StorageLayoutGuidelines

Storage Layout Guidelines

The purpose of this page is to provide useful information on layout strategies for new users (of which I am). It is a work in progress. When it is ready, I'll request it be linked in to the appropriate place. Until then, please add or make suggestions as usual.

This document is meant to provide some of the design-oriented help that new users tend to need. The goal is not to impose any particular storage strategy for any type of problem, but to suggest possible ways of dealing with typical examples in a conceptually portable way. When users have a firm understanding of their application domain, this page should help them design a plausible storage layout from a set of (hopefully multiple) options while taking into consideration some of the more important trade-offs. The best storage layout for a given application may not always be obvious. Understanding the possible storage layouts and their trade-offs is key.

Examples

Examples of data models from the real world, useful for framing the rest of the guide:

- Object Modeling
  - objects with named properties
  - collections of objects with named properties
  - objects with named collections of objects with named properties
- Problem Domains
  - Non-Key Indexes
  - Temporal Data
  - Relations

The Cassandra Coordinate System

One useful way to think of the storage model of Cassandra is as a collection of coordinate grids (column families), where you choose the interpretation of the axes and grid intersections (row and column). Each grid (column family) is named, and lives within a named keyspace.

One of the most useful aspects of Cassandra is the fact that the columns which are present for a given row are completely independent of the other rows. As far as Cassandra is concerned, a column is defined for a given row only if the column is present. There is no true Cartesian product of all columns for all rows in the traditional sense. This structure logically resembles a list of a list, as opposed to a multi-dimensional array, where elements are sometimes arranged in a region of storage sized by the product of all dimensions. Cassandra addresses this by organizing the storage internally as a sparse map.

Some named grids are actually "3D" in nature, since an additional dimension is added between the row and columns in the grid. These are called SuperColumns.

The semantics of what a grid represents are entirely unrestricted. This means that you can decide what "row" and "column" mean within your design. However, to make ranges and sets meaningful to an application, you have to tell Cassandra how to interpret the x-axis. You can build your own, or use one of the built-in settings: Bytes, Ascii, UTF8, Long, LexicalUUID, or TimeUUID. Providing this to Cassandra means that hashing and ranging operations along the x-axis (columns) will work as expected.

Mapping Data Domains to Cassandra

This section will strive to bring the last two sections together. It will provide possible mappings between the examples and a usable Cassandra layout, with some of the trade-offs taken into consideration.

Performance Constraints and Costs

Loose estimates of some of the common building blocks.

- Size of column data
- Number of columns
- Type of access
- Single Row, Single Column
- Single Row, Multiple Columns, by range
- Single Row, Multiple Columns, by subset
- Multiple Rows, Multiple Columns, by range
- Multiple Rows, Multiple Columns, by set
  ** ranged ** column sets
- number of columns

Layout Strategies

Key Addressing

Cluster distribution of rows within a family is controlled by the row key. This suggests implicitly that the row serves as an optimal work unit within the application. When your application wants to get or put data some cohesive data, accessing it by a single logical key will be faster than if you have to scan across keys. When you have to scan across keys, Cassandra might need to do additional work. Although it may still be fast, it may be an unnecessary cost to pay in the absence of other requirements. As a scaling factor, this may be significant.

Organize logical units of data to be addressed by a single key when requirements allow.