Accordion: Balancing Performance and Cost in Cloud-Native Data Analysis with Intra-Query Runtime Elasticity

Xukang Zhang
Renmin University of China
Beijing, China
zhangxk@ruc.edu.cn

Huanchen Zhang*
Tsinghua University
Beijing, China
huanchen@tsinghua.edu.cn

Xiaofeng Meng†
Renmin University of China
Beijing, China
xfmeng@ruc.edu.cn

ABSTRACT

Cloud databases empower users to leverage vast computing resources for efficient data analysis. However, achieving cost-effective utilization of these resources remains a challenge. Users often struggle to balance computing resource allocation with their temporal and financial constraints. To address this, we propose the concept of *Intra-Query Runtime Elasticity* (IQRE), which allows a cloud-native OLAP engine to dynamically adjust the query *Degree of Parallelism* (DOP) during query execution. We introduce Accordion, the first IQRE engine. Accordion features a friendly user interface for parallelism adjustments. It includes an auto-tuner that supports both manual and automatic DOP tuning during query execution. In this demonstration, we present Accordion's architecture and provide attendees with hands-on experience, allowing them to execute queries and adjust query parallelism during query execution based on their time or cost budgets.

PVLDB Reference Format:
Xukang Zhang, Huanchen Zhang, and Xiaofeng Meng. Accordion: Balancing Performance and Cost in Cloud-Native Data Analysis with Intra-Query Runtime Elasticity. PVLDB, 18(12): 5307 - 5310, 2025.
doi:10.14778/3750601.3750658

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Blueratzxk/Accordion_engine.

1 INTRODUCTION

The development of cloud databases has significantly enhanced data analysis capabilities by leveraging the massive computational resources available in the cloud. Modern cloud-native databases and data warehouses generally adopt a storage-compute separation architecture, wherein the system is decomposed into a storage layer and a compute layer. The storage layer is responsible for persistent data storage, while the compute layer accesses data over the high-speed network for data processing. Utilizing the auto-scaling services (as shown in Figure 1) provided by cloud vendors, the two layers can be scaled independently, adding or reducing resources at any time based on the user's requirements.

When a query is submitted to a cloud database, the query engine parses it and decomposes execution into multiple parallel tasks. The number of tasks spawned is controlled by a system parameter – degree of parallelism (DOP). Higher DOP allows queries to leverage more compute resources, accelerating execution but incurring increased costs. Consequently, query execution involves a trade-off between performance and cost (as shown in Figure 1). Users may have varying cost-performance preferences for query execution. For latency-sensitive queries, users typically provide more computing resources to achieve faster execution despite higher costs. Conversely, for queries with less latency requirements, users tend to allocate fewer resources to save cost.

However, it is difficult for users to determine the optimal degree of parallelism for a query to meet time and cost constraints. This difficulty arises from the inability to accurately predict the relationship between query performance and resource consumption in advance. Current query engines require users to set parallelism before execution. The parallelism cannot be changed during data processing. So when a query's execution performance or cost fails to meet users' expectations, they have to wait for the query or terminate, reconfigure, and restart the query, resulting in wasted time and monetary costs.

In order to solve the above problem, existing approaches [5, 7] typically rely on users to provide representative workloads and construct cost-performance models after workload execution. These methods are often time-consuming, lack generalizability, and are less accessible for non-specialized users [8]. However, the time-resource relationship is not available only after the query is executed. During query execution, runtime metrics—such as table scan rates and throughput rates—can be used for predicting the remaining execution time under different degrees of parallelism. If the parallelism could be adjusted during query execution, users could balance the cost and performance more flexibly and effectively.

For intra-query elasticity, currently, the database and big data fields use "dynamic query optimization" to change resource usage during query execution. It can be categorized into three types: adaptive query processing [10], adaptive query execution [1], and query re-planning [6]. These approaches require pausing query execution and materializing intermediate results, cannot support flexible DOP tuning, and may incur additional time and cost overheads.

Our proposal. In this paper, we introduce a novel concept called *Intra-Query Runtime Elasticity* (IQRE) and present the first IQRE query engine, **Accordion**. IQRE enables the dynamic adjustment

*Huanchen Zhang is also affiliated with the Shanghai Qi Zhi Institute.
†Xiaofeng Meng is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750658

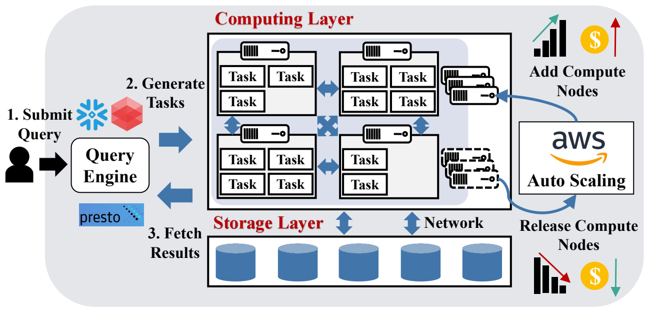


Figure 1: Cost-performance tradeoffs of cloud data analysis.

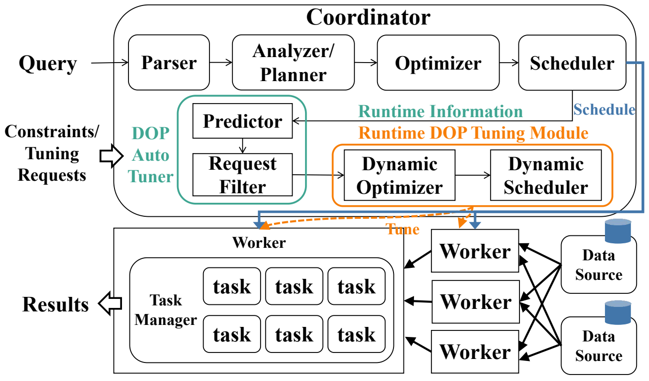


Figure 2: Architecture of Accordion.

of query parallelism during execution without pausing data processing. Accordion allows users to initiate a query with minimal computational resources and subsequently adjust execution speed or resource consumption based on their requirements. Accordion is implemented from scratch in C++, following the execution model of Presto [2], an open-source distributed SQL query engine developed at Meta. Unlike morsel-driven parallelism [4], which optimizes fine-grained thread scheduling for query in single-node multi-core systems, IQRE focuses on optimizing query performance and cost runtimely in elastic, distributed resource environments.

Demonstration. Participants can interact with Accordion to experience its (a) friendly user interface, (b) detailed, real-time query runtime information display, and accurate execution progress indication, and (c) interactive parallelism tuning interface, including a DOP tuning guide service, and a DOP auto-tuning service.

2 SYSTEM OVERVIEW

This section presents the architecture and design of Accordion.

2.1 Accordion Architecture

Accordion is a distributed query engine that adopts the vectorized push-based model. As illustrated in Figure 2, a Accordion cluster consists of a coordinator node and multiple worker nodes. The coordinator is responsible for query parsing, analyzing, planning, optimizing, and task scheduling. Worker nodes are responsible for query processing and result return. Upon receiving a query, the

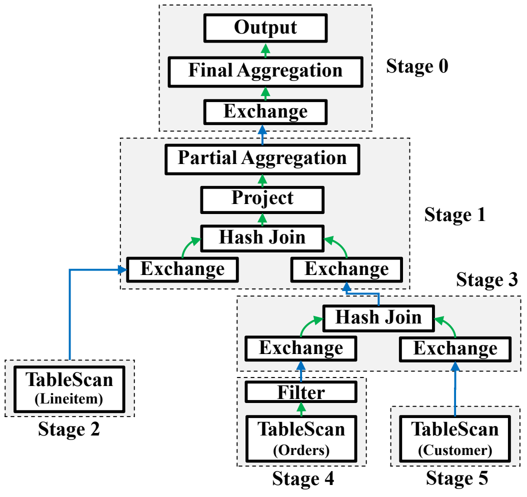


Figure 3: The distributed physical plan of the example query.

coordinator analyzes the SQL statement, generates a distributed physical plan through optimization, and then schedules tasks — the smallest unit for distributed execution — on the worker nodes. Each worker node contains a task manager for creating and terminating tasks. Worker nodes execute these tasks to process data from base tables or to handle intermediate data generated by other workers. Accordion uses the Apache Arrow [3] as the data exchange format.

Once a query is scheduled for execution, Accordion is able to modify the parallelism of the query by performing an operation on the query at runtime. This operation is implemented by two modules, the automatic DOP tuner (green) and the runtime DOP tuning module (orange). The auto-tuner contains a predictor and tuning request filter. The Predictor handles prediction tasks. It obtains query runtime information from the scheduler to estimate the remaining execution time and the anticipated time after parallelism adjustments. These results are returned to users or used for DOP auto-tuning. The request filter is used to filter unreasonable tuning requests (e.g., requests that would cause a waste of resources and requests for finished queries). The runtime DOP tuner includes a dynamic optimizer and scheduler. When a tuning request arrives, the optimizer decides the DOP tuning type and invokes the scheduler to execute it.

2.2 Query Execution Model

This section describes the details of query plan construction and DOP runtime tuning. Consider an SQL statement:

```sql
SELECT c_custkey, SUM(l_extendedprice)
FROM lineitem
    INNER JOIN orders   ON l_orderkey=o_orderkey
    INNER JOIN customer ON c_custkey=o_custkey
WHERE o_orderdate < 1994-03-05 GROUP BY c_custkey
```

Accordion obtains a physical plan after parsing, analyzing, and optimizing the query. During query optimization, there are two special types of nodes in the plan: the exchange node and the local exchange node. These nodes are introduced during the query optimization phase to partition the plan into sub-plans.

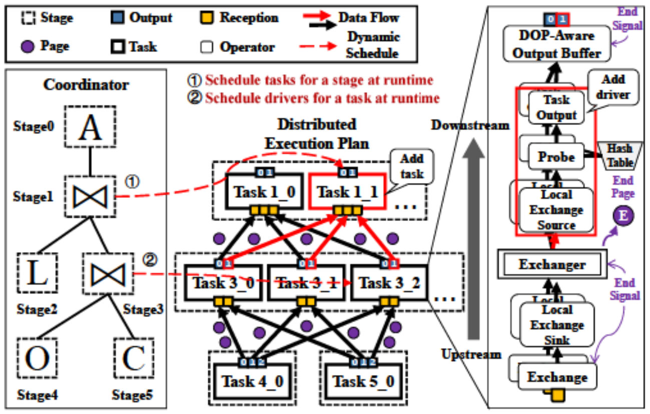


Figure 4: DOP tunings in Accordion.

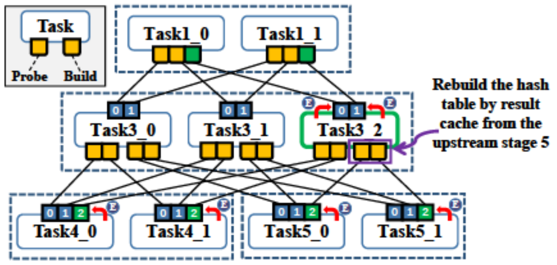


Figure 5: Increasing or decreasing the DOP of stage 3.

The query optimizer divides the physical plan into multiple fragments based on the locations of the exchange nodes, resulting in a fragment (stage) tree as illustrated in Figure 3. During the query scheduling phase, the scheduler constructs an initial distributed execution plan based on the stage tree, traversing it in a bottom-up manner to generate tasks for each stage and establish communication links between them. Figure 4 presents a partially distributed execution plan for the stage tree, displaying only stages 1, 3, 4, and 5. Each stage is assigned some tasks, with each task identified by a unique task ID that consists of the stage number and the task sequence number.

A fragment cannot be executed directly within a task; it first be subdivided into a collection of pipelines. The division is performed by pipeline breakers, which include the local exchange node and the hash join node in this plan. A pipeline produces multiple drivers, each representing an executable physical operator sequence that can be scheduled on threads.

2.3 Intra-Query DOP runtime tuning

Figure 4 shows the two DOP tuning types supported by Accordion, the intra-task DOP tuning (①) (changing the number of drivers for the pipeline) and the intra-stage DOP tuning (②) (changing the number of tasks for stage).

Intra-Task DOP Tuning. Accordion dynamically creates new drivers during task execution to increase intra-task DOP (Figure 4). To gracefully terminate a driver, Accordion uses an "end page" mechanism—a special page with no data that notifies drivers of normal execution completion. The end page propagates through all operators in the driver, ensuring proper termination. To reduce task parallelism, the system sends an end signal to components like the exchanger, task output buffer, and exchange operator, which generate end pages to notify downstream tasks and drivers to exit.

Intra-Stage DOP tuning. Figure 5 shows the stage DOP tuning process, using the partial execution plan for the query shown in Figure 3. Increasing stage parallelism requires three steps: (1) generating task3_2 for stage 3, (2) adding the address of task 3_2 to the tasks of the downstream stages of stage 3. (3) task3_2 notifies the tasks of the upstream stages to generate a new buffer ID "2" for data exchange (stage 3 contains the join operation, so task3_2 needs to get data from the tasks in stage 5 to rebuild the hash table). To reduce the parallelism of stage 3, the system sends end signals to the task output buffers of the upstream stage, specifying the buffer ID "2" to be closed. The corresponding output buffers generate end pages, informing the task3_2 in stage 3 to terminate. After each driver in task3_2 exits normally, the entire task exits normally. Then task1_0 and task1_1 of stage 1 will delete the address of task3_2. At this point, the DOP reduction for stage 3 is complete. Data processing is not paused during all DOP tunings. More details about DOP tuning and overhead can be found in our research paper [9].

2.4 Automatic DOP Tuning

Accordion provides a user-friendly interface for tuning query parallelism, with the auto-tuner managing requests uniformly. Users can send two types of requests to the auto-tuner, "stage/task parallelism tuning request" (where users manually adjust parallelism) and "stage execution time constraint request" (where parallelism is automatically determined based on the desired execution time). The auto-tuner incorporates three key components: runtime bottleneck localization to identify stages requiring tuning, request filter to discard invalid requests, and stage remaining time prediction, which estimates stage runtime under different DOPs based on table scan rates. More details are available in our research paper [9].

3 DEMONSTRATION OVERVIEW

This section presents the setting and plan of our demonstration.
Demonstration Setup. The following describes the system setup.

Environment. We will deploy Accordion on AWS for demonstration using 21 EC2 (c5.2xlarge) instances, each with 16GB RAM, 30GB SSD, and 10Gbps NIC. The deployment consists of 10 storage nodes, 10 compute nodes, and 1 coordinator node. Among the compute nodes, 5 nodes form the initial cluster, while the remaining 5 nodes are used for resource scaling.

Datasets. We will demonstrate Accordion using TPC-H with SF100. Accordion currently supports both built-in TPC-H queries and simple custom SQL queries without sub-queries.
Demonstration Plan. We will showcase Accordion as follows. Users are invited to play with the interface of Accordion and explore its internal functionalities to understand how these could adjust the performance/cost of queries.

System User Interface. Upon starting the Accordion engine, users can access its main interface via "IP:9082", as shown in Figure 6. The interface consists of a query input box on the left and a query progress indicator on the right. Users can configure the initial task/stage DOP through a configuration file (default: 1, meaning the query starts with minimal computational resources).

Users enter SQL statements into the input box to execute queries. Once a query is submitted to the Accordion cluster, the execution progress is dynamically visualized in the right panel. Each query is represented by multiple progress bars, corresponding to its table scan stages. Query execution is considered complete when all progress bars are fully filled.

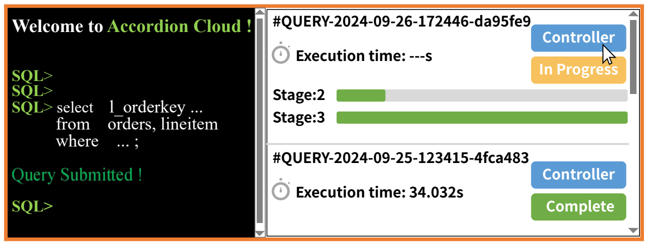
When query execution begins, users may find the progress bar advancing slowly and become dissatisfied with performance, as queries are initially executed in a "cost-first" manner by default. To accelerate execution, users can leverage IQRE. As shown in Figure 6, clicking the "Controller" button opens the controller interface (Figure 7). The controller interface provides comprehensive query execution insights, including (1) the query plan, (2) throughput rates and estimated remaining execution times for each stage, and (3) DOP information of stages, and task pipelines. The interface offers a clear overview of the query execution, enabling users to effectively monitor performance and manage resource allocation.

Manual DOP Tuning. To assist users in manually adjusting the degree of parallelism (DOP), the auto-tuner provides a guided service. Users begin by entering a DOP multiplier and clicking the "Get Tip" button (box ① in Figure 7). The system then displays suggestions in the information box at the top of the interface (box ② in Figure 7). In this case, the auto-tuner recommends increasing the parallelism of stage 1, as it is currently the execution bottleneck. The estimated remaining execution time after adjustment is 12.43 seconds.

To apply the suggested changes, users click the "Stage 1" button (box ③ in Figure 7) and then select "Add Task" to increase the number of tasks for stage 1 (add two tasks). Additionally, they can click the "Driver" button to enhance intra-task parallelism. Upon a successful request, the "Response" box will display "ACCEPT". At this point, Accordion adds compute nodes and adds tasks for the query, leading to a noticeable increase in the throughput of stage 1. When users return to the main interface, they will observe a faster progress bar, reflecting improved execution speed. After the query finishes, users get the result using the "Complete" button.

Users can refine the initial query parallelism through an iterative process of executing queries and adjusting DOPs, updating the configuration file as needed. However, as parallelism settings evolve, some queries may consume more resources than necessary, especially when fast execution is not a priority. To optimize resource utilization and reduce costs, users can leverage the "Close Task" button in the interface to dynamically lower the parallelism at each query stage, freeing up compute resources.

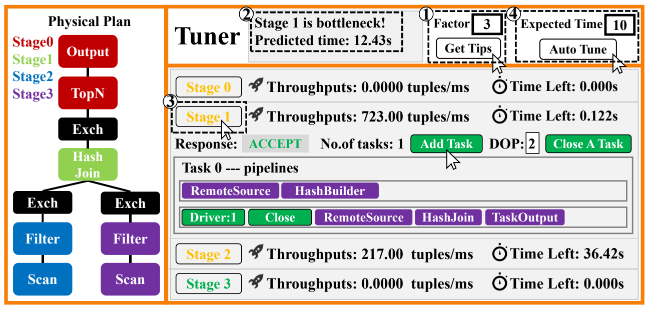
Automatic DOP Tuning. Users may find it tedious to adjust DOP manually, so Accordion provides an automatic DOP tuning service. Users simply enter the desired execution time for the current stage in the "Expected Time" box and click the "Auto Tune" button (box ④ in Figure 7). Accordion then automates the tuning process by identifying the stage requiring adjustment, analyzing its remaining execution time, and optimizing the DOP accordingly. The system ensures that the selected DOP meets the user's time constraints while minimizing computational resource consumption.


```
Welcome to Accordion Cloud !

SQL>
SQL>
SQL> select   l_orderkey ...
     from     orders, lineitem
     where    ... ;

Query Submitted !

SQL>
```

#QUERY-2024-09-26-172446-da95fe9
Execution time: ---s
Stage:2
Stage:3

#QUERY-2024-09-25-123415-4fca483
Execution time: 34.032s

Figure 6: The main interface – it includes a SQL input box on the left and the query execution progress tracking box on the right.


Stage 1 is bottleneck!
Predicted time: 12.43s

Factor 3
Expected Time 10

Stage 0 Throughputs: 0.0000 tuples/ms Time Left: 0.000s
Stage 1 Throughputs: 723.00 tuples/ms Time Left: 0.122s
Response: ACCEPT No.of tasks: 1 DOP: 2
Task 0 --- pipelines
RemoteSource HashBuilder
Driver:1 Close RemoteSource HashJoin TaskOutput
Stage 2 Throughputs: 217.00 tuples/ms Time Left: 36.42s
Stage 3 Throughputs: 0.0000 tuples/ms Time Left: 0.000s

Figure 7: The query controller interface – includes the query plan box, the auto-tuner box, and the stage-info box.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (No. 62172423).

REFERENCES

[1] 2025. https://docs.databricks.com/en/optimizations/aqe.html.
[2] 2025. https://github.com/prestodb/presto.
[3] 2025. https://github.com/apache/arrow.
[4] 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* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 743–754. https://doi.org/10.1145/2588555.2610507
[5] Viktor Leis and Maximilian Kuschewski. 2021. Towards cost-optimal query processing in the cloud. *Proc. VLDB Endow.* 14, 9 (May 2021), 1606–1612. https://doi.org/10.14778/3461535.3461549
[6] Kshiteej Mahajan, Mosharaf Chowdhury, Aditya Akella, and Shuchi Chawla. 2018. Dynamic query re-planning using QOOP. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 253–267.
[7] Chunxu Tang, Beinan Wang, Zhenxiao Luo, Huijun Wu, Shajan Dasan, Maosong Fu, Yao Li, Mainak Ghosh, Ruchin Kabra, Nikhil Kantibhai Navadiya, Da Cheng, Fred Dai, Vrushali Channapattan, and Prachi Mishra. 2021. Forecasting SQL Query Cost at Twitter. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. 154–160. https://doi.org/10.1109/IC2E52221.2021.00030
[8] Huanchen Zhang, Yihao Liu, and Jiaqi Yan. 2024. Cost-Intelligent Data Analytics in the Cloud. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p78-zhang.pdf
[9] Xukang Zhang, Huanchen Zhang, and Xiaofeng Meng. 2025. Intra-Query Runtime Elasticity for Cloud-Native Data Analysis. *Proc. ACM Manag. Data* 3, 3, Article 178 (June 2025), 28 pages. https://doi.org/10.1145/3725315
[10] Junyi Zhao, Huanchen Zhang, and Yihan Gao. 2023. Efficient Query Re-optimization with Judicious Subquery Selections. *Proc. ACM Manag. Data* 1, 2, Article 185 (June 2023), 26 pages. https://doi.org/10.1145/3589330