

Dynamic Query Optimization through RapidsDB Federation Connectors

Technical White Paper

(March, 2022)



Table of Contents

1. Introduction	2
2. SQL statements	4
3. Dynamic Query Optimization and Compilation	4
3.1 Query Parsing and Basic Plan Generation	4
3.2 Predicate Push-Down	5
3.3 Adaptive Condensing	7
3.4 Query Execution	8
3.5 Query Plan Distribution	9
3.6 Code Generation and Compilation	10
3.7 Just-In-Time Native Code Compilation	10
4. Metadata Optimization for Heterogeneous Data	10
5. Summary	11

1. Introduction

RapidsDB is a fully distributed, ANSI-compliant, MPP, in-memory federated query system that is designed to support complex analytical SQL queries. It has an advanced, rules-based and cost-based SQL Compiler and Optimizer that is responsible for taking a user's SQL query and building the optimum query execution plan for that query.

The process of analyzing the query and its data to determine the best plan is called query optimization. Query optimization often involves rearranging parts of a query plan. The order in which the various operations occur can be very significant for query performance. The best order depends on the size and makeup of the data involved. Query optimization is a common feature of many relational database management systems (RDBMS). As a federating query system, RapidsDB may be called upon to integrate data from multiple heterogeneous sources in a single query. As a result, this makes a query optimization process different than most database management systems (DBMS).

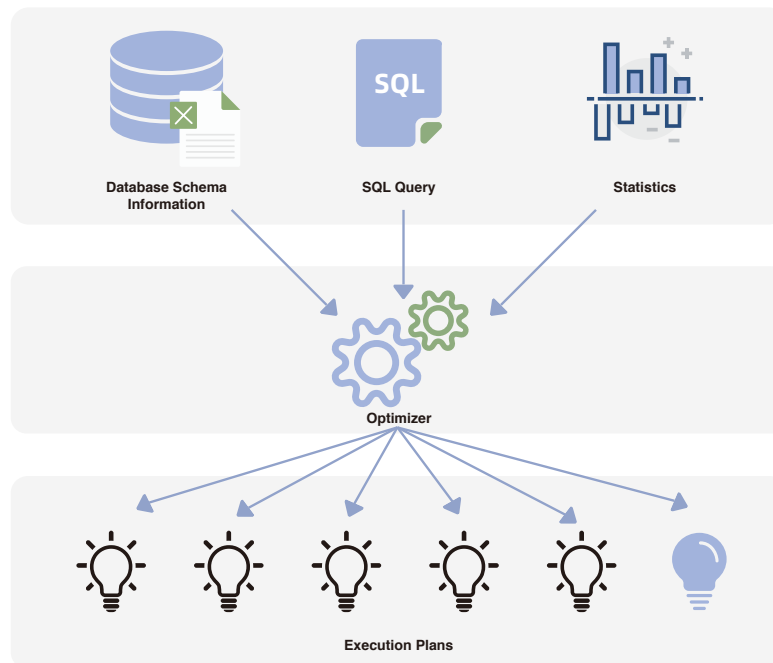


Fig.1 Query Optimization of Most DBMS

RapidsDB Federation Connectors are deeply integrated into the query optimization process, allowing them to guide optimization decisions based on backend data statistics. Connectors are also able to control how work is delegated to underlying data systems using the RapidsDB "Adaptive Condensing" model (see below). This allows RapidsDB to take maximum advantage of the capabilities of federated data sources.

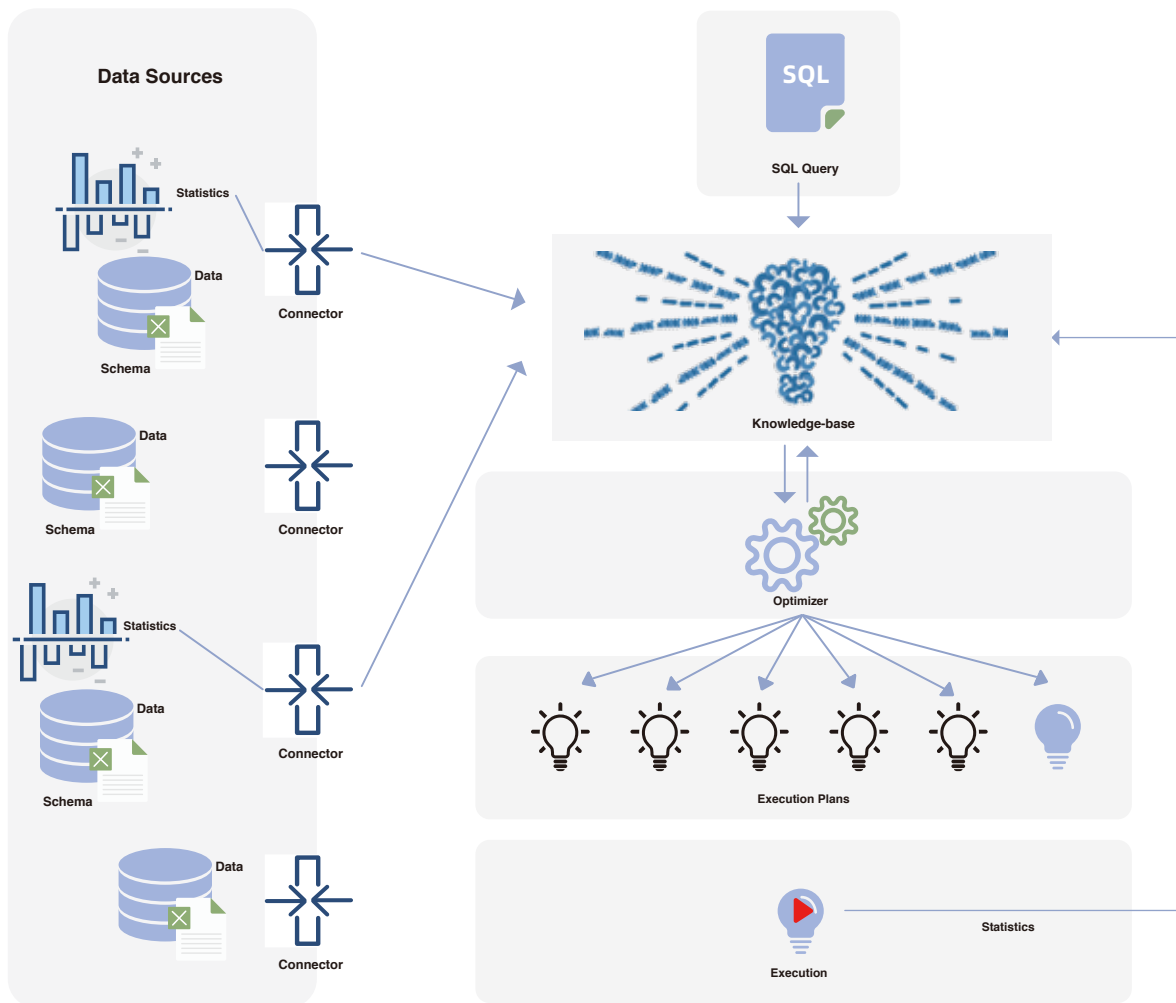


Fig.2 RapidsDB Query System

2 . SQL statements

A SQL statement is a series of words (keywords, table names, column names, etc.) describing a database operation. RapidsDB accepts several types of SQL statements, including Data Definition Language (DDL) statements and Data Manipulation Language (DML) statements.

DDL statements allow users to define the table structures and data types used to organize the data in a database. CREATE TABLE is a common DDL statement. Once users have defined the structures and filled them with data, they use DML statements to manipulate and query the data. The SELECT statement, used to query data, is the most common DML statement.

3 . Dynamic Query Optimization and Compilation

The SELECT statement allows a user to specify a query in terms of the results the user wishes to produce. For the most part, the user is not expected to say how the query should be accomplished. It is the job of the query planner to determine the best way to accomplish the query. This is a complex process consisting of several phases.

3.1 Query Parsing and Basic Plan Generation

When a user enters a SELECT statement into RapidsDB, the RapidsDB query planner first parses the statement to validate its syntax and identify the tables and columns involved. Because RapidsDB is a federating query system, a single query may involve tables from several different underlying data sources. The query planner identifies the source of each table, verifies all column names and checks that the requested operations are valid.

After parsing and validating the query, RapidsDB decomposes it into simple building blocks such as reading a table, filtering rows, joining, grouping, sorting, etc. These building blocks are organized internally in a tree-shaped data structure describing the operations to be performed and the flow of data between them. The result is a sort of block diagram for how to accomplish the query. This is called the query plan.

In general, an initial or “naïve” query plan is usually derived in a straightforward way from the SELECT statement of a SQL query. A given query, however, can often be accomplished in numerous different ways, which can take different processing time from a millisecond to hours. Therefore, a naïve plan may be less than ideal.

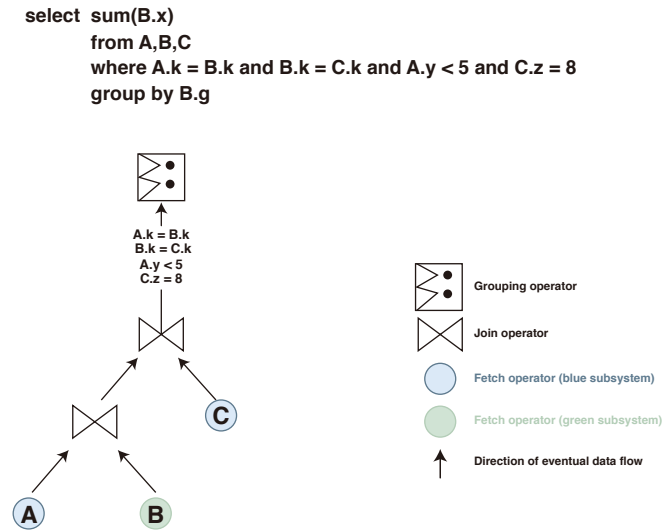


Fig.3 Basic/Naïve Query Plan

For example, in the above query graph, we assume that data source A contains one table of 100 rows. Data source B and data source C each contains one table of 1,000 rows. The total number of rows being fetched from the joins of A, B and C into the predicates would be a massive number of 100 million rows (100 rows of A x 1,000 rows of B x 1,000 rows of C)!

3.2 Predicate Push-Down

In the SQL query example above, the SELECT statement includes several predicates. The RapidsDB SQL Compiler and Optimizer will analyze all these predicates and determine which ones can be pushed down to the underlying data source. This process is called predicate push-down. The goal for the operation is to get the predicates closer to the underlying data source and filter the data at an earlier point so that a query can be executed more efficiently.

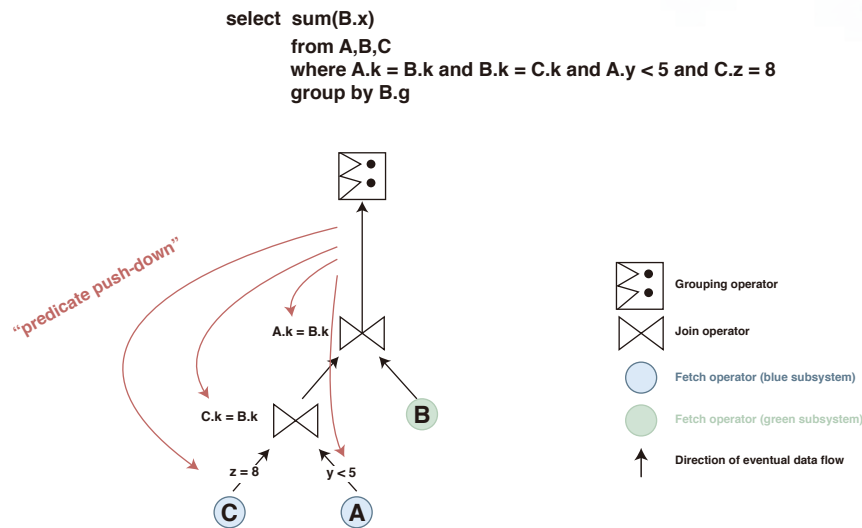


Fig.4 Basic Optimization

Once these predicates are pushed down, it will greatly reduce the rows fetched from each data source. For example, in Fig.4, after the predicate of $z=8$ is pushed down to and applied at data source C, the number of returned rows might be cut down from 1,000 rows to 500 rows. For $y < 5$, the number of returned rows from data source A might be cut down from 100 rows to only 1 to 2 rows. Now instead of 100,000 rows, only 500 or 1,000 rows will be fetched for the join result of A and C. After the query plan takes the join predicates of $C.K = B.K$ and $A.K = B.K$, the number of rows that come out of the joins of A, B and C will further be reduced, which makes the joins much more efficient.

Predicate pushdown is a rule-based optimization. RapidsDB implements a number of valuable rule-based optimizations as well as cost-based optimizations. Cost-based optimizations are very prevalent in modern query systems. They enable a query plan to be configured in some way that will be less expensive or more performant by reordering the operators and rearranging the query plan.

The above query graph also shows the re-order of the joins. In the query graph of Fig.3, A and B are joined first. And then the join result of A and B is joined to C. When these are normal inner joins, they can actually be re-ordered so that the predicates can be leveraged at an earlier point to dramatically cut down the number of rows of the join result. In this example, first of all, because A and C both have predicates on them, it means that the join of A and C is probably going to return fewer rows than the join of A and B. Secondly, as A and C are both coming from the blue database system, they can be put together to offer some condensing opportunities when the query plan goes to the next stage.

3.3 Adaptive Condensing

What makes RapidsDB unique is that it has an additional optimization phase, the condensing phase, that is not found in most systems. The condensing phase is a part of the dynamic optimization process. In this phase, elements of the query plan associated with a particular Connector are coalesced into a special element called a "condenser." The condenser accepts or rejects plan elements under control of the Connector, based on the capabilities and efficiencies of the associated data source. As the condensers are closely intertwined with the Connectors, it makes the RapidsDB Federation Connectors super powerful by actively participating in the execution of a query plan. Systems with simple data import APIs or wrappers usually simply pull data from one data source. But in RapidsDB, each Connector communicates with its associated data source and checks whether the underlying system is capable of taking on some of the workload.

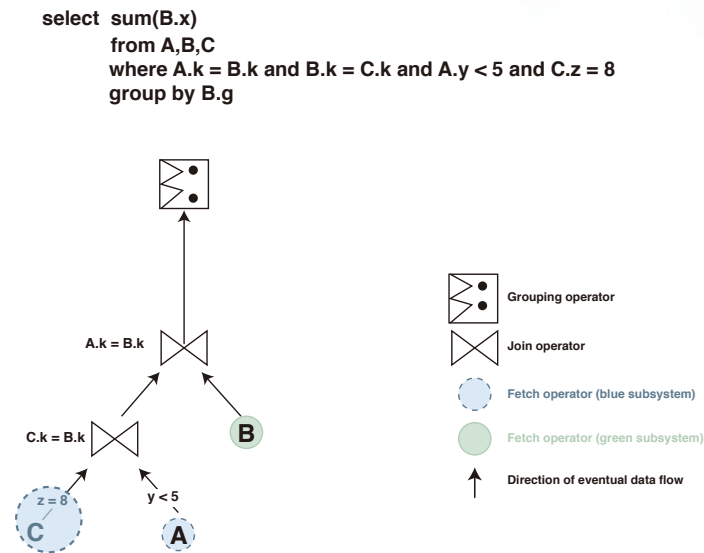


Fig.5 Condensing

In this example, the Connector associated with data source C will check whether C can filter $z=8$. If the answer is yes, the condenser will not only fetch data from data source C but also perform the filtering operation. If the answer is no, which means data source C can only supply the raw data without the ability to condense, then the data will be fetched from data source C and processed in the RapidsDB Execution Engine. This approach of having the underlying data source perform as much of the query as possible tremendously reduces the amount of data that has to be transferred over a network and boosts the performance of the database.

3.4 Query Execution

When a user query is executed, each condenser is responsible for accomplishing the work of the query elements it has accepted. It will generate an equivalent query to the underlying data source, adapting to the data source in terms of syntactical rules, data types, operational functions, etc. All predicates, joins or any other operations will be condensed into one single operation and pushed down to the associated underlying data source. As some data sources are capable of performing some or all of the work of a query independently of RapidsDB, it increases query performance by reducing the amount of network transmission needed. Many data sources are capable of filtering

data at the source. Some RDBMS systems, such as Oracle or Postgres, are even able to aggregate or join data at the source. In Fig.6 below, B is a simple fetch operator with no predicates. The C and A fetch operators and associated predicates have all been condensed into a single operation, which is pushed down to the blue data source as a SQL statement (in the SQL dialect appropriate for that data source). The blue data source will filter and join the data for C and A before sending the result over the network to the RapidsDB engine. The result of this pushed-down join will in turn be joined with the data from B. The RapidsDB Execution Engine executes this second join and completes the query by performing the Group By operation.

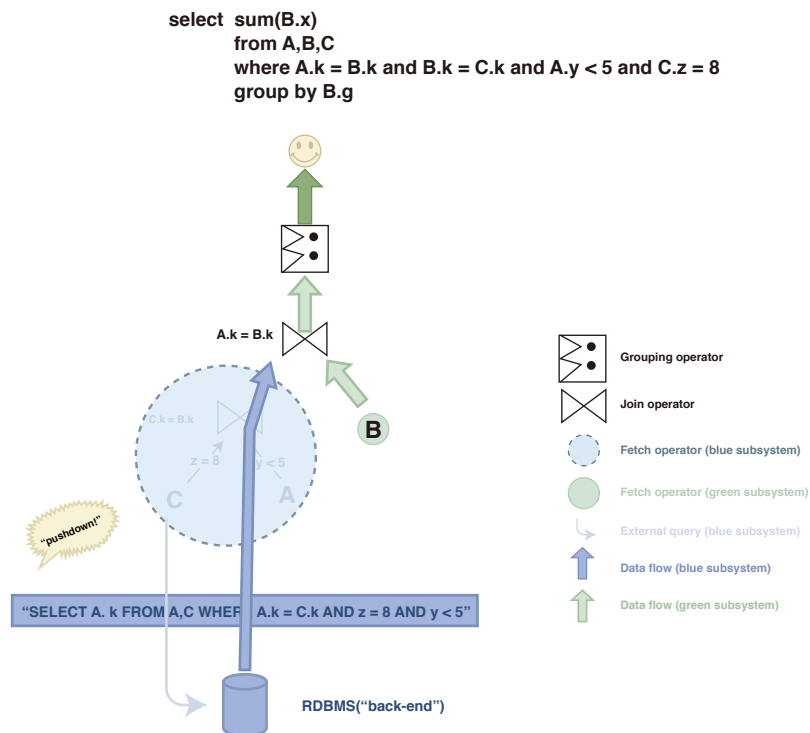


Fig.6 Condensing

3.5 Query Plan Distribution

Since RapidsDB is a distributed query engine, query plans typically run in parallel on multiple computers. Once the query plan is finalized, it is broken into fragments which are sent over the network to the computers in the RapidsDB cluster.

Different parts of the query plan will run on different computers, based on which data Connectors are involved and where the relevant data is located. The RapidsDB execution system works to distribute each plan fragment to the most efficient location. In particular, it strives to run fragments at the same location as the data they will use.

3.6 Code Generation and Compilation

Once all the parts of the query plan reach their run locations, they are ready for execution. RapidsDB uses a compilation-based execution model. This produces higher performance than systems that directly interpret query plans.

The RapidsDB query compiler processes the elements of the query plan and generates a program to perform the query. The program is generated in the form of Java “bytecode,” a special internal language of the Java Virtual Machine (JVM) used by the RapidsDB execution engine.

Bytecode is generated for query operations to be performed directly in the RapidsDB execution engine, along with calls to library functions and User Defined Functions (UDFs) and also calls to the appropriate data Connectors for any parts of the query being pushed down to the underlying data sources. Once all code is generated and the various parts of it are linked together, it is submitted to the JVM for execution.

3.7 Just-In-Time Native Code Compilation

As execution of the query proceeds, the JVM selectively compiles portions of the bytecode into the native instruction set for the CPU. The most heavily used parts of the query program are further

optimized based on the actual flow of data through the query. The result is a highly optimized custom native code program for accomplishing the query specified by the user's SELECT statement.

4 . Metadata Optimization for Heterogeneous Data

In general, optimizers of most traditional data systems have statistics of their data such as the number of rows in a table or the distribution of values in various columns. These statistics are very important to cost estimation. RapidsDB is a federated query system across multiple data sources, which means it may or may not be able to obtain statistics from some of the underlying data sources. For example, for a RDBMS, it might have lots of statistics and information available for data and tables. But a streaming data source most likely will not be able to provide any statistics. The RapidsDB dynamic optimizer design generates optimization metadata independently by observing and analyzing the execution of queries. In some cases, queries may be augmented with extra operations that measure data metrics as queries are executed.

RapidsDB maintains an internal knowledgebase to record information about queries and data, whether supplied by the underlying data sources or gathered by the RapidsDB Federation Connectors during query execution. This knowledgebase helps the RapidsDB Optimizer to make optimization decisions. When useful metrics or statistics are lacking, the RapidsDB Optimizer can learn them on the fly and make educated guesses about queries based on past experience on related data or similar queries.

5 . Summary

RapidsDB's dynamic query optimization provides a lightweight but powerful dynamic processing framework, which is adaptable to a very wide and extensible range of data sources. It allows the RapidsDB Federation Connectors to fully participate in the optimization process and enables a federated query to use different databases and schemas from entirely different systems in the same SQL statement.

Since a single query may involve multiple data sources, the RapidsDB Optimizer works interactively with the Connectors to determine which operations from the plan should be pushed down to which data source. Operations for a given Connector are condensed into a single building block, for which the Connector and data source will be responsible. The remaining parts of the query plan will be performed by the RapidsDB execution engine.

The knowledge-based model empowers Connectors to guide the dynamic optimization process based on the different capabilities of the underlying data sources. It helps Connectors gather heterogeneous statistics by dynamically inserting instrumentation into a condensed query, which then can be pushed down to the related data source.


RapidsDB allows the user to view a collection of disparate data sources as a single database, performing standard SQL queries against this federated view. The automated query optimization process works to find the best way to process a given query, parceling out work to the different data sources and efficiently integrating the results. This approach abstracts away the complexity of the data preparation pipelines so that users can focus more on analyzing data to solve business problems instead of spending tremendous amounts of time simply preparing data or rearranging queries manually to adjust to different systems and maximize performance. It makes the integration of heterogeneous data more efficient and agile.

Contact Us

Hong Kong Headquarters:

 Boray Data Technology Limited


 Address: Flat A, 12/F, MW Tower II, 5 Kimberley Street, Tsim Sha Tsui, Kowloon, Hong Kong


 Phone: 1-303-731-2699

 Email: info@rapidsdata.com

China Headquarters:

 Borruai Data Technology (Beijing) Co. Ltd.


 Address: 601B/F, Building 19(T1), Poly International Plaza, Zone 7, Wangjing East Park, Chaoyang District, Beijing, China

 Phone: 86-010-64700868

 Email: market@boraydata.com


US Lab:


 Address: 6253 Goddess Ct., San Jose, CA 95129, USA

 Phone: 1-303-731-2699

 Email: info@rapidsdata.com

Asean Regional Office (Singapore, Malaysia, Indonesia, Thailand, Philippines, Vietnam):

 Address: 7 Temasek Boulevard #12-07 Suntec Tower One, Singapore 038987

 Phone: 65-90015111

 Email: yylai@rapidsdb.sg

Dynamic Query Optimization
through RapidsDB Federation Connectors



RapidsDB

POWERED BY BORAY DATA

For more information, please contact : info@rapidsdata.com , www.rapidsdb.com



@RapidsDB

Copyright © 2022 Boray Data Technology Ltd. All rights reserved.