Accelerate AI and Analytics Workloads With Faster Data Access

Performance Benchmark Summary and Sizing Recommendations

Overview

Enterprise AI and analytics workloads are increasingly constrained by slow, single-threaded data access. Arrow Flight SQL accelerates governed access to Teradata data, delivering up to 84× faster read throughput enabling faster AI, analytics, and Python workflows. This datasheet summarizes Arrow Flight SQL Server (Teradata Fabric Arrow connector) benchmarks and gives sizing guidance for customers.

Apache Arrow is an in-memory columnar format. Apache Arrow is extended by **Apache Arrow Flight** to allow transfer of Arrow data over the wire using gRPC protocol (allowing multiple steams in parallel). One final extension is **Apache Arrow Flight SQL** which applies this protocol to interacting with SQL databases.

The dominant protocol thus far for SQL databases is JDBC. However, there are limitations. It is by design single threaded for data reads. The Teradata Arrow Flight SQL Server can deliver data in parallel, columnar gRPC streams; making **reads (downloads) dramatically faster than native JDBC**. Our testing sought to compare data reads with traditional JDBC and new Teradata Arrow Flight SQL Server capability; testing both single stream Arrow Flight SQL and multistream.

This was done on the Teradata Cloud platform for all major CSPs (AWS, Azure, Google Cloud) with similar instance types and additionally on a higher-tier AWS configuration. Several tests were also performed to measure the upload using JDBC and Arrow Flight SQL (Arrow Flight's DoPut). Add this to the end of this paragraph: Your results may vary and you may need tuning based on your ecosystem.

Our testing results and recommended configuration

Recommended configuration (budget option)

Deploy **2 Arrow driver nodes**, **1 stream per node**, with at least **8 GB Java heap + 8 GB direct memory** (16 GB direct memory preferred). Against the 2-node standard Teradata database (r6i.2xlarge / E8s v5) used in testing, this roughly doubled throughput versus a single driver node, removed out-of-memory (OOM) failures, and held steady through concurrency 20—a cost-effective starting point. The right driver-node count scales with the database's size and I/O (see Scaling note).

Scaling note

Customers can add more Arrow driver nodes to push throughput higher, but extra nodes help little once the Teradata database becomes the bottleneck. A database with limited I/O caps total throughput regardless of driver count—to go faster, scale the database (larger instances or more nodes) first (see below).

Best read throughput on standard hardware

~270–350 MB/s with 2 driver nodes, holding through concurrency 20 (e.g., Azure: 267 MB/s, +120% over a single driver node). For high concurrency on a single node, record batch 1000 / buffer 4 sustains ~200 MB/s to concurrency 20 without OOM.

Versus JDBC

JDBC is a single-stream client path that bypasses the Arrow driver and measured only ~10–19 MB/s—row-by-row and client-bound, so largely hardware-independent. Against that, standard-hardware Arrow is an estimated **~15–30× faster** with 2 driver nodes (~10–20× single-node, tuned); the baseline and the measured **47–84×** come from the higher-tier run below.

Recommended sizing by cloud provider (standard Teradata Cloud)

Cloud	Arrow driver nodes (recommended)	Teradata DB nodes (test setup —not a recommendation)
AWS	2× m5n.2xlarge (8 vCPU / 32 GB)	2× r6i.2xlarge (8 vCPU / 64 GB)
Azure	2× D8s v5 (8 vCPU / 32 GB)	2× E8s v5 (8 vCPU / 64 GB)
Google Cloud	2× c3-standard-8 (8 vCPU / 32 GB)	2× c3-highmem-8 (8 vCPU / 64 GB)*

Cost covers the Arrow driver nodes only—the component customers scale; the Teradata DB nodes are shown only as the benchmark test environment (not a sizing recommendation) and are billed separately. Driver cost: on-demand, Linux, single US region, ~730 hrs/month; reserved / committed-use plans cut it ~30–60%. *Google Cloud DB instance type was not specified in the benchmark; a comparable 8 vCPU / 64 GB instance is shown as the test environment.

Higher-tier hardware: Improvement over standard

Read throughput improves with more compute. All rows below run on a **2-node Teradata database**—gains come from a larger DB instance and larger/more driver nodes—a mid-size system, so headroom remains with a more powerful database. (Throughput is bound by Teradata AMP I/O, ~870 MB/s, not the Arrow driver—CPU plateaus near 40%—so scale the database first; memory and streams won't help.)

Configuration	Arrow driver node(s)	Teradata DB node(s) (test setup)	Peak read	Improvement
Standard hardware (recommended)	2× standard (8 vCPU / 32 GB)	2× r6i.2xlarge	~270–350 MB/s	baseline
Upgraded Teradata DB instance	3× standard (8 vCPU / 32 GB)	2× r6id.12xlarge	~590 MB/s	**~2× baseline**
Higher-tier compute (AWS)	3× mn5n.8xlarge	2× r6id.12xlarge	**1,276 MB/s**	**~4× baseline**

Standard-instance runs hold steady through concurrency 20 with 2 driver nodes (Azure: 267 MB/s peak, +120% over a single driver node). Every row uses a 2-node Teradata database—not the largest configuration—so further headroom exists with a more powerful database. The Arrow-vs-JDBC comparison (47–84×, 2–10× less CPU) was measured on the higher-tier run. Standard Teradata Cloud drivers: m5n.2xlarge (AWS), Standard_D8s_v5 (Azure), c3-standard-8 (GCP).

Configuration and tuning Notes

Direct memory: 8 GB minimum (2 GB fails even at 5 concurrent); 16 GB is the sweet spot (64 GB is 5–23% slower due to Netty fragmentation). The unpooled Netty allocator lowers peak retained memory at high concurrency.

Streams per node: 1 stream per node is optimal on memory-constrained standard instances. 2 streams/node adds a little throughput on high-memory hardware but risks OOM at high concurrency.

Upload (DoPut) vs. direct JDBC

Uploads use a **single stream**, so they do not get the multi-stream read speedup; the goal is parity with JDBC, not faster.

Upload size (single session)	Arrow vs. JDBC	5 concurrent (small ≤89 MB)	5+ concurrent (≥449 MB)
100 MB	–9.5%	Pass (5/5)	—
500 MB	–2.4%	—	Journal overflow
1 GB	**–1.4% (near parity)**	—	Journal overflow

Takeaway: Single-session upload is at near-parity with JDBC (~1% overhead at 1 GB). Large concurrent uploads (≥449 MB at 5+ sessions) hit Teradata journal overflow (Error 1338) in **both Arrow and JDBC**—a DB-side limit, not Arrow-specific. Recommended: batch ≤20K rows, commit every ≤300K rows, and keep per-session data under ~100 MB (or concurrency 1–3) for large uploads.

Single-stream direct query (download) vs. direct JDBC

Certain queries (e.g., with ORDER BY) use the single-stream Direct Query path and run **~10 – 15% slower than direct JDBC**: the server converts the JDBC ResultSet to Arrow and adds a gRPC/TLS hop, with no second stream to offset that cost. The 47 – 84× advantage is for the parallel read path, not Direct Query.

Direct Query size	Arrow MB/s	JDBC MB/s
100 MB	30.4	33.8
500 MB	39.9	46.2
~780 MB	40.3	47.5

Recommendations Summary

Reads

On standard hardware, 2 driver nodes / 16 GB direct memory gave ~270–350 MB/s, no OOM through concurrency 20 (any cloud)—an estimated ~15–30× over single-stream JDBC. Scale the DB first for more (driver nodes alone won't help once the DB is the bottleneck); a higher-tier 2-node system hit 1,276 MB/s (~4×, measured 47 – 84× vs JDBC).

Uploads and Direct Query

Single-stream paths—expect JDBC parity on upload (keep per-session small, commit every ≤300K rows) and ~10–15% below JDBC on single-stream Direct Query.

17095 Via Del Campo, San Diego, CA 92127 Teradata.com

teradata.