Lx-DB

Architecture

Table of Contents

1. Architecture

Lx-DB is a distributed database. It can scale transactions from 1 node to thousands. This is achieved by following a number of basic principles, having a scalable architecture, and through intelligent implementation of the various database concepts.

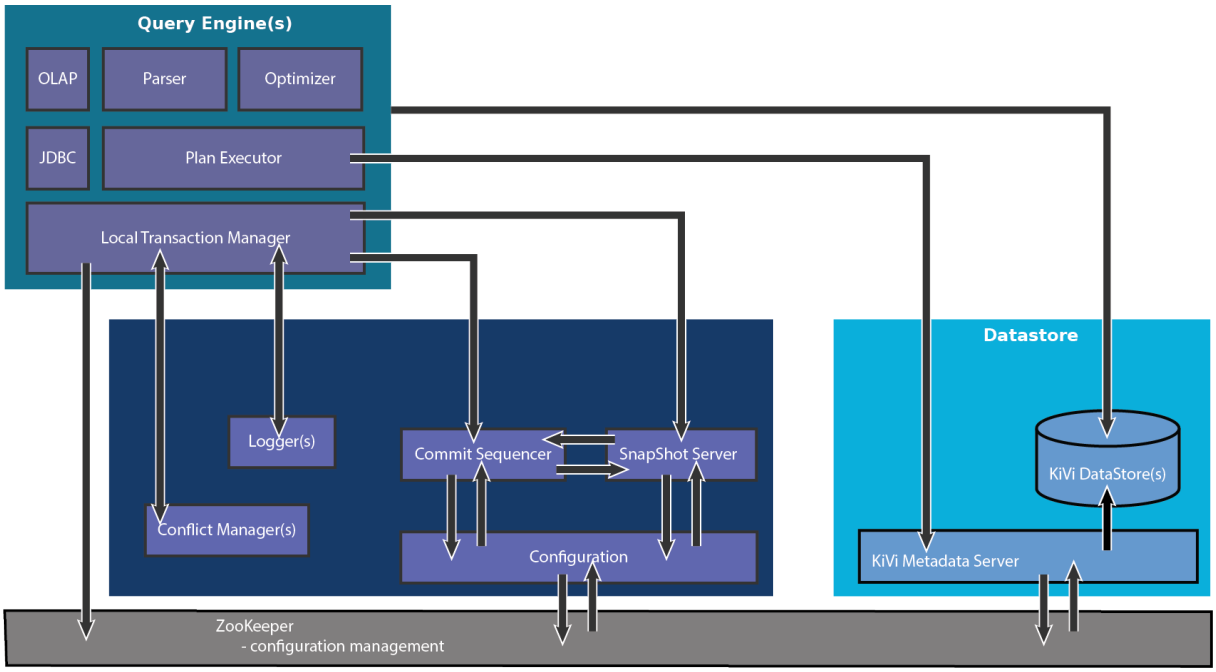
2. Principles

The basic principles for Lx-DB transaction management are the following:

- Separation of commit from the visibility of committed data
- Proactive pre-assignment of commit timestamps to committing transactions
- Detection and resolution of conflicts before commit
- Transactions can commit in parallel due to:
    - They do not conflict
    - They have their commit timestamp already assigned that will determine its serialization order
    - Visibility is regulated separately to guarantee the reading of fully consistent states

ACID transactions can scale because they are decomposed in different components that can be distributed and scaled independently.

3. Components



Lx-DB is made up of the following components:

- Query engine
  - OLAP Workers
  - JDBC
  - Parser
  - Optimizer
  - Plan Executor
  - Local Transaction Manager
- Transaction Manager
  - Commit Sequencer
  - Snapshot Server
  - Config Manager
  - Conflict Managers
  - Loggers
- KiVi Data Store
  - KiVi Metadata Server
- Zookeeper

- Distributed configuration management

3.1. Query Engine

The Query Engine transforms SQL queries into a query plan that is a tree of executable algebraic query operators over the datastore and that moves data across operators till finally producing the overall result requested by the original SQL statement.

3.2. JDBC

The JDBC API provides standardized connection and query execution for the query engine.

3.2.1. Parser

The SQL parser builds an execution plan from the queries.

3.2.2. Optimizer

The SQL optimizer that transforms the execution plan for efficiency.

3.2.3. Plan Executor

The plan executor is an SQL query engine that orchestrates the execution of the query plan.

3.2.4. OLAP Workers

For analytical query operators, the query engine can parallelize the query operators leveraging OLAP workers in the different Query Engine components. This way, analytical queries are parallelized and results can be gotten faster and more efficiently.

3.2.5. Local Transaction Manager

The local transaction manager is in charge of the life cycle of transactions and interfacing between the client side of the storage engine and the other transactional manager components.

3.3. Transaction Management

The transaction manager is in charge of guaranteeing the coherence of the operational data in the advent of failures and concurrent accesses. It provides abstracted transactions that enable

to bracket a set of data operations to request that they are atomic.

The transaction manager layer is composed of several components.

3.3.1. Loggers

The loggers are components that log all transaction updates (called writesets) to persistent storage to guarantee durability of transactions. The logger subsystem is not built from a single logger process, but logger can be distributed in different processes to achieve better performance.

Each logger takes care of a fraction of the log records. Loggers log in parallel and are uncoordinated. There can be as many loggers as needed to provide the necessary IO bandwidth to log the rate of updates.

Loggers can also be replicated.

3.3.2. Conflict Managers

Conflict Managers are in charge of detecting write-write conflicts among concurrent transactions. By detecting these conflicts the LTMs can abort transactions that cannot enforce write isolation.

Each conflict manager takes care of a set of keys and there can be as many conflict managers as needed. They scale in the same way as hashing based key-value data stores

3.3.3. Commit Sequencer

The commit sequencer is in charge of distributing commit timestamps to the Local Transactional Managers.

3.3.4. Snapshot Server

The Snapshot Sever provides the most fresh coherent snapshot on which new transactions can be started.

3.3.5. Configuration Manager

The configuration manager handles system configuration and deployment information. It also monitors the other components.

3.4. DataStore

The data store is fully ACID, highly efficient Relational Key-Value datastore, which also supports columnar storage.

3.4.1. KiVi Data Stores

KiVi data stores supports secondary local indexes and performs operations taking advantage of modern CPU’s vectorial SIMD operations. They are designed to get the most of current multi-core and NUMA architectures, and implement a novel data structure that combines the advantages of B+ Trees for range queries and of LSM-Trees for random updates and inserts.

3.4.2. KiVi Metadata Server

This is the component that keeps the metadata information and coordination among different KiVi Data Stores.

3.5. ZooKeeper

Apache Zookeper is a centralized replicated service for maintaining configuration information and providing distributed synchronization and group services.

LeanXcale data management uses ZooKeeper for coordination among components and to keep configuration information and the health status of each of them.