. The CCD boundary of `r7a` falls at 8 physical cores, i.e., `2xlarge`. Doubling from `xlarge` to `2xlarge` activates a full CCD, saturates the link to memory, and drops per-vCPU bandwidth by nearly half, as shown in Figure 4(a). In contrast, on Intel `r7i` in Figure 4(b), a milder drop occurs at `8xlarge`, where the instance spans two or more tiles and cross-tile accesses incur additional latency. The two monolithic families not shown (`r6i` and `r6in`) have per-vCPU bandwidth roughly constant across sizes.

The MLC measurement establishes what the hardware delivers in isolation. To connect it to query-level behaviors, we further look into actual CPU time of query execution. If the per-vCPU throughput were constant across iso-cost configurations, the total CPU time across workers would be equal, because the configurations share the same number of total vCPUs and process the same data. Any deviations point to per-vCPU efficiency loss. Figure 5 applies this test to TPC-DS Q28, a shuffle-intensive query, on four families. Both AMD families show a staircase: on `r7a` (hyperthreading off), the CPU time rises from `2xlarge`, coinciding with the CCD boundary; on `r6a` (hyperthreading on), the



(a) `r7a` (AMD CCD) (b) `r7i` (Intel tiled)

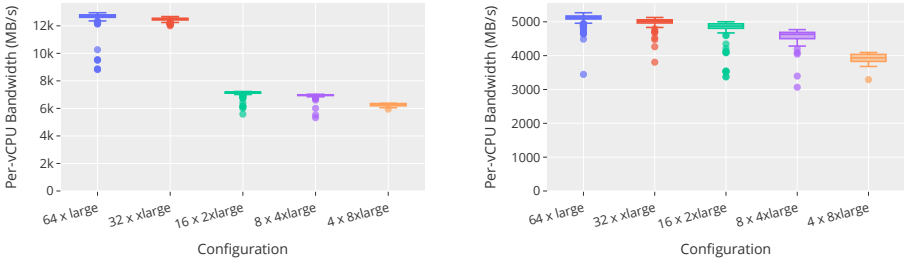
(c) `r6i` (Intel monolithic) (d) `r6in` (network-enhanced)

Fig. 3: Execution time of TPC-DS Q23-full on four instance families.

rise starts at `4xlarge`, shifted by one size because hyperthreading doubles the vCPUs per CCD. On Intel tiled `r7i` and monolithic `r6i`, the CPU time stays flat. The difference between AMD and Intel chiplet designs is consistent with their memory-controller placement: AMD routes all memory traffic through a shared I/O die, so crossing a CCD boundary increases contention sharply, whereas Intel Sapphire Rapids integrates memory controllers into each tile, producing a more homogeneous bandwidth distribution [10]. Intel's tile boundary, although visible in MLC throughput as a mild per-vCPU drop (Figure 4(b)), does not translate into significant per-vCPU efficiency loss at the query level. The staircase is also query-dependent on AMD: TPC-DS Q09, another shuffle-intensive query, shows flat CPU time on both `r7a` and `r6a`, just as it does on `r7i`. The difference likely

Table 2: Microarchitecture and chiplet boundary of the instance families. The L3 cache capacity per core is reported at `8xlarge`.

Family	Architecture	Hyperthreading	Chiplet Boundary	L3/core
`r7a`	AMD CCD (8c)	Off	`2xlarge` (CCD sat.)	4.0 MB
`r6a`	AMD CCD (8c)	On	`4xlarge` (CCD sat.)	4.0 MB
`r7i`	Intel tiled (12c)	On	`8xlarge` (cross-tile)	6.6 MB
`r6i`	Intel monolithic	On	-	3.4 MB
`r6in`	Intel monolithic	On	-	3.4 MB



(a) `r7a` (AMD CCD) (b) `r7i` (Intel tiled)

Fig. 4: Per-vCPU memory bandwidth on two instance families. Measured using Intel Memory Latency Checker (MLC) [13], with a 2:1 read-to-write ratio, a 1 GB buffer that bypasses all caches.

reflects operator mix, as Q28's operators are more memory-bandwidth sensitive than Q09's. We investigate more query-specific behaviors in Section 3.3.

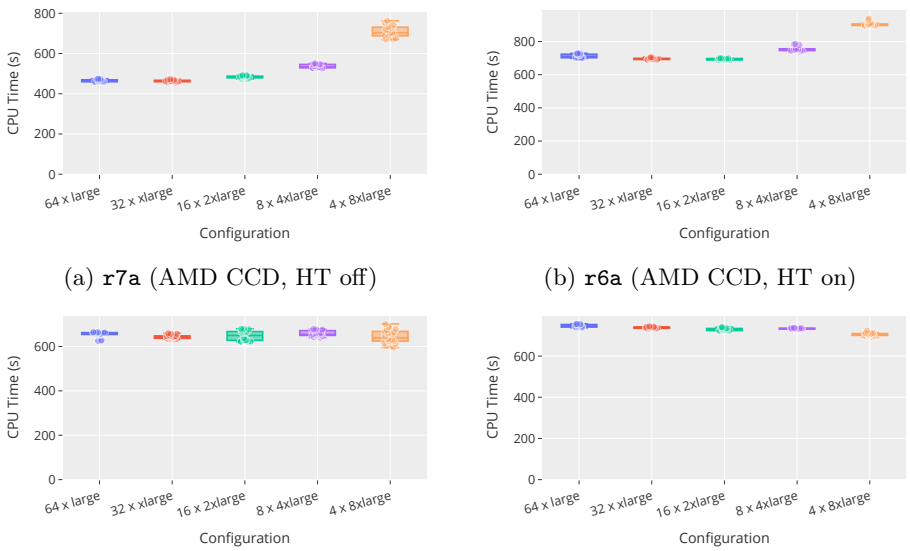
The families above all use standard network bandwidth. Network-enhanced `r6in` shares `r6i`'s monolithic Ice Lake die, but provides roughly twice the baseline bandwidth at every size (Table 1). Figure 3(d) shows that, the performance of Q23-full on `r6in` improves monotonically (with abnormal `xlarge`) toward larger instances, breaking the general sweet-spot pattern. Figure 6 confirms the trend on two additional queries. On all the three queries, the two smallest configurations are consistently the slowest; on Q23-full, `xlarge` burns ∼30% more aggregate CPU time than `4xlarge` despite the monolithic die. This instability does not appear on `r6i`, which shares the same Ice Lake die, pointing to the network-enhanced Nitro configuration rather than the CPU as the likely cause.

3.3 Differences among Queries

Next we examine why some individual queries deviate from the common pattern of each instance family. We discover two key factors: *data skewness and plan complexity* may push the optimum of a query toward larger instances, while *shuffle intensity* can push it toward smaller instances.

Small-Instance Penalties: Data Skewness and Plan Complexity. With a distributed query engine spanning multiple nodes like ours, the data partitions (splits) that are initially of comparable byte sizes may develop into vastly different sizes at downstream. For example, a date-range filter can eliminate most rows from some partitions, but almost none from others; a hash repartition may concentrate data onto a few receivers if the data distribution is highly skewed.

Such data skew could lead to significantly different performance impacts on different configurations. With many small workers each receiving few partitions, a worker that draws a disproportionately heavy one becomes a straggler. With



(a) `r7a` (AMD CCD, HT off)

(b) `r6a` (AMD CCD, HT on)

(c) `r7i` (Intel tiled)

(d) `r6i` (Intel monolithic)

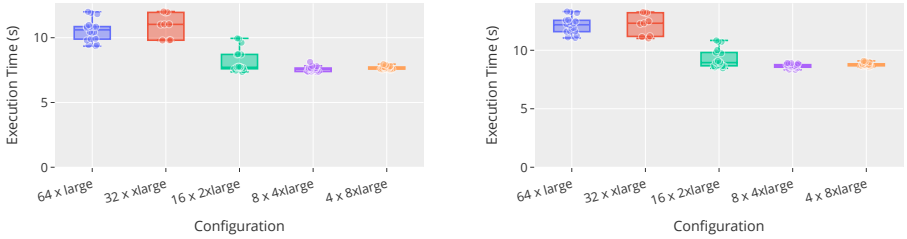
Fig. 5: CPU time of TPC-DS Q28 on four instance families.

fewer large workers each receiving many partitions, statistical averaging could potentially flatten out the imbalance.

Figure 7(a) shows TPC-DS Q22 on `r7i`. Q22 scans a fact table filtered to a 12-month date range. The data generator produces rows in chronological order, so row groups overlapping the target period retain nearly all their rows, while others produce almost nothing, yielding per-task output variation of $\sim$80$\times$ on `xlarge`. Execution time improves monotonically from $64 \times$ `large` to $4 \times$ `8xlarge`, a 2.5$\times$ speedup at identical cost. The run-to-run variance also shrinks at the same time, because each worker now averages over enough splits to stabilize against unlucky assignments. Figure 7(b) shows the same query on `r7a`. Performance improves through `2xlarge` but plateaus beyond it, and `8xlarge` is marginally slower than `4xlarge`. The plateau coincides with the CCD boundary, where per-vCPU memory bandwidth drops and offsets the skew absorption gain. On `r7i`, where the bandwidth drop is milder, Q22 improves all the way to `8xlarge`.

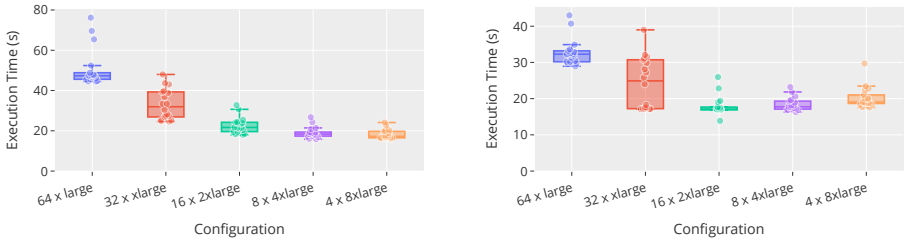
Q22's skew is statistical: random split assignment determines per-worker load, so more splits per worker always helps. On the other hand, TPC-DS Q67 has a different structure: a Window operator partitions by the product category, of which TPC-DS defines only 10 categories. The improvement saturates once the worker count approaches the key-space size, and the optimum on `r7i` falls at $8 \times$ `4xlarge` rather than at the largest instance.

Besides data skewness, query plan complexity produces another independent small-instance penalty. Presto schedules each pipeline stage as a set of drivers on a per-node thread pool. When the plan has many concurrent stages, the total



(a) Q28 (b) Q09

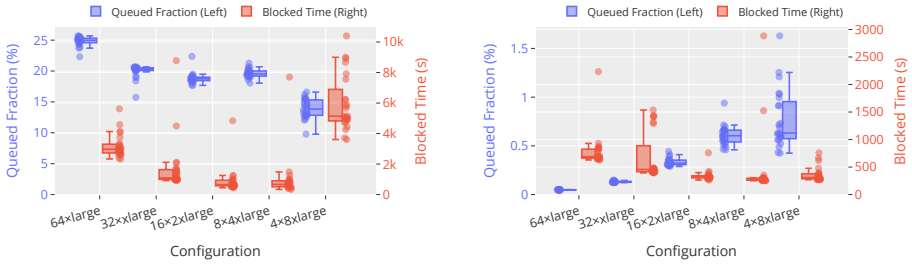
Fig. 6: Execution time of TPC-DS Q28 and Q09 on `r6in`, both of which do not follow the sweet-spot pattern.



(a) Q22 on `r7i` (b) Q22 on `r7a`

Fig. 7: Execution time of TPC-DS Q22 on `r7i` (a) and `r7a` (b).

driver count exceeds the available threads and some drivers must be queued for execution. The penalty is severe on small instances: on $64 \times$ `large`, each node has only 2 vCPUs, and the thread pool is tiny. As a concrete example, Q23-full and Q23-pruned share the same data and core query logic, but differ in the plan structure: Q23-full has many concurrent pipeline stages, while Q23-pruned has few. While Q23-pruned achieves near iso-performance across configurations (only ∼10–15% difference, Figure 1), Q23-full on `r7i` is roughly 30% slower on $64 \times$ `large` than on $8 \times$ `4xlarge`. Figure 8 shows the diagnostic: on Q23-full, critical-path stages spend ∼25% of their non-blocked time queued for a CPU thread on $64 \times$ `large`, falling to ∼15% on $4 \times$ `8xlarge`; on Q23-pruned the corresponding waiting is below 1% at all sizes. The queuing also cascades: stages that progress slowly consume exchange buffer data at a reduced rate, eventually blocking upstream stages due to full buffers. The aggregate CPU time confirms this diagnosis. On Q23-full, the highest CPU time is on the largest instances (per-vCPU memory overhead from Section 3.2), while the longest wall-clock time is on the smallest. The two metrics point in opposite directions, confirming that queuing, not extra computation, is what penalizes small-instance configurations on complex plans.



(a) Q23-full, critical-path stages (b) Q23-pruned, corresponding stages

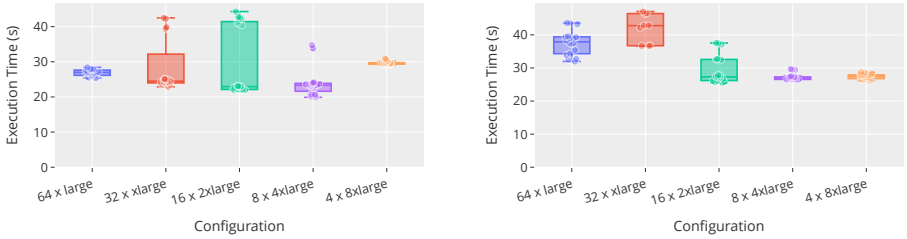
Fig. 8: Queued time fraction (left axis) and blocked time (right axis) of critical-path pipeline stages in TPC-DS Q23-full (a) and Q23-pruned (b) on `r7i`.

Large-Instance Penalties: Shuffle and Network Tier. Shuffle-heavy queries push in the opposite direction. As shown in Table 1, iso-cost configurations share the same aggregate *baseline* network bandwidth, but the *burst* bandwidth per vCPU decreases with the instance size. Clusters of many small instances can temporarily exceed their aggregate baseline during short shuffles; clusters of a few large instances cannot. Figure 9(a) shows TPC-DS Q80, a shuffle-intensive query, on `r7i`. Table 1 shows that `8xlarge` is the only size whose baseline bandwidth equals its burst one and hence has no burst headroom; for Q80's shuffle volume, this fixed bandwidth is insufficient, making $4 \times$ `8xlarge` consistently the slowest when comparing the best runs of each configuration. Smaller configurations can burst above their baselines, but the burst capacity is governed by a credit mechanism: credits accumulate during low-traffic periods and deplete under sustained shuffle. Figure 10 shows the consequence over 24 hours, plotting per-run execution time in (a) alongside aggregate cluster network throughput in (b). The two plots are complementary: latency spikes coincide with throughput drops. The effect is most visible on `xlarge` and `2xlarge`, whose burst headroom is large enough that credit depletion causes pronounced swings. TPC-DS Q28, Q80, and TPC-H Q17, along with seven other queries across both benchmarks, exhibit this pattern on standard-bandwidth families.

Large instances are slow on shuffle-heavy queries because their aggregate baseline bandwidth is too low, not because they are large. On network-enhanced `r6in`, which is of the same size but provides roughly twice the baseline network bandwidth (e.g., reaching 50 Gbps at `8xlarge`), the bottleneck disappears. Figure 9(b) shows Q80 on `r6in`: the bimodality is absent and performance improves monotonically with instance size.

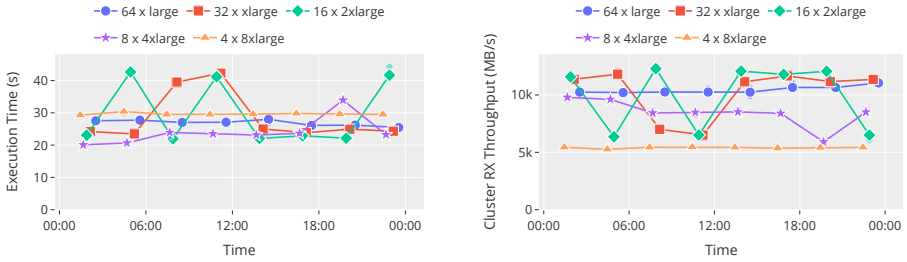
4 Lessons and Discussions

From the observations and analyses in the previous section, we draw several insightful lessons that can potentially guide future cloud OLAP system designs.



(a) `r7i` (standard) (b) `r6in` (network-enhanced)

Fig. 9: Execution time of TPC-DS Q80 on standard `r7i` (a) and network-enhanced `r6in` (b).



(a) Q80 latency time series (b) Aggregate cluster network RX

Fig. 10: Latency time series (a) and aggregate network throughput (b) of TPC-DS Q80 on `r7i` over 24 hours.

Parallelism planners should be aware of hardware chiplet topology. On AMD families the chiplet boundary dominates the sweet-spot profile (Section 3.2), yet existing query engines set their parallelism based on vCPU count alone. The chiplet boundary is a fixed, readable property of the instance family. So the engine can easily inspect this information at startup, and choose worker count and per-worker parallelism accordingly, extending the single-machine observation of Fogli et al. [10] to cluster sizing.

Node instability should be an input to placement, not just a dashboard metric. Both burst credit depletion (Section 3.3) and memory-side contention straggler cascades (Section 2) are observable at runtime, yet existing engines log these signals without acting on them. Feeding them into an online, dynamic scheduler could avoid assigning shuffle-heavy stages to credit-depleted nodes and memory-sensitive operators to degraded workers.

Cluster sizing is a first-class lever for skew mitigation. Prior skew mitigation approaches [2, 21] work inside the engine. Figure 7 has shown that simply reducing the worker count of Q22 from 64 to 4 yields a 2.5× speedup at iso cost with no engine-side adaptation. Consequently, we argue that skew

mitigation should not be confined to engine-internal algorithms; future research could also incorporate cluster-sizing knobs.

Straggler isolation should be structural, not statistical. The back-pressure cascade in Section 2 shows that shared exchange buffers associate cluster progress to the slowest producer. Speculative execution [2,20] detects slow tasks afterwards but does not prevent the buffer from stalling. A structural solution would decouple downstream consumers from any single upstream producer so that healthy producers continue while a degraded one is bypassed.

Machine-learning-based configuration advisors can use our newly discovered mechanisms as extra model features. Current advisors [1,7,12] treat configuration selection as a black box with vCPU count and price as the only features, limiting transfer across workloads and families. Our results supply four concrete features: (1) chiplet boundary location, (2) network bandwidth tier, (3) skewness profile, and (4) shuffle intensity. Among them, (1) and (2) are readable from the instance family, while (3) and (4) are computable from the query plan. We have not built such an advisor, but argue that these features, combined with certain machine learning methods, would give predictive tools a principled way to generalize beyond their current scopes.

Generality beyond AWS and Presto. Our current results come from one cloud platform and one query engine. We next discuss which mechanisms are likely to carry over and which are specific to AWS or Presto. The per-vCPU memory-bandwidth cliff is not specific to AWS, because it comes from the processor architecture: the same AMD EPYC and Intel Xeon chips are used by other cloud providers [11,18]. The VM performance instability is not AWS-specific, either, as prior studies report similar variability across providers [6,16]. For network behavior, size-dependent bandwidth limits and a network-enhanced tier are common across major clouds [9]. However, the burst-credit instability is specific to AWS's network credit mechanism. For engine behavior, skew can interact with cluster sizes in any partition-parallel engine, but the effect is weaker when the engine repartitions skew at runtime. Spark does so through adaptive query execution [3,4,26], whereas Presto does not perform comparable runtime repartitioning. The plan-complexity penalty and the straggler cascade, however, are more specific to Presto and may not appear in other engines.

5 Related Work

This work investigates the efficiency of various cloud instance configurations when running OLAP queries. Two categories of prior research are relevant to our study. The first is cloud configuration selection. Predictive tools [1, 7, 12] sample configurations and model the latency-cost surface but do not explain why one configuration outperforms another, limiting their generalization across workloads and families. Analytical frameworks [15, 25] characterize trade-off dimensions in closed-form models but do not connect them to query-level behaviors on a real engine. Neither approach has connected microarchitectural mechanisms to query-level outcomes. The second category is infrastructure characterization.

Fogli et al. [10] characterize chiplet effects on OLAP processing on a single physical machine; our analysis extends theirs to a cloud setting where the instance size is the tunable parameter. Data skewness mitigation via adaptive partitioning [21] or speculative execution [2,20] treats skewness as a query-level symptom; our observation that skewness interacts with granularity is complementary.

6 Conclusion

Given a fixed budget for a query, should a cloud-native OLAP system use fewer large VMs or many small ones? We study this question by running two benchmark suites on a Presto tenant across five families of AWS EC2 instances. We find that the answer depends on the hardware: mid-range instances usually outperform both extremes on standard-bandwidth instances, with AMD's chiplet boundary noticeably shifting the optimum toward smaller instances, while on network-enhanced families, larger instances are faster. Query-level characteristics, such as data skewness, plan complexity, and shuffle intensity, also affect the sweet spot. The above indicators of the hardware and the query should be leveraged in cluster sizing decisions, potentially with the help of machine learning.

Acknowledgments. The authors thank the anonymous reviewers for their valuable suggestions, and the Tsinghua IDEAL group members for constructive discussion. Mingyu Gao is the corresponding author. The technical solution and experimental design presented in this paper were independently completed by the authors. AI tools were used solely for language polishing and formatting optimization, and did not participate in the development of the research ideas or core content.

A TPC-DS Q23-pruned: Baseline Validation Query

Q23-pruned is a hand-simplified version of TPC-DS Q23 used as the baseline validation query in Section 2. The original Q23 contains two near-identical subquery patterns; Q23-pruned extracts the core scan, aggregation, and shuffle logic from one of them. The resulting query has balanced data, low shuffle volume, and a simple plan with few concurrent pipeline stages.

```
SELECT Count(*) AS total_rows,
       Sum(cnt) AS total_cnt
FROM   (SELECT Count(*) cnt
        FROM   store_sales, date_dim, item
        WHERE  ss_sold_date_sk = d_date_sk
               AND ss_item_sk = i_item_sk
               AND d_year IN (1999, 2000, 2001, 2002)
        GROUP  BY Substr(i_item_desc, 1, 30),
                  i_item_sk,
                  d_date
        HAVING Count(*) > 4);
```

References

1. Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI). pp. 469–482 (2017)
2. Ananthanarayanan, G., Ghodsi, A., Shenker, S., Stoica, I.: Effective Straggler Mitigation: Attack of the Clones. In: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI). pp. 185–198 (2013)
3. Apache Spark: Performance Tuning: Adaptive Query Execution. https://spark.apache.org/docs/latest/sql-performance-tuning.html (2026), Apache Spark Documentation. Accessed: 2026
4. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., Zaharia, M.: Spark SQL: Relational Data Processing in Spark. In: 2015 International Conference on Management of Data (SIGMOD). pp. 1383–1394 (2015)
5. Armenatzoglou, N., Basu, S., Bhanoori, N., Cai, M., Chainani, N., Chinta, K., Govindaraju, V., Green, T.J., Gupta, M., Hillig, S., Hotinger, E., Leshinksy, Y., Liang, J., McCreedy, M., Nagel, F., Pandis, I., Parchas, P., Pathak, R., Polychroniou, O., Rahman, F., Saxena, G., Soundararajan, G., Subramanian, S., Terry, D.: Amazon Redshift Re-Invented. In: 2022 International Conference on Management of Data (SIGMOD). pp. 2205–2217 (2022)
6. Baresi, L., Dolci, T., Quattrocchi, G., Rasi, N.: A Multi-Faceted Analysis of the Performance Variability of Virtual Machines. Software: Practice and Experience **53**(11), 2067–2091 (2023)
7. Bilal, M., Canini, M., Rodrigues, R.: Finding the Right Cloud Configuration for Analytics Clusters. In: 11th ACM Symposium on Cloud Computing (SoCC). pp. 208–222 (2020)
8. Dageville, B., Cruanes, T., Zukowski, M., Antonov, V., Avanes, A., Bock, J., Claybaugh, J., Engovatov, D., Hentschel, M., Huang, J., Lee, A.W., Motivala, A., Munir, A.Q., Pelley, S., Povinec, P., Rahn, G., Triantafyllis, S., Unterbrunner, P.: The Snowflake Elastic Data Warehouse. In: 2016 International Conference on Management of Data (SIGMOD). pp. 215–226 (2016)
9. De Sensi, D., De Matteis, T., Taranov, K., Di Girolamo, S., Rahn, T., Hoefler, T.: Noise in the Clouds: Influence of Network Performance Variability on Application Scalability. Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS) **6**(3), 49:1–49:27 (2022)
10. Fogli, A., Zhao, B., Pietzuch, P.R., Bandle, M., Giceva, J.: OLAP on Modern Chiplet-Based Processors. Proceedings of the VLDB Endowment (PVLDB) **17**(11), 3428–3441 (2024)
11. Google Cloud: Compute Engine: CPU Platforms. https://cloud.google.com/compute/docs/cpu-platforms (2026), Google Cloud Documentation. Accessed: 2026
12. Hsu, C.J., Nair, V., Freeh, V.W., Menzies, T.: Arrow: Low-Level Augmented Bayesian Optimization for Finding the Best Cloud VM. In: 38th IEEE International Conference on Distributed Computing Systems (ICDCS). pp. 660–670 (2018)
13. Intel Corporation: Intel Memory Latency Checker (MLC). https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html (2025), version 3.12. Accessed: 2026

14. Kornacker, M., Behm, A., Bittorf, V., Bobrovytsky, T., Ching, C., Choi, A., Erickson, J., Grund, M., Hecht, D., Jacobs, M., Joshi, I., Kuff, L., Kumar, D., Leblang, A., Li, N., Pandis, I., Robinson, H., Rorke, D., Rus, S., Russell, J., Tsirogiannis, D., Wanderman-Milne, S., Yoder, M.: Impala: A Modern, Open-Source SQL Engine for Hadoop. In: 7th Biennial Conference on Innovative Data Systems Research (CIDR) (2015)
15. Leis, V., Kuschewski, M.: Towards Cost-Optimal Query Processing in the Cloud. Proceedings of the VLDB Endowment (PVLDB) **14**(9), 1606–1612 (2021)
16. Leitner, P., Cito, J.: Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds. ACM Transactions on Internet Technology (TOIT) **16**(3), 15:1–15:23 (2016)
17. Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M., Vassilakis, T., Ahmadi, H., Delorey, D., Min, S., Pasumansky, M., Shute, J.: Dremel: A Decade of Interactive SQL Analysis at Web Scale. Proceedings of the VLDB Endowment (PVLDB) **13**(12), 3461–3472 (2020)
18. Microsoft Azure: Sizes for Virtual Machines in Azure. https://learn.microsoft.com/azure/virtual-machines/sizes (2026), Microsoft Azure Documentation. Accessed: 2026
19. Pedreira, P., Erling, O., Basmanova, M., Wilfong, K., Sakka, L., Pai, K., He, W., Chattopadhyay, B.: Velox: Meta's Unified Execution Engine. Proceedings of the VLDB Endowment (PVLDB) **15**(12), 3372–3384 (2022)
20. Perron, M., Fernandez, R.C., DeWitt, D.J., Madden, S.: Starling: A Scalable Query Engine on Cloud Functions. In: 2020 International Conference on Management of Data (SIGMOD). pp. 131–141 (2020)
21. Rödiger, W., Idicula, S., Kemper, A., Neumann, T.: Flow-Join: Adaptive Skew Handling for Distributed Joins over High-Speed Networks. In: 32nd IEEE International Conference on Data Engineering (ICDE). pp. 1194–1205 (2016)
22. Sethi, R., Traverso, M., Sundstrom, D., Phillips, D., Xie, W., Sun, Y., Yegitbasi, N., Jin, H., Hwang, E., Shingte, N., Berner, C.: Presto: SQL on Everything. In: 35th IEEE International Conference on Data Engineering (ICDE). pp. 1802–1813 (2019)
23. Transaction Processing Performance Council: TPC-H Benchmark Specification (2022), revision 3.0.1. Available at https://www.tpc.org/tpch/
24. Transaction Processing Performance Council: TPC-DS Benchmark Specification (2024), revision 4.0.0. Available at https://www.tpc.org/tpcds/
25. Wang, Z., Adamiak, E., Aiken, A.: A Model for Query Execution Over Heterogeneous Instances. In: 14th Conference on Innovative Data Systems Research (CIDR) (2024)
26. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM (CACM) **59**(11), 56–65 (2016)
27. Zhang, H., Liu, Y., Yan, J.: Cost-Intelligent Data Analytics in the Cloud. In: 14th Conference on Innovative Data Systems Research (CIDR) (2024)