Hercules: *Fast Analysis, No Paralysis* in GaussDB

Vasilis Gavrielatos, Antonios Katsarakis, Evangelos Vazaios, Maurice Baileu,
Evangelos Maliaroudakis, Giorgos Stilianakis, Massimo Perini, Mahesh Dananjaya,
Colin Reynolds, Pawel Guzewicz, Antonis Papaioannou, Lefteris Zervakis, Anderson Chaves Carniel,
Zhang Jiahao*, Zhou Pinggao*, Pratanu Roy, Nikos Ntarmos
Huawei Company*, Huawei Technologies R&D (UK) Ltd

Abstract

Modern many-core servers with terabytes of memory should deliver fast analytical (AP) queries while still honoring transactional (TP) performance when both run concurrently. In practice, stream-centric analytical execution with blocking exchanges causes *analysis paralysis*: long-running analytical pipelines monopolize CPU and memory bandwidth, depressing the throughput and latency of short-lived transactions. At the same time, analytics performance falls short: vectorized engines commonly incur avoidable data copies and redundant work, and their dominant operators (hash joins and hash aggregations) rely on coordination schemes that spawn many execution threads while also amplifying synchronization overheads, stalls, and context-agnostic OS scheduling.

We present Hercules, a hybrid TP/AP execution framework implemented inside *GaussDB*, that co-designs and enhances proven techniques into three pillars. ❶ *Fast Analysis* removes unnecessary copies and processing via early bloom filters, late materialization, and intermediate result sharing. ❷ *Fast Dominant Operators* rebuilds joins and aggregations as concurrency-first primitives that exploit partition-wise execution, concurrent hash tables, and non-blocking streams. ❸ *No Paralysis* introduces suspendable, task-based execution with user-level scheduling and NUMA-awareness.

On a Kunpeng 920 Arm server running TPC-H scale factor (SF) 100, Hercules achieves 4.1× higher throughput and 4.3× lower latency than the original GaussDB, 2.5× lower latency than the fastest competitor engine, and completes 5 concurrent TPC-H instances faster than that engine completes 1. Meanwhile, under mixed transactional/analytical workloads, Hercules preserves the high throughput for short-running transactions via user-level scheduling while performing analytics efficiently with locality-aware, preemptible execution.

1 Introduction

Hybrid TP/AP engines promise to serve both workloads from a single system, yet on modern many-core NUMA servers the promise breaks down. When analytical queries run alongside transactions, stream-centric execution with blocking exchanges causes *analysis paralysis*: each query spawns dozens of threads that require expensive data shuffling (i.e., stream data exchanges), and a single stalled thread could block every downstream consumer. Analytical (AP) performance is far from optimal in isolation, and transactional (TP) performance dips significantly under AP interference. For example,

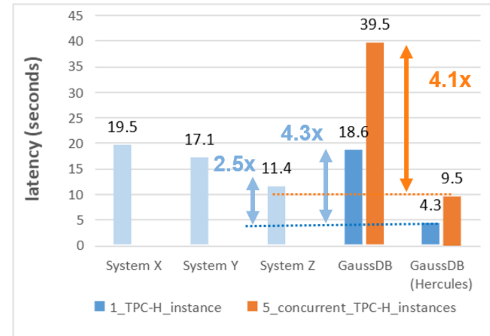

Figure 1. *TPC-H SF100 completion in state-of-the-art analytical engines (also 5 concurrent instances for GaussDB/Hercules).*

on a Kunpeng 920, GaussDB spawns numerous threads under concurrent TPC-H load, which collapses TPC-C throughput to almost half the moment AP queries arrive (Figure 2).

Root causes span three execution layers: wasted work and data copies, inefficient operators, and agnostic scheduling. First, vectorized engines perform avoidable work: copying columns that will be filtered away, duplicating I/O and computation, and materializing intermediate results no downstream operator needs. Second, hash joins and aggregations — the most dominant analytical operators [2, 3] — rely on data shuffling and barrier schemes that incur costly copies, serialize under concurrency, and ignore NUMA topology. Third, thread-based OS scheduling cannot preempt a long-running analytical pipeline for a short transaction, nor schedule work to a NUMA-local core. Hercules addresses all three via hardware-conscious co-design and the expansion of known techniques inside GaussDB — not through any single new mechanism, but via their joint effect on modern hardware.

Hercules: Fast Analysis. The first pillar chains early bloom filters; late materialization; late reads; and intermediate sharing, so downstream operators work on fewer data or avoid work altogether. Early bloom filters are constructed dynamically during the hash build of a join (eschewing pre-processing overheads needed by techniques like Yannakakis [6]) and pushed below to the scan operators, discarding rows that will fail downstream join predicates before they enter the pipeline. Because fewer rows survive filtering, late materialization becomes more effective: columns not required until a later operator are neither read from storage nor copied into intermediate buffers at the scan stage, saving both memory bandwidth and cache capacity. Ultra-late

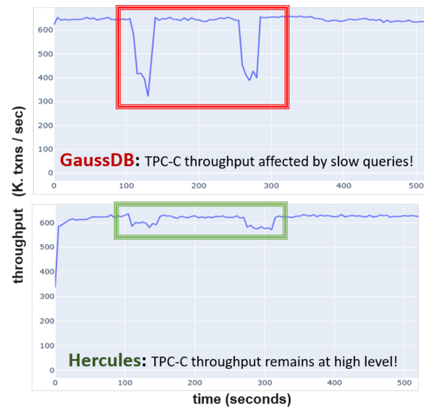

Figure 2. *TPC-C throughput in isolation and while injecting 5 concurrent instances of TPC-H SF100 Q1 (at ~120 & 280 sec).*

materialization (late reads) extends this principle further — even for rows that survive the bloom filter, column values are not fetched until the latest possible operator that actually consumes them, so intermediate pipeline stages carry only compact row identifiers and join keys rather than full tuples. Finally, when multiple concurrent queries overlap in their scan ranges or join inputs, intermediate result sharing allows one query's already-filtered and materialized output to serve another, avoiding redundant I/O and computation. Each mechanism compounds the savings of its predecessor: bloom filters eliminate rows, late materialization avoids copying columns for eliminated rows, late reads defer the remaining columns, and sharing skips duplicating work.

Hercules: Fast Dominant Analytic Operators. Even after the first pillar reduces data volume, hash joins and hash aggregations remain the dominant cost in analytical workloads. Under concurrency, conventional implementations become bottlenecks: they partition inputs, synchronize at barriers, and spawn per-partition threads that contend on shared state. Hercules reconstructs these operators as concurrency-first primitives while minimizing stream overhead. Concurrency-aware (CA) hash joins and CA hash aggregations operate over fast hash tables, specialized variants of DLHT [4]. CA hash joins employ shared hash tables with efficient concurrent insertions and probes at degrees of parallelism greater than one, eliminating the partition and barrier coordination pattern entirely.

CA hash aggregations avoid a shared hash table to eschew synchronization costs and instead leverage non-blocking streaming to redistribute data with fewer copies and stalls than original stream-centric operators. Finally, when input data is already partitioned by join or group keys, partition-wise execution detects this and maps tasks to existing partitions, fully avoiding synchronization and streaming costs for both hash joins and aggregations. The combined effect of concurrent hash tables, non-blocking streams, and partition-wise execution removes unnecessary shuffles, blocking, and coordination, ensuring that the dominant operators scale with core count rather than collapse under concurrency.

Hercules: No Paralysis. Reducing work and removing operator-level bottlenecks is insufficient if the scheduler cannot prioritize between TP and AP workloads or exploit hardware locality. Thread-based execution assigns a fixed number of OS threads per query; these threads cannot be preempted without kernel intervention, and the OS scheduler is unaware of workload priorities or NUMA topology. Hercules replaces this model with suspendable task-based execution using user-level scheduling.

Every query plan is decomposed into fine-grained tasks that the Hercules scheduler can suspend, resume, and reorder. When an analytical task blocks — because its input is empty or its output is full, it gets suspended, and the scheduler immediately dispatches a ready task, keeping the core productive. TP tasks receive scheduling priority, ensuring transactional performance remains high even under heavy analytical load. NUMA-aware placement and scheduling complement this by binding tasks and their associated memory — hash tables, intermediate buffers, scan ranges — to the same NUMA node. The scheduler dispatches tasks to cores local to their data, minimizing cross-socket memory traffic. Together, suspendable tasks provide TP/AP prioritization while NUMA awareness maximizes locality.

Evaluation. We compare against 3 state-of-the-art analytical engines, reported as X, Y, and Z (a leading commercial engine) to respect DeWitt's clause [5], and the original GaussDB [1] on a single Kunpeng 920 server (2 sockets, 4 NUMA nodes, 160 cores, 320 HW threads, 2TB memory). On TPC-H SF100, Hercules achieves 4.1× higher throughput and 4.3× lower latency than original GaussDB, and 2.5× lower latency than the fastest competitor (Figure 1). In fact, under 5 concurrent TPC-H instances, Hercules finishes faster than the fastest competitor finishes 1. In a mixed TP/AP load, Hercules preserves 95% of TPC-C original throughput while GaussDB collapses to half (Figure 2), confirming the transactional benefits of suspendable task-based execution.

Summary. Hercules, implemented in GaussDB, eliminates analysis paralysis through work reduction, concurrency-first hash joins and hash aggregations, and user-level NUMA-aware task scheduling, achieving 4.1× higher TPC-H throughput and 4.3× lower latency while preserving high transactional performance under mixed load.

References

[1] GaussDB. https://www.huaweicloud.com/product/gaussdb.html, 2026.
[2] Peter Boncz, Thomas Neumann, and Orri Erling. TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark.
[3] Markus Dreseler and et. al. Quantifying TPC-H choke points and their optimizations. *Proc. VLDB Endow.*, 13(8):1206–1220, April 2020.
[4] Antonios Katsarakis, Vasilis Gavrielatos, and Nikos Ntarmos. DLHT: A non-blocking resizable hashtable with fast deletes and memory-awareness. HPDC '24, page 186–199, New York, NY, USA, 2024.
[5] Brian Moran. The devil's in the dewitt clause. SQL Server Magazine, April 2003.
[6] Hangdong Zhao and et al. I can't believe it's not yannakakis: Pragmatic bitmap filters in microsoft sql server. CIDR '26.