Chapter No. 1
"An Overview of Architecture and Data Modeling in Cassandra"
In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.1 "An Overview of Architecture and Data Modeling in Cassandra"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Sanjay Sharma has been the architect of enterprise-grade solutions in the software industry for around 15 years and using Big Data and Cloud technologies over the past four to five years to solve complex business problems. He has extensive experience with cardinal technologies, including Cassandra, Hadoop, Hive, MongoDB, MPP DW, and Java/J2EE/SOA, which allowed him to pioneer the LinkedIn group, Hadoop India. Over the years, he has also played a pivotal role in many industries, including healthcare, finance, CRM, manufacturing, and banking/insurance. Sanjay is highly venerated for his technological insight and is invited to speak regularly at Big Data, Cloud, and Agile events. He is also an active contributor to open source.

I would like to thank my employer, Impetus and iLabs, and its R&D department, which invests in cutting-edge technologies. This has allowed me to become a pioneer in mastering Cassandra- and Hadoop-like technologies early on.

But, most importantly, I want to acknowledge my family, my beautiful wife and son, who have always supported and encouraged me in all my endeavors in life.

For More Information:

www.packtpub.com/cassandra-design-patterns/book
Cassandra Design Patterns

Big Data has already become a buzzword in today's IT world and provides lots of choices for end users. Cassandra is one of the most popular Big Data technologies today and is used as a NoSQL data store to build web-scale applications in the Big Data world.

The idea behind this book is for Cassandra users to learn about Cassandra's strengths and weaknesses and, more importantly, understand where and how to use Cassandra correctly, so as to use its strengths properly and overcome its weaknesses.

One of the most critical decisions taken while writing the book was using the term design pattern to explain Cassandra's where and how usages. Traditionally, design patterns are linked to object-oriented design patterns made popular by Gang of Four (GoF).

However, in the context of this book, patterns refer to general, reusable solutions to commonly recurring software problems and will be used interchangeably for use case patterns, design patterns, execution patterns, implementation strategy patterns, and even applied design patterns.

What This Book Covers

Chapter 1, An Overview of Architecture and Data Modeling in Cassandra, discusses the history and origins of Cassandra. In this chapter, we understand the parentage of Cassandra in Amazon Dynamo and Google BigTable and how Cassandra inherits the best of the abilities as well as evolves some of its own. The importance of reading this chapter lies in the fact that all the subsequent chapters refer to the capabilities mentioned in this chapter that can be used for solving various business use cases in the real world.

Chapter 2, An Overview of Case and Design Patterns, offers an overview of all the patterns that will be covered in this book.

Chapter 3, 3V Patterns, covers the known 3V or Volume, Velocity, and Variety model associated with Big Data and also how Cassandra fulfils all the requirements for building Big Data applications facing the 3V challenge.

For More Information:
www.packtpub.com/cassandra-design-patterns/book
Chapter 4, Core Cassandra Patterns, describes how Cassandra can be used to solve some interesting business problems due to its unique capabilities covering high availability, wide rows, and counter columns.

Chapter 5, Search and Analytics Applied Use Case Patterns, takes the next step and goes beyond using only Cassandra to solve some interesting real-world use cases. This chapter covers how Cassandra can be used easily with other Big Data technologies to solve various business problems.

Chapter 6, Patterns and Anti-Patterns, covers some design patterns and anti-patterns that can be used to solve various software problems. The chapter also describes how and where to use these patterns.

For More Information:  
www.packtpub.com/cassandra-design-patterns/book
The demand from web-scale applications in today's social and mobile-driven Internet has resulted in the recent prominence of NoSQL data stores that can handle and scale up to terabytes and petabytes of data.

An interesting fact is that there are more than 100 NoSQL data stores in the software industry today, and clearly, Cassandra has emerged as one of the leaders in this congested arena, thanks to its distinct capabilities that include easy scalability and ease of usage.

Let us look at the Cassandra architecture and data modeling to figure out the key reasons behind Cassandra's success.

**Understanding the background of Cassandra's architecture**

Cassandra's architecture is based on the best-of-the-world combination of two proven technologies—Google BigTable and Amazon Dynamo. So, it is important to understand some key architectural characteristics of these two technologies before talking about Cassandra's architecture.

For More Information:
www.packtpub.com/cassandra-design-patterns/book
An Overview of Architecture and Data Modeling in Cassandra

Before starting the discussion of these architectures, one of the important concepts to touch upon is the **CAP theorem**, also known as the Brewer Theorem—named after its author Eric Brewer. Without going into the theoretical details of the CAP theorem, it would be simple to understand that CAP stands for Consistency, Availability, and Partition tolerance. Also, the theorem suggests that a true distributed system can effectively cater to only two of the mentioned characteristics.

Amazon Dynamo

Amazon Dynamo is a proprietary key-value store developed at Amazon. The key design requirements are high performance and high availability with continuous growth of data. These requirements mean that firstly, Dynamo has to support scalable, performant architectures that would be transparent to machine failures, and secondly, Dynamo used data replication and autosharding across multiple machines. So, if a machine goes down, the data would still be available on a different machine. The autosharding or automated distribution of data ensures that data is divided across a cluster of machines. A very important characteristic of Amazon Dynamo design is peer-to-peer architecture, which means that there is no master involved in managing the data—each node in a Dynamo cluster is a standalone engine. Another aspect of Dynamo is its simplicity in data modeling, as it uses a simple key-value model.

So where does Dynamo stand with respect to the CAP theorem? Dynamo falls under the category of **Availability and Partition tolerance above Consistency (AP)**. However, it is not absolutely true that Dynamo does not support Consistency—this cannot be expected from a production-grade, real-world data store. Dynamo uses the concept of **Eventual Consistency**, which means that the data would eventually become more consistent over time. Remember, Dynamo keeps replicas of the same data across nodes, and hence, if the state of the same dataset is not the same as its copies due to various reasons, say in the case of a network failure, data becomes inconsistent. Dynamo uses **gossip protocol** to counter this, which means that each Dynamo node talks to its neighbor for failure detection and cluster membership management without the need for a master node. This process of cluster awareness is further used to enable the passing around of messages in the cluster in order to keep the data state consistent across all of the copies. This process happens over time in an asynchronous way and is therefore termed as Eventual Consistency.

For More Information:
www.packtpub.com/cassandra-design-patterns/book
Google BigTable

Google BigTable is the underlying data store that runs multiple popular Google applications that we use daily, ranging from Gmail, YouTube, Orkut, Google Analytics, and much more. As it was invented by Google, BigTable is designed to scale up to petabytes (PB) of data and cater to real-time operations necessary for web-scale applications. So, it has real fast reads and writes, scales horizontally, and has high availability—how many times have we heard Google services failing!

Google BigTable uses an interesting and easy-to-understand design concept for its data storage—the data writes are first recorded in a commit log and then the data itself is written to a memory store. The memory store is then persisted in the background to a disk-based storage called Sorted String Table (SSTable). The writes are super fast because the data is actually being written to a memory store and not directly to a disk, which is still a major bottleneck for efficient reads and writes. The logical question to ask here is what happens if there is a failure when the data is still in memory and not persisted to SSTable. A commit log solves this problem. Remember that the commit log contains a list of all the operations happening on the data store. So, in the case of failures, the commit logs can be replayed and merged with the SSTable to reach a stage where all the commit logs are processed and cleared and become a part of the SSTable. Read queries from BigTable also use the clever approach of looking up the data in a merged view of the memory store and the SSTable store— the reads are super fast because the data is either available in the memory or because SSTable indexing returns the data almost immediately.

How fast is "super fast"?
Reads and writes in the memory are around 10,000 times faster than in the traditional disks. A good guide for every developer trying to understand the read and write latencies is Latency Numbers Every Programmer Should Know by Jeff Dean from Google Inc.

Google BigTable does have a problem; it falls under the category of Consistency and Partition tolerance (CP) in the CAP theorem and uses the master-slave architecture. This means that if the master goes down, there are chances that the system might not be available for some time. Google BigTable uses a lot of clever mechanisms to take care of high availability though; but, the underlying principle is that Google BigTable prefers Consistency and Partition tolerance to Availability.
Understanding the background of Cassandra modeling

Dynamo's data modeling consists of a simplistic key-value model that would translate into a table in RDBMS with two columns—a primary key column and an additional value column. Dynamo supports the `get()` and `put()` functions for the reads and the insert/update operations in the following API formats:

- `get(key)`: The datatype of key is bytes
- `put(key, value)`: The datatype of key and value is bytes

Google BigTable has a more complex data model and uses a multidimensional sorted map structure for storing data. The key can be considered to be a complex key in the RDBMS world, consisting of a key, a column name, and a timestamp, as follows:

```
(row:string, column:string, time:int64) -> string
```

The row key and column names are of the string datatype, while the timestamp is a 64-bit integer that can represent real time in microseconds. The value is a simple string.

Google BigTable uses the concept of column families, where common columns are grouped together, so the column key is actually represented as `family: qualifier`.

Google BigTable uses Google File System (GFS) for storage purposes. Google BigTable also uses techniques such as Bloom filters for efficient reads and compactions for efficient storage.

An overview of architecture and modeling

When Cassandra was first being developed, the initial developers had to take a design decision on whether to build a Dynamo-like or a Google BigTable-like system, and these clever guys decided to use the best of both worlds. Hence, the Cassandra architecture is loosely based on the foundations of peer-to-peer-based Dynamo architecture, with the data storage model based on Google BigTable.

Cassandra uses a peer-to-peer architecture, unlike a master-slave architecture, which is prone to single point of failure (SPOF) problems. Cassandra is deployed on multiple machines with each machine acting as a node in a cluster. Data is autosharded, that is, automatically distributed across nodes using key-based sharding, which means that the keys are used to distribute the data across the cluster. Each key-value data element in Cassandra is replicated across the cluster on other nodes (the default replication is 3) for high availability and fault tolerance. If a node goes down, the data can be served from another node having a copy of the original data.
Sharding is an old concept used for distributing data across different systems. Sharding can be horizontal or vertical. In horizontal sharding, in case of RDBMS, data is distributed on the basis of rows, with some rows residing on a single machine and the other rows residing on other machines. Vertical sharding is similar to columnar storage, where columns can be stored separately in different locations.

Hadoop Distributed File Systems (HDFS) use data-volumes-based sharding, where a single big file is sharded and distributed across multiple machines using the block size. So, as an example, if the block size is 64 MB, a 640 MB file will be split into 10 chunks and placed in multiple machines.

The same autosharding capability is used when new nodes are added to Cassandra, where the new node becomes responsible for a specific key range of data. The details of what node holds what key ranges is coordinated and shared across the cluster using the gossip protocol. So, whenever a client wants to access a specific key, each node locates the key and its associated data quickly within a few milliseconds. When the client writes data to the cluster, the data will be written to the nodes responsible for that key range. However, if the node responsible for that key range is down or not reachable, Cassandra uses a clever solution called Hinted Handoff that allows the data to be managed by another node in the cluster and to be written back on the responsible node once that node is back in the cluster.

The replication of data raises the concern of data inconsistency when the replicas might have different states for the same data. Cassandra uses mechanisms such as anti-entropy and read repair for solving this problem and synchronizing data across the replicas. Anti-entropy is used at the time of compaction, where compaction is a concept borrowed from Google BigTable. Compaction in Cassandra refers to the merging of SSTable and helps in optimizing data storage and increasing read performance by reducing the number of seeks across SSTables. Another problem that compaction solves is handling deletion in Cassandra. Unlike traditional RDBMS, all deletes in Cassandra are soft deletes, which means that the records still exist in the underlying data store but are marked with a special flag so that these deleted records do not appear in query results. The records marked as deleted records are called tombstone records. Major compactions handle these soft deletes or tombstones by removing them from the SSTable in the underlying file stores. Cassandra, like Dynamo, uses a Merkle tree data structure to represent the data state at a column family level in a node. This Merkle tree representation is used during major compactions to find the difference in the data states across nodes and reconciled.

For More Information:
www.packtpub.com/cassandra-design-patterns/book
The Merkle tree or Hash tree is a data structure in the form of a tree where every non-leaf node is labeled with the hash of children nodes, allowing the efficient and secure verification of the contents of the large data structure.

Cassandra, like Dynamo, falls under the AP part of the CAP theorem and offers a tunable consistency level. Cassandra provides multiple consistency levels, as illustrated in the following table:

<table>
<thead>
<tr>
<th>Operation</th>
<th>ZERO</th>
<th>ANY</th>
<th>ONE</th>
<th>QUORUM</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Reads from one node</td>
<td>Read from a majority of nodes with replicas</td>
<td>Read from all the nodes with replicas</td>
</tr>
<tr>
<td>Write</td>
<td>Asynchronous write</td>
<td>Writes on one node including hints</td>
<td>Writes on one node with commit log and Memtable</td>
<td>Writes on a majority of nodes with replicas</td>
<td>Writes on all the nodes with replicas</td>
</tr>
</tbody>
</table>

A summary of the features in Cassandra

The following table summarizes the key features of Cassandra with respect to its origins in Google BigTable and Amazon Dynamo:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cassandra implementation</th>
<th>Google BigTable</th>
<th>Amazon Dynamo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Peer-to-peer architecture, ring-based deployment architecture</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Data model</td>
<td>Multidimensional map (row, column, timestamp) -&gt; bytes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CAP theorem</td>
<td>AP with tunable consistency</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Storage architecture</td>
<td>SSTable, Memtables</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Storage layer</td>
<td>Local filesystem storage</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fast reads and efficient storage</td>
<td>Bloom filters, compactions</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

For More Information:  
www.packtpub.com/cassandra-design-patterns/book
Summary

Cassandra packs the best features of two technologies proven at scale—Google BigTable and Amazon Dynamo. However, today Cassandra has evolved beyond these origins with new unique and enterprise-ready features such as Cassandra Query Language (CQL), support for collection columns, lightweight transactions, and triggers.

In the next chapter, we will talk about the design and use case patterns that are used in the world of Cassandra and utilize its architectural and modeling strengths.
Where to buy this book

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.