Examination of the Cassandra Distributed Storage System

Gabriel Bassett
Information Security Analytics
gabe@infosecanalytics.com

ABSTRACT

In this paper we examine Cassandra, a distributed storage system originally designed by Facebook, and later adopted by the Apache organization as a top-level project. Cassandra is a key-value store designed to address many of the shortcomings of current databases including scalability, performance and fault tolerance. Cassandra also purposefully adopts some specific limitations to facilitate its primary features. Cassandra is uniquely well suited for many use cases including backup and high availability. The features inherent in Cassandra make appropriate for the storage of very large attribute graphs that could facilitate significant innovation in the information security space.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Distributed databases.*

General Terms

Algorithms, Documentation, Performance, Design, Security, Standardization, Languages.

Keywords

Cassandra, Facebook, Apache, Distributed, Tuning, Fault Tolerance, Performance, CQL, Scaling, Information Security, Security, Graph, Key Value.

1. INTRODUCTION

Cassandra is a data store original created by Facebook in 2008. Its original purpose was to power the Facebook Inbox Search (Lakshman & Malik, 2008). Facebook required a storage system that met its strict performance, reliability, and efficiency requirements as well as the needs of its expected growth. Cassandra was implemented to meet these needs. It combines multiple previously documented methods for ensuring availability and scalability in a single storage system. As with Hadoop, Cassandra is written in Java.

Cassandra is most directly comparable to a database though many differences exist. Since its release in 2008, it has undergone significant development with major releases in 2010, 2011, and 2013. This has added functionality including Apache Hadoop MapReduce support, caching improvements, the Cassandra Query Language (CQL), read performance improvements and lightweight transactions among other improvements. (The Apache Software Foundation, 2014)

2. CASSANDRA FUNCTIONALITY

Cassandra shares many similarities with a Relational Database Management System (RDBMS). Cassandra has tables with rows and columns. It has a query language very similar to Structured Query Language (SQL) known as CQL.

However, Cassandra has many design decisions that make it very different form a traditional RDBMS. Cassandra represents a key-value store. In such a case, rows, rather than having predefined columns, has key-value pairs associated with it. Columns are organized into column families and rows into tables. Figure 1 illustrates this structure.

Column Family 1			
Node_1			
13e82ddbfe317f36e105d7b44b8669a7	class	attribute	domain
	attribute	domain	google-public-dns-a.google.com.
Node_2			
db4730a5745d877b42a0a17a7f719e2a	class	attribute	ip
	attribute	ip	8.8.8.8

Column Family 2				
Edge_1				
f299e5896e7ed34313321df245bd572b	source	Relationship	target	Described_by
	Node_2	Described_by	Node_1	Resolves_to

Figure 1 Key Value Store

Rows must have a primary key that is hashed and used to partition the dataset across servers. Each server is issued a token and is responsible for all data from the previous server's token to its token. Rows can then be replicated in one of two ways. A node may be responsible for previous nodes' data to the level necessary for the redundancy desired. Alternately, redundancy may be assigned to avoid correlated failures. Such replication would provision data across network aggregation zones, and data centers.

The data in Cassandra rows also differ from those in a normal relational database. While a normal relational database must have a schema defined at table creation, Cassandra schemas may change online. Additionally, each row may have a different set of columns. When a specific row is required, its key is hashed and the row is requested through a binary search.

Finally, Cassandra lacks a master node, router node, or slave nodes. There are no unique roles for nodes in the Cassandra cluster. This directly supports fault tolerance.

3. IMPROVEMENTS OVER COMPETITIVE TECHNOLOGY

Cassandra addresses many shortcomings of traditional data storage systems.

3.1 Fault Tolerance

Cassandra is designed with the ethos that failures are common operational occurrences. The designers of the data store worked under the assumption that failures would occur regularly and need to be handled gracefully.

Cassandra's implementation of an architecture with a single role for all nodes provides graceful degradation. Because there are not multiple roles, there are no single points of failure. Additionally, because all nodes can perform all roles, there is no concern over 'conflicting masters' in which two nodes may attempt to perform some role such as the 'master' role which should only occur on one node (Lakshman & Malik, 2008).

Additionally, Cassandra has robust recovery. Because data is spread throughout the cluster, rebuilding the damaged data of a single node is less of a performance impact than in other less robust data stores. Additional capacity or replacement hardware can be added to the cluster without downtime. This combination of graceful degradation and zero impact recovery make Cassandra a clear choice for those needing robust fault tolerance.

3.2 Performance

Cassandra has had significant effort placed on developing its performance. Initial development focused on write speed, however, with version 1.0 and beyond, read speed has been significantly improved including additional optimizations in 2.0 (The Apache Software Foundation, 2014). Table 1 provides a comparison of various NoSQL data stores based on data collected by DataStax[1], (DataStax Corporation, 2013). As can be seen in Table 1, Cassandra provides significant performance increases when compared to alternatives.

Table 1. NoSQL Performance (Operations/Shard/sec)

Workload	Cassandra	HBase	MongoDB
Load	3778	1336	243
Read	602	93	50
Write	6727	2344	53
Read/Write	1055	264	56
Read/Modify /Write	428	131	43

Significant performance can be attributed to the partitioning of data within Cassandra. Cassandra supports two partitioning mechanisms. The first mechanism randomly partitions the data which provides good load balancing. Cassandra can also attempt to partition data by placing similar data together. While this decreases the efficiency of load balancing, it increases the chance that a single node can answer a query, improving performance.

3.3 Scalability

Cassandra benefits from nearly linear scalability. Figure 2 demonstrates this scalability as tested by Netflix in the Amazon Web Services cloud (Cockcroft, 2013). Cassandra achieves this scaling through efficient partitioning and indexing of data.

[1] DataStax provides a Cassandra-based product.

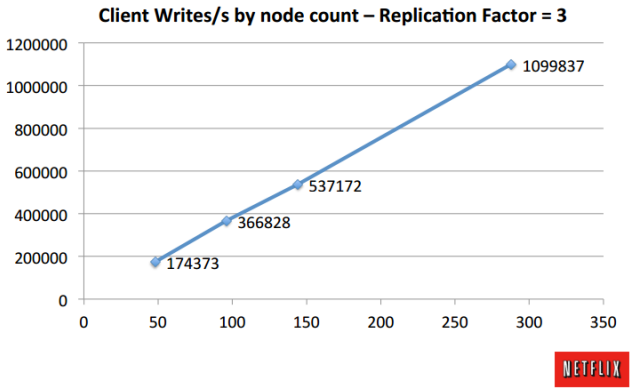

Figure 2 Cassandra Linear Performance Scaling

3.4 Malleability of Data Structure

The ability to define data schemas during run time as well as vary the columns associated with a row in a column family open Cassandra to many more use cases. This malleability is desirable in any situation where the data structures are unknown, may change over time, or simply may not be documented at all. Even with known internal data, the structure may change for many reasons. The flexibility for the data store to change with the data structure is essential to facilitating these use cases.

3.5 Tunable

Cassandra is uniquely tunable. Writes can be tuned from fully asynchronous to fully synchronous. In situations where the correct information must be read, Cassandra supports fully synchronous replication. In situations where old data is acceptable, asynchronous replication may be used. Cassandra also provides a compromise between the two extremes in which consensus is reached in the replication prior to read.

3.6 Robust Interfaces

Of significant benefit over many of its competitors are Cassandra's robust interfaces. Cassandra exposes a SQL like language known as CQL. CQL supports common SQL commands such as: SELECT, INSERT, UPDATE, DELETE, CREATE, and DROP (The Apache Software Foundation, 2011). This provides ease of use for the large body of people who have SQL experience. It also has facilitated the development of multiple language bindings including Java, Python, and Node.JS. Some key SQL commands are missing however as described in Section 4. While Cassandra does have a Thrift interface, it is depreciated in current versions.

Cassandra also supports MapReduce. This allows MapReduce to read data from Cassandra as well as write data to MapReduce. Data may also be loaded into and stored from Pig through custom LoadFunct and StoreFunct implementations. Other tools supported include Hive and Oozie. In addition to capabilities provided with Cassandra, it is also supported in many other distributed systems such as Apache Spark and TitanDB.

3.7 Simple Installation

Cassandra has only one dependency (Java). Cassandra may be installed as a Linux package or OS X brew formula with no other configuration. Once installed, it may be immediately used through the built in CQL shell where the previously articulated CQL commands can be issued. This ease of use lowers the barrier

to entry and improves the overall install base leading to more robust support.

3.8 Proven

Cassandra has been in production since its introduction in 2008 managing some of the largest data sets in the world. This includes data sets from Facebook and Netflix as previously established. It also includes companies such as Backupify and OpenWave as documented in Section 5. Dozens of other companies utilize Cassandra to manage their big data sets (Ellis, Apache Cassandra: Real World Scalability, today, 2012).

Wide use leads to significant benefits. The identification of defects, particularly in edge cases, is much more likely under wide use. Also, the additional usage leads to additional development helping complete the feature set. It also establishes a strong community to provide support and documentation for Cassandra.

4. LIMITATIONS

While Cassandra has many features, it also has some notable limitations. These can be grouped into two groups: CQL limitations and storage limitations (The Apache Software Foundation, 2013).

4.1 CQL Limitations

CQL notably does not support joins, subqueries, or aggregation. These features are purposefully omitted by design. By omitting these commands, the developers of Cassandra hope to increase the chance that a single node can answer a query. Additionally, ordering of data is done per-partition at create time. This is to prevent a developer attempting to sort a very large data set in production.

4.2 Storage Limitations

There also exist hard limits on the size of Cassandra partitions. They must fit on disk and do not allow a single column to be larger than 2GB. A single partition may only have 2 billion cells and collection values have a maximum size of 64KB.

5. REPRESENTATIVE USE CASES

Cassandra has robust support in production environments in industry. The below examples provide clear demonstrations of the benefits of Cassandra.

5.1 Backupify

Backupify provides backup up solution for cloud storage. Backup is a scenario in which multiple writes must be done but very few reads. Because of this, Cassandra is an excellent choice for the storage of metadata of files and objects. Additionally, Cassandra has the added feature of durability. Durability indicates that when Cassandra reports a write as complete, it has been written to disk rather than just to memory where it may be lost due to power outage (Ellis, Apache Cassandra: Real World Scalability, today, 2012).

5.2 OpenWave

OpenWave provides a turnkey, black box messaging solution for service providers. User expectations for messaging are increasing quickly. Users expect zero downtime, the ability to share media such as audio and video, indefinite storage, and the ability to quickly search the entire dataset. All these features are expected at no cost putting significant cost pressure on the provider. (Mainstay LLC, 2013).

Cassandra efficiently meets all of these expectations. It scales with user demand. It is quickly searchable due to its MapReduce integration. It is highly available with graceful degradation and non-impacting recovery. Cassandra can store large blob data and store it indefinitely thanks to distributed nature and ability to expand with no impact to operations. Thanks to Cassandra's open source license and simple interfaces, its implementation is very cost efficient.

6. APPLICATION TO INFORMATION SECURITY

6.1 Information Security Opportunities

Cassandra has direct application to information security. There exists an unrealized opportunity in information security (and other classification problems) to classify atomic objects based on their relationships with other atomic objects. For example, An Internet Protocol (IP) may be classified as malicious if it is associated with 3 domains, each of which is associated with 3 other IP addresses which are themselves already classified as malicious but only one domain associated with one IP address classified as benign. Figure 3 illustrates this graph.

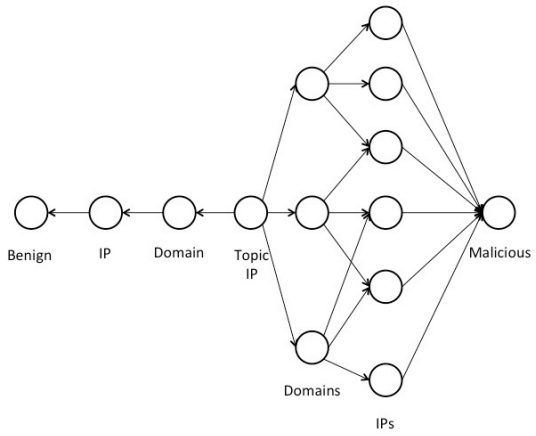

Figure 3 Information Security Graph Structure

The effectiveness of this approach is directly dependent on the ability to store very large amounts of data and its relationships. Records are generated at speed of hundreds of thousands or millions per second. Each record contains multiple atomic key-value pairs. Each of these atomic key-value pairs needs to be stored along with its relationship to other atomic values in the associated record.

A schema for this type of storage exists (Bassett, 2013). It relies and a triplet of key-value pairs for atomic identification, with support for additional optional keys. All nodes and relationships are defined as Unique Resource Identifier (URIs). URIs for nodes must describe their required key triple as outlined below. URIs for edges may be randomly assigned or derived from their unique triple.

The required node keys are of the form: *class*:<class>, <class>:<key>, <key>:<value>. In our IP address example, the node key-value triplet would be: *class:attribute, attribute:ip, ip:8.8.8.8*. Optional keys include *start_time*, *finish_time*, *id*, and *comment*.

Relationships have similar required key-value triples. At a minimum, a relationship must have *source*, *target*, and

relationship. Additional keys include *start_time*, relationship value as a key, *finish_time*, *confidence*, *comment*, *id*, and *directed*.

This combination of semi-structured key-value pairs in both nodes and edges makes their storage perfect for Cassandra. Cassandra provides the parallel performance necessary for the massive ingestion of data. Cassandra also provides the long-term storage and storage expansion necessary to hold the quantity of data expected. Its indexing methodology including secondary index support allows the ability to quickly find atomic nodes and associated relationships. Its ability to store different columns for different rows facilitates the flexible nature of the data associated with nodes.

6.2 Barriers to Use

There exist, however, some barriers to usage of Cassandra for this purpose. First, The Cyber Attack Graph Schema (CAGS) 1.0 definition allows for keys for each attribute type. It is unclear how many attribute types are likely to exist in the graph or how well Cassandra will perform under regular addition of keys.

Second, Cassandra, as with many distributed open source platform, requires an operational workflow designed around commodity hardware and the associated regular failures. As such, to use Cassandra at scale may require a revision of data center operations plans. An alternate approach is to host the infrastructure in a public cloud. This alternative provides its own impediments as many organizations may not be willing to move sensitive information security data to the cloud or may not be able to afford the throughput necessary to store such data off site.

Finally, to implement such a construct requires a unique skill set which cannot be easily acquired. A single person must fully understand the schema of the graph, its use, and the algorithms likely to be applied to both storing and retrieving data. Additionally, the same individual must fully understand the storage and compute stacks. Without both understandings, implementation of storing of data, maintenance of the schema, and loading of the data from storage will be challenging. And if implementation is complete, without the dual understanding, accomplishment of performance objectives is unlikely.

7. Conclusion

In conclusion, Cassandra represents the integration of multiple technologies to provide a single data storage system capable of meeting the performance, availability/fault tolerance, and scaling needs of today's businesses. Its implementation is tested, robust, and feature rich. It has multiple applications in many areas with a unique application in information security.

Cassandra is still growing with version 2.1 slated for imminent release. Tests show that release candidates of Cassandra version 2.1 offers significant performance benefits over the current version of Cassandra (Ellis, Cassandra 2.1: now over 50% faster, 2014). Given its wide penetration and solid record, we can expect a robust future for Cassandra.

8. ACKNOWLEDGMENTS

My thanks to my wife, Tabitha Bassett, my peers, Jasmine Henry and Kindall Deitmen, and Curt Mast and Phil Prichard for supporting me through the development of this paper.

9. REFERENCES

Bassett, G. (2013, 07 29). *Cyber Attack Graph Schema (CAGS) 1.0*. Retrieved from Information Security Analytics Blog: http://blog.infosecanalytics.com/2013/07/cyber-attack-graph-schema-cags-10.html

Cockcroft, A. (2013). Global Netflix: Replacing Datacenter Oracle with Global Apache Cassandra on AWS. *15th International Workshop on High Performance Transaction Systems (HPTS)*. Pacific Grove, CA: Netflix.

DataStax Corporation. (2013). *Benchmarking Top NoSQL Databases.*

Ellis, J. (2012). Apache Cassandra: Real World Scalability, today. *NoSQL Matters.* Barcelona: NoSQL Matters conference 2012, Cologne.

Ellis, J. (2014, 07 16). *Cassandra 2.1: now over 50% faster*. Retrieved from DataStax: http://www.datastax.com/dev/blog/cassandra-2-1-now-over-50-faster

Lakshman, A., & Malik, P. (2008). *Cassandra - A Decentralized Structured Storage System.* Facebook.

Mainstay LLC. (2013). *DataStax Case Study of Openwave Messaging.* Santa Clara, CA: DataStax.

The Apache Software Foundation. (2011, 10 12). *Cassandra Query Language (CQL) v2.0*. Retrieved from The Apache Software Foundation: https://cassandra.apache.org/doc/cql/CQL.html

The Apache Software Foundation. (2013, 11 13). *Cassandra Limitations*. Retrieved from Cassandra Wiki: http://wiki.apache.org/cassandra/CassandraLimitations

The Apache Software Foundation. (2014, 08 19). *Cassandra Change Log*. Retrieved from The Apache Software Foundation: https://issues.apache.org/jira/browse/CASSANDRA/?selectedTab=com.atlassian.jira.jira-projects-plugin:changelog-panel