Instituto Tecnologico de Costa Rica

Sede Central de Cartago

Escuela de Computación

Resumen 5 y 6 (R5 y R6)

IC-4302 Bases De Datos II GR 1

Estudiantes: Deyan Sanabria Fallas #2021046131

Profesor: Gerardo Nereo Campos Araya

Fecha de Entrega: 25/10/2022

Segundo Semestre 2022

Indice

Introduction

Spanner started out as a key-value store offering multirow transactions, external consistency, and transparent failover across datacenters. It has evolved into a relational database system. Features like a SQL Query processor were "bolted on", because of the demand of this type of features, thus system was forced to evolve to make it behave more like a traditonal database.

- Architercture of the distributed storage stack forced changes in query compilation and execution
- Demands of the query processor forced changes in the way data was store and manage.

Spanner Query Processor Implements a dialect of SQL called Standard SQL used by other google products. Spanner query processor is built with a mix of transactional and analytical workloads, and support both low-latency and long-running queries.

Background

- Spanner is a sharded, geo-replicated relational database system. A database is horizontally row-range sharded. Within a data center, shards are distributed across multiple servers. Shards are then replicated to multiple, geographically separated datacenters.
- Shard boundaries are specified as ranges of key prefixes and preserve row co-location of interleaved child tables.
- Spanner’s transactions use a replicated write-ahead redo log, and the Paxos consensus algorithm is used to get replicas to agree on the contents of each log entry. Log records are only replicated.
- Concurrency control uses a combination of pessimistic locking and timestamps.
- Spanner provides a special RPC framework, called the coprocessor framework, to hide much of the complexity of locating data.
- A given replica stores data in an append-only distributed filesystem called Colossus.
- The Spanner query compiler first transforms an input query into a relational algebra tree. the query compiler extensively uses correlated

join operators, both for algebraic transformations and as physical operators.

- Both the compiler and runtime attempt to aggressively perform partition pruning to minimize the amount of work done by queries by restricting the data they operate on without changing results.

Query Distribution

Executes code in parallel on multiple machines hosting the data to server both online and long running queries.

Distributed Query Compilation

The Spanner SQL query compiler uses the traditional approach of building a relational algebra operator tree and optimizing it using equivalent rewrites.

- A Distributed Union operator is inserted immediately above every Spanner table in the ralational algebra. where possible, query tree transformations pull Distributed Union up the tree to push the maximum amount of computation down to the servers responsible for the data shards.

- Spanner supports table interleaving, a mechanism of co-locating rows from multiple tables sharing a prefix of primary keys. Spanner also pushes down operations that potentially take as input rows from multiple shards.

Distributed Execution

Distributed Union minimizes latency by using the Spanner coprocessor framework to route a subquery request addressed to a shard to one of the nearest replicas that can serve the request. Before making a remote call to execute its subquery, Distributed Union performs a runtime analysis of the sharding key filter expression to extract a set of sharding key ranges.

- When the sharding key range extracted from the filter expression covers the entire table, Distributed Union may change its subquery evaluation strategy

- Distributed Union reduces latency by dispatching subqueries to each shard or group in parallel. Distributed Union may detect that the target shards are hosted locally on the server and can avoid making the remote call by executing its subquery locally.

Distributed Joins

Another important topic in query distribution are joins between independently distributed tables. Spanner implements a Distributed Apply operator by extending Distributed Union and implementing Apply style join in a batched manner – as two joins, a distributed join that applies batches of rows from the input to remote subquery, and another that applies rows from each batch to the original join's subquery locally on a shard.

- Distributed Apply allows Spanner to minimize the number of cross-machine calls for key-based joins and parallelize the execution.
- Distributed Apply performs shard pruning using the following steps:
    1. Evaluate the sharding key filter expression for each row of the batch using column values from each row.
    2. Merge the sharding key ranges for all rows from (1) into asingle set.
    3. Compute the minimal set of shards to send the batch to by intersecting the sharding keys ranges from (2) with the shard boundaries.
    4. Construct a minimal batch for each shard by intersecting the sharding key ranges from (2) with the shard boundaries.

Query Distribution APis

Spanner exposes two kinds of APIs for issuing queries and consuming results.

- The single-consumer API is used when a single client process consumes the results of a query.
- The parallel-consumer API is used for consuming query results in parallel from multiple processes usually running on multiple machines.

Query Range Extraction

Query range extraction refers to the process of analyzing a query and determining what portions of tables are referenced by the query. The referenced row ranges are expressed as intervals of primary key values. Spanner employs several flavors of range extraction:

- Distribution range extraction: Knowing what table shards are referenced by the query is essential for routing the query to the servers hosting those shards.
- Seek range extraction: Once the query arrives at a Spanner server, we figure out what fragments of the relevant shard to read from the underlying storage stack.

- Lock range extraction: In this case, the extracted key ranges determine what fragments of the table are to be locked, or checked for potential pending modifications.

Compile-time rewriting

Our implementation of range extraction in Spanner relies on two main techniques:

- At compile time, a filtered scan expression is normalized and rewrited into a tree of correlated self-joins that extract the ranges for successive key columns.
- At runtime, a special data structure is used to call a filter tree for both computing the ranges via bottom-up interval arithmetic and for efficient evaluation of postfiltering conditions.

Filter tree

The filter tree is a runtime data structure that is simultaneously used for extracting the key ranges via bottom-up intersection / union of intervals, and for post-filtering the rows emitted by the correlated self-joins.

Query Restarts

Spanner automatically compensates for failures, resharding, and binary rollouts, affecting request latencies in a minimal way. The client library transparently resumes execution of snapshot queries if transient errors or other system events occur, even when partial results have been consumed by user's code.

Usage scenarios and benefits

- Hiding transient failures.
- Simpler programming model: no retry loops.
- Streaming pagination through query results.
- Improved tail latency for online requests.
- Forward progress for long-running queries.
- Recurrent rolling upgrades.

- Simpler Spanner internal error handling.

Contract and requirements

The restart implementation has to overcome the following challenges:

- Dynamic resharding
- Non-determinisim
- Restarts across server versions

Common SQL Dialect

Spanner's SQL engine shares a common SQL dialect, called "Standard SQL", with several other systems at Google including internal systems such as F1 and Dremel (among others), and external systems such as BigQuery. The effort to standardize implementations both covers gaps in the SQL standard where details are unclear or left to the implementation and covers Google specific extensions. To ensure consistency between systems, several shared components were built as a collaboration between teams, referred to internally as the GoogleSQL library.

Blockwise-Columnar Storage

Spanner originally used significant parts of the Bigtable code base, in particular the on-disk data format and block cache. It is self-describing and therefore highly redundant, and traversal of individual columns within the same locality group is particularly inefficient.

Because Spanner scales so well horizontally, SSTables have proven to be remarkably robust even when used for schematized data consisting largely of small values, often traversed by column. But they are ultimately a poor fit and leave a lot of performance on the table. Ressi is the new low-level storage format for Spanner, which will be entering production in the first half of 2017.

As with SSTables, Ressi stores a database as an LSM tree, whose layers are periodically compacted. Within each layer, Ressi organizes data into blocks in row-major order, but lays out the data within a block in column-major order (essentially, the PAX layout).