CLOUDERA | NVIDIA

WHITE PAPER

TURBOCHARGE YOUR ETL PIPELINES WITH NVIDIA GPUS AND CLOUDERA DATA PLATFORM

CLOUDERA DATA PLATFORM

Table of Contents

Executive Summary

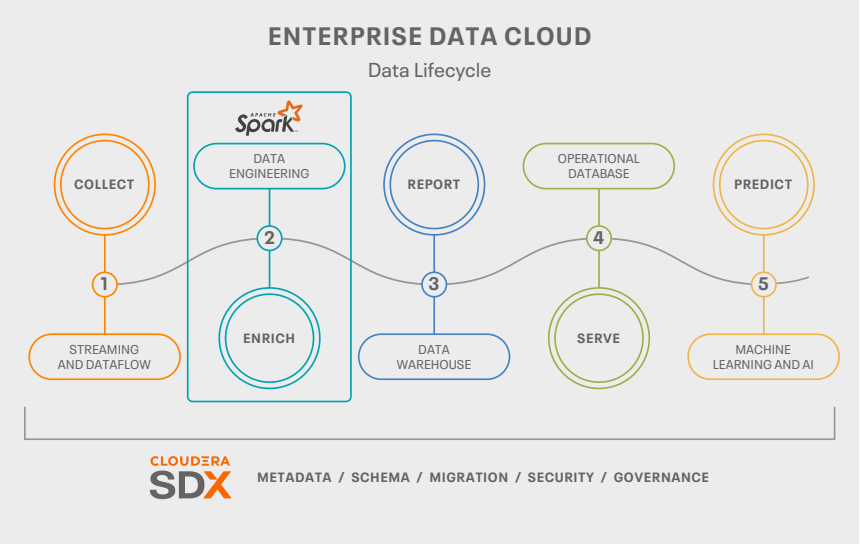
- Realize massive performance gain on ETL pipelines and analytic queries with Cloudera Data Platform and NVIDIA Ampere GPUs
- Enabling GPU acceleration for Apache Spark's data frame API leads to speedups of 7x overall on a suite of representative decision support queries and speedups of over 15x on select individual queries.
- This capability is jointly supported by Cloudera and NVIDIA and is generally available in Cloudera Data Platform for on-premises customers.

Introduction

You probably already know that NVIDIA GPUs power accelerated machine learning and have made the deep learning revolution possible. However, you may not know that the same powerful accelerators that enable computers to recognize your friends and family in your photos, understand the questions you ask your virtual assistant, and safely drive your car can also turbocharge your enterprise analytics, data engineering, and database workloads. The RAPIDS Accelerator for Apache Spark is a plug-in to accelerate large scale ETL operations using GPUs with zero code change to existing data pipelines..

Cloudera Data Platform (CDP) is a hybrid data platform designed for unmatched freedom to choose—any cloud, any analytics, any data. CDP delivers faster and easier data management and data analytics for data anywhere—public cloud or on premises, with optimal performance, scalability, and security. Cloudera and NVIDIA have partnered to bring GPU acceleration for Apache Spark to enterprises everywhere by shipping and supporting the RAPIDS Accelerator for Apache Spark in CDP Private Cloud Base.

In the rest of this paper, you'll see the benefits of GPU acceleration for analytics and data engineering workloads and learn how to get started with Cloudera Data Platform and NVIDIA GPUs.



Data-driven enterprises depend on advanced analytics to support key business goals, and analytic capabilities depend upon managing, organizing, and characterizing massive data at scale. Business analysts develop complex queries on structured data to characterize it and determine whether or not there are patterns to exploit. Data engineers federate, enrich, and publish data for internal consumption.

Almost every enterprise benefits from data science, advanced analytics, and database queries at scale, and many organizations use Apache Spark to realize these capabilities. Apache Spark's Data Frame API and relational query capabilities support crucial data-understanding, cleaning, and federation tasks. In organizations that have adopted machine learning in production, structured-data and ETL workloads are no less valuable; rather, they remain valuable in themselves and a necessary prerequisite to more advanced techniques.

Accelerating Data Engineering Workloads

Relational databases and data-processing frameworks like Apache Spark evaluate structured queries by constructing, transforming, optimizing, and executing a query plan. This plan includes an order in which to execute subqueries and concrete strategies for filtering and joining relations (for example, whether to use an index or a hash of key values). The process of planning enables the execution engine to evaluate a query more efficiently than a naive approach would by taking tunable heuristics, metrics about data sources, and details of data distributions and indices into account.

The RAPIDS Accelerator for Apache Spark provides transparent acceleration of Spark data frame jobs via a plugin that integrates with Spark's query planner. The plugin rewrites data frame query plans in order to evaluate accelerable operations with implementations that execute on the GPU. Operations that cannot be accelerated will run on the CPU with Spark's built-in implementations; if there is a branch of a query plan that includes both accelerable and non-accelerable operations, the RAPIDS Accelerator plugin will automatically insert transfers between host and device memory so that both kinds of operations can work together transparently to execute a given query plan.

Figure 1 shows the results of rewriting a query plan, which is represented as a graph. The graph on the left represents a low-level plan ready for execution on Apache Spark without GPU acceleration; the root represents the node computing the ultimate result (e.g., returning a collection to memory or writing output to a Parquet file) and the other nodes represent intermediate steps along the way. The graph on the right shows a query plan after transformation by the RAPIDS Accelerator; the green nodes represent intermediate operations that will be scheduled on the GPU. Note that one node in the transformed plan still executes on the CPU; the orange nodes, which are inserted transparently, represent transferring records from the GPU to main memory and back.

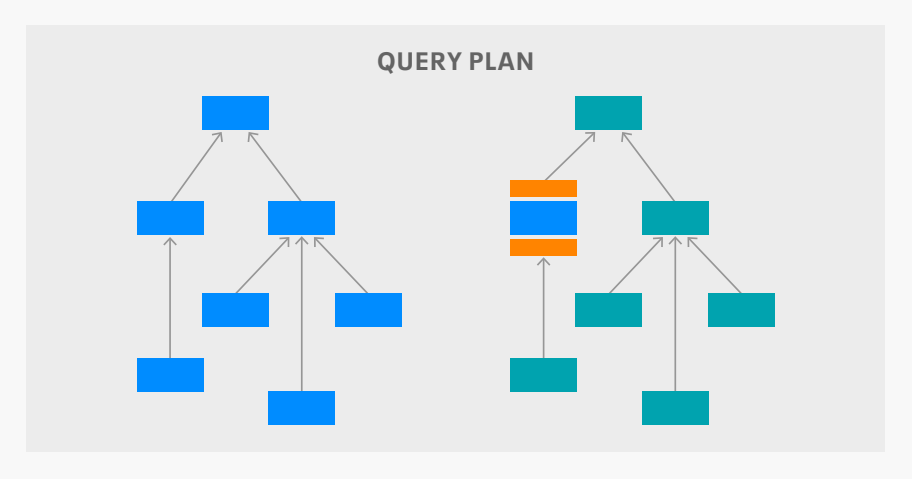


Figure 1. A Spark query plan, before (left) and after (right) transformation by the RAPIDS Accelerator. Blue nodes represent operations executed on the CPU, green nodes represent operations executed on the GPU, and orange nodes represent transfers between main memory and GPU memory.

Transparent acceleration has both a benefit and a cost for users. The benefit of transparent acceleration is that users can expect that a Spark application that runs successfully on the CPU will also run successfully with the RAPIDS Accelerator for Apache Spark enabled. The cost of transparent acceleration is that the performance improvement any given application can expect may be difficult to predict and will be a function of several factors: what percentage of its runtime it spends in accelerable data frame operations, how much work can be performed on the GPU between CPU-only operations, and how much each accelerable operation can improve when running on the GPU. Certain kinds of queries are likely to exhibit impressive speedups with at most minor modifications, including:

- High-cardinality joins or select-distinct operations
- Analytic queries like data cube or rollup operations
- Windowed aggregates and other complex aggregations

The RAPIDS Accelerator includes tooling to analyze Spark application logs to determine whether or not an application is likely to accelerate well on the GPU, as well as functionality to explain why a given application might not be making full use of GPU acceleration. It is often possible to make minor changes to application code or particular queries that result in greater speedups by enabling more of an application to run on the GPU.

However, the appeal of transparent acceleration on unmodified code is enormous—improved velocity, cost and complexity reductions, and tighter SLAs with minimal or no engineering effort is a win for any organization. In the next section, we'll see how the RAPIDS Accelerator for Apache Spark enables speedups across a suite of standard decision-support queries.

Experimental Setup

We began with the following physical cluster:

- **Servers**—4 Dell R7525 nodes
- **CPU**—2 AMD EPYC 7452 per node (128 virtual cores per node)
- **CPU**—2 NVIDIA A100 PCIe GPUs per node
- **CPU** Memory—80GB per GPU / 160GB per node
- **System memory**—512GB per node
- **Networking**—Mellanox ConnectX-6 SmartNIC supporting 100Gb RoCE per node
- **Storage**—4x8TB NVME per node, configured as RAID Level 0

The software stack on our cluster included Cloudera Data Platform Private Cloud Base (version 7.1.7), Apache Spark (version 3.2), and version 22.02 of the RAPIDS Accelerator for Apache Spark.

With this baseline hardware configuration, we investigated multiple Spark configurations—both CPU-only and GPU-accelerated—in order to identify the best for each.

- A low-core-count CPU configuration, in which Spark used sixteen cores per CPU for executors (leaving the remaining cores free for garbage collection, data management, and other overhead),
- A high-core-count CPU configuration, in which Spark executors were able to use all sixty-four cores per CPU,
- A low-concurrency GPU configuration, in which Spark was able to use sixteen cores per CPU and two concurrent tasks on each of two GPUs per node, and
- A high-concurrency GPU configuration, in which Spark was able to use sixteen cores per CPU and either four, six, eight, ten, twelve, or sixteen concurrent tasks on each of two GPUs per node.

The best-performing CPU configuration was the low-core-count configuration, which provided the right balance between concurrency and communication. The best-performing GPU configuration was the high-concurrency configuration with ten Spark tasks per GPU.

Organizations with long-lived Spark workloads running on existing hardware will note differences between the specifications of our cluster and typical enterprise data processing cluster nodes. (Most existing Spark clusters do not feature nodes with NVMe storage, ½ TiB of RAM, and 128 CPU cores!) This has two consequences for the evaluation that follows:

1. Migrating legacy workloads to more contemporary hardware and Apache Spark 3.2 will provide dramatic performance benefits by itself; keep in mind that the performance improvements we show in the next section are in addition to the baseline improvements afforded by modernizing hardware and software, and
2. Adding GPUs to a more modest server configuration may make it possible to achieve accelerated performance at a lower price point.

The additional cost of adding GPUs to these server configurations and to Cloudera Data Platform is roughly 35%. Therefore, the bar to have improved total cost of operation is a speedup of 1.35x (i.e., we'd like to be able to do at least 35% more work over the lifetime of the cluster if we're increasing the cost by 35%). As we'll see, analytical query workloads regularly exceed this threshold.

Experimental Results

We evaluated the performance of NVIDIA GPUs and the RAPIDS Accelerator for Apache Spark on Cloudera Data Platform Private Cloud Base by running the NDS benchmark in several configurations. The NDS benchmark workload is derived from TPC-DS and consists of a Spark application that executes multiple concurrent workload streams, each of which uses Spark SQL to execute all of the TPC-DS queries in an arbitrary order. While the queries in NDS are derived from TPC-DS, NDS is not the same benchmark as TPC-DS and NDS results are not comparable to official TPC-DS results.

We chose this set of queries because they are well-understood by data practitioners, cover a broad range of analytic use cases, and allow us to compare the performance impact of various configurations for realistic data-processing use cases in Apache Spark. For these experiments, we ran two concurrent streams and took the mean execution time for each query. We then compared the mean execution times across configurations to calculate speedups:

- The GPU configurations were faster than the CPU configurations for every query in every case.
- We observed an overall workload speedup—that is, speedups over the time it took to run the entire suite of queries—of 7x when comparing the best CPU configuration we evaluated to the best GPU configuration we evaluated.
- In every pair of comparisons, between 80-90% of queries accelerated at least roughly 2x and between 30-50% of queries accelerated at least roughly 4x.
- The high-concurrency GPU configuration was faster overall than the low-concurrency GPU configuration, but not every query was faster in the high-concurrency configuration. For the purposes of this benchmark, we did not want to cherry-pick the best configuration for each individual query, but your real-world workloads will benefit from specific performance tuning just as they do today.

The following cumulative distribution plot in Figure 2 shows observed speedups when comparing the fastest CPU configuration to the fastest GPU configuration. The x-axis represents observed speedups and the y-axis represents the fraction of queries that accelerated at least as much as the value on the x-axis. For example, the plot shows that about 83% of queries accelerated at least 2x when run on the GPU configuration relative to the CPU configuration and 20% of queries accelerated just under 6x.

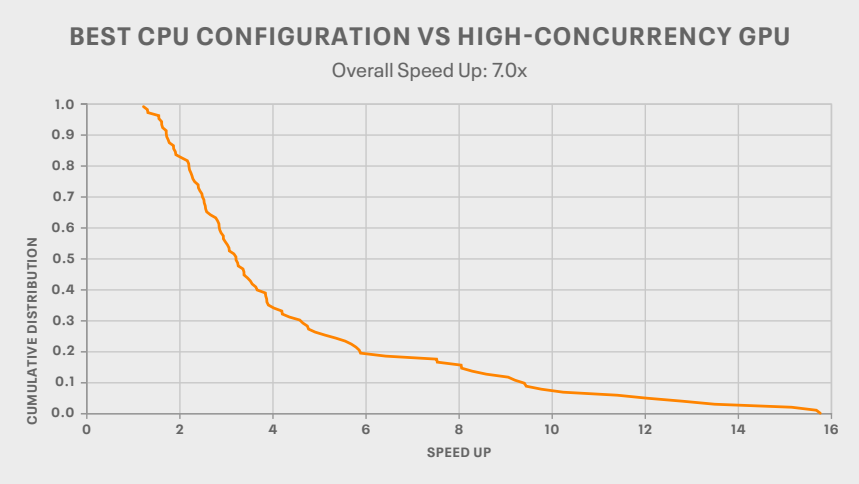


Figure 2.

Recall that some common features of analytical queries accelerate particularly well, including high-cardinality join and select-distinct operations; windowed or global aggregates; and analytical reporting and summarization operations like data cube or rollup. The queries from our set that accelerate the most combine several of these features. We'll examine the characteristics of the top five queries now:

- Query #97, which accelerates 15.8x, generates counts of a category of sales as well as total sales, and calculates the ratio of these for a particular item category and sales channel for customers in a particular time zone. It involves a full outer join between multiple subqueries featuring grouped aggregates and filtering.
- Query #82, which accelerates 15.7x, identifies items and their current prices that meet certain criteria: sold through a particular channel, produced by one of a set of manufacturers, in a given price range, and having had a certain range of inventory counts in a given calendar period. It involves filtering, grouped aggregation, and multiple ordering criteria.
- Query #93, which accelerates 15.2x, identifies the impact of a given merchandise return reason on revenue by subtracting the cost of returned items from the total cost of purchases. It involves performing an aggregate on the result of joining two subqueries, each of which involves multiple filtering criteria.
- Query #4, which accelerates 13.5x, identifies customers whose catalog spend is greater than their store spend, preferred customers, and their countries of origin. This query involves a complex filter expression applied to the union of three subqueries, each of which involves a filtered, grouped aggregate.
- Query #23 is divided into two parts, the first of which accelerates 12.1x and the second of which accelerates 12.8x. The first part of query #23 identifies items sold frequently in a given range of years through a given channel, the maximum store-channel sales for any customer in that period, and the customers who are in the top 5% of store-sales customers in that period. The second part of query #23 uses the information from the first part to compute the total web- and catalog-channel sales made by those customers in a given month. Both parts involve multiple subqueries with filtering, aggregation, and set operations.

About Cloudera
At Cloudera, we believe that data can make what is impossible today, possible tomorrow. We empower people to transform complex data into clear and actionable insights. Cloudera delivers an enterprise data cloud for any data, anywhere, from the Edge to AI. Powered by the relentless innovation of the open source community, Cloudera advances digital transformation for the world's largest enterprises.

Learn more at cloudera.com

Connect with Cloudera
About Cloudera:
cloudera.com/more/about.html

Read our Blog:
blog.cloudera.com

Follow us on Twitter:
twitter.com/cloudera

Visit us on Facebook:
facebook.com/cloudera

See us on YouTube:
youtube.com/c/ClouderaInc

Join the Cloudera Community:
community.cloudera.com

Read about our customers' successes:
cloudera.com/more/customers.html

Key Takeaways

The RAPIDS Accelerator for Apache Spark and NVIDIA GPUs make it possible to transparently accelerate Spark data frame workloads on Cloudera Data Platform. This acceleration can mean many things for your organization: improved throughput, solving problems at scale without sampling, scaling your capabilities without enlarging your datacenter footprint, and enabling analysts to ask complex questions at interactive latencies.

A wide range of analytical queries, ETL workloads, and data engineering jobs can realize incredible performance improvements. We saw an absolute speedup of 7x by adding GPUs to a cluster executing a suite of analytical queries and speedups on individual queries of up to nearly 16x. Given that GPU hardware adds 35% to the cost of the cluster, this 7x speedup translates into a 5.2x reduction in total cost of operation for the suite of queries. Finally, these categories of workload often overlap: for example, the results of analytical queries can be used to enrich records during ETL and data engineering pipelines often depend on both analytical queries and federation workloads.

Every one of the 103 queries in our benchmark suite saw an absolute performance improvement, and all but three queries represented improved cost-adjusted performance (i.e., greater than 1.35x speedup) or reduced total cost of operation. Of these three queries, one accelerated 1.23x, one accelerated 1.32x, and one accelerated 1.33x.

Customers whose workload mix is dominated by complex ETL or reporting jobs or demanding analytical queries may see even greater overall speedups or cost reductions; we've linked such a case study below.

Learn More

- Learn about Cloudera Data Platform and talk to your Cloudera account executive to learn more about accelerating Spark jobs by expanding your cluster with NVIDIA-Certified Systems.
- Read about a real-world success story where a public sector customer achieved a 20x speedup and a 50% cost reduction by using Cloudera Data Platform and the RAPIDS Accelerator for Apache Spark for an advanced analytic workload.
- See how the RAPIDS Accelerator for Apache Spark can accelerate data engineering, data preparation, and on-line analytic processing in this ebook.