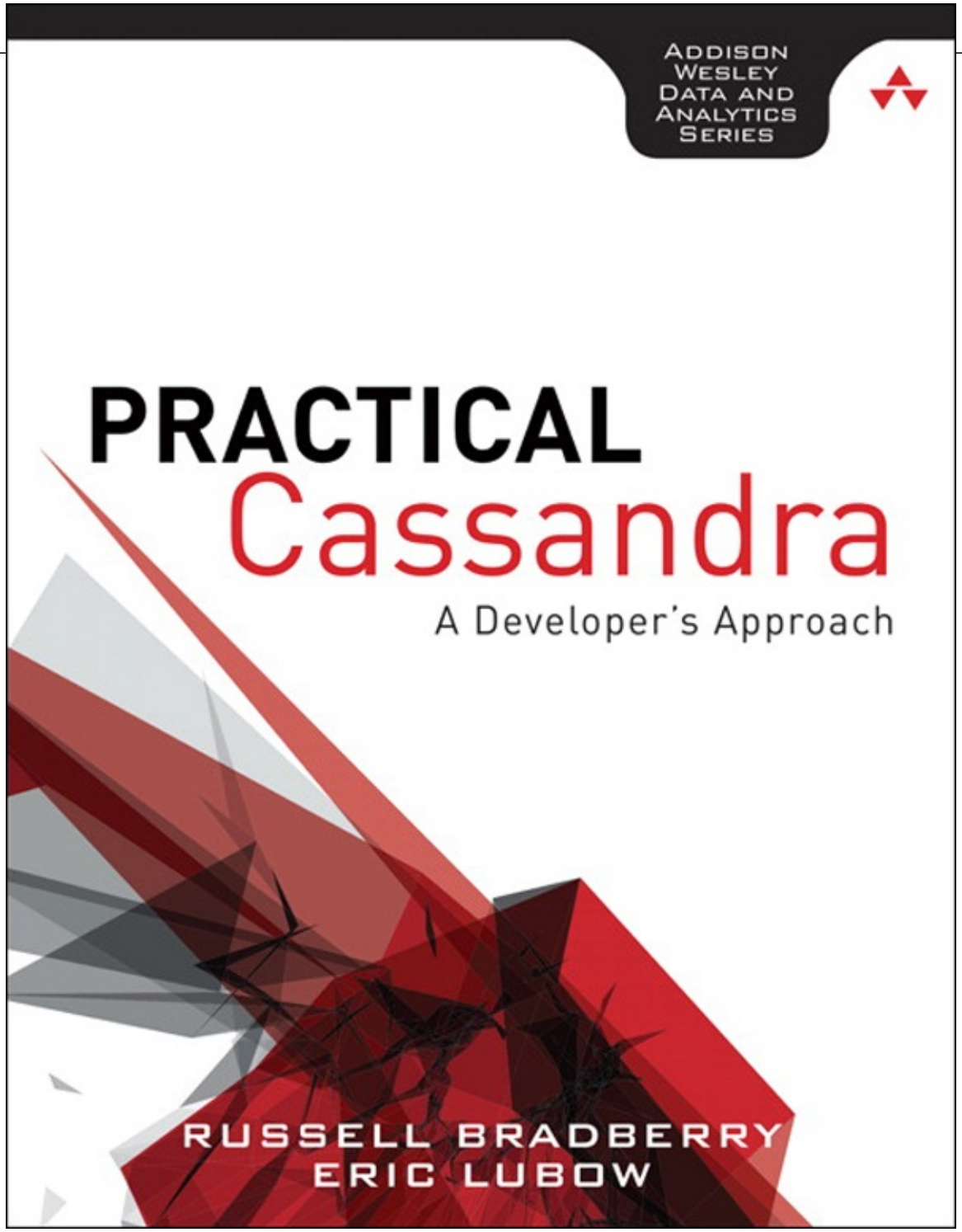

Addison
Wesley
Data and
Analytics
Series
PRACTICAL
Cassandra
A Developer's Approach
RUSSELL BRADBERRY
ERIC LUBOW

About This eBook

ePUB is an open, industry-standard format for eBooks. However, support of ePUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the eBook in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a "Click here to view code image" link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Practical Cassandra

A Developer's Approach

Russell Bradberry
Eric Lubow

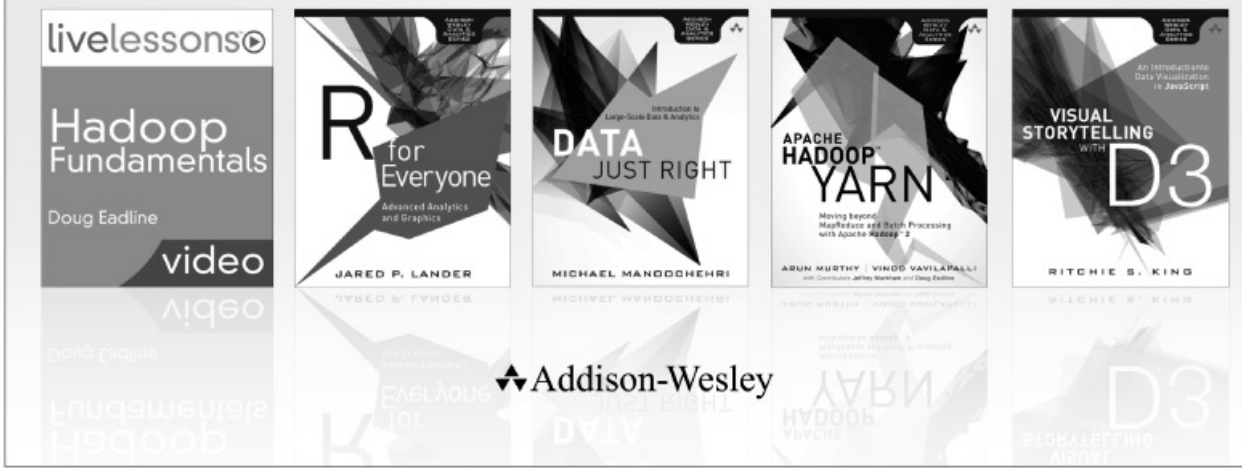
Addison-Wesley
Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Cataloging-in-Publication Data is on file with the Library of Congress.

ISBN-13: 978-0-321-93394-2
ISBN-10: 0-321-93394-X

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, December 2013

The Addison-Wesley Data and Analytics Series



Visit **informit.com/awdataseries** for a complete list of available publications.

The **Addison-Wesley Data and Analytics Series** provides readers with practical knowledge for solving problems and answering questions with data. Titles in this series primarily focus on three areas:

1. **Infrastructure:** how to store, move, and manage data
2. **Algorithms:** how to mine intelligence or make predictions based on data
3. **Visualizations:** how to represent data and insights in a meaningful and compelling way

The series aims to tie all three of these areas together to help the reader build end-to-end systems for fighting spam; making recommendations; building personalization; detecting trends, patterns, or problems; and gaining insight from the data exhaust of systems and user interactions.

Make sure to connect with us!
informit.com/socialconnect

informIT.com the trusted technology learning source | Addison-Wesley | Safari Books Online

This book is for the community. We have been a part of the Cassandra community for a few years now, and they have been fantastic every step of the way. This book is our way of giving back to the people who have helped us and have allowed us to help pave the way for the future of Cassandra.

Contents

Foreword by Jonathon Ellis

I was excited to learn that *Practical Cassandra* would be released right at my five-year anniversary o working on Cassandra. During that time, Cassandra has achieved its goal of offering the world's mos reliable and performant scalable database. Along the way, Cassandra has changed significantly, and a modern book is, at this point, overdue. Eric and Russell were early adopters of Cassandra at SimpleReach; in *Practical Cassandra*, you benefit from their experience in the trenches administerin Cassandra, developing against it, and building one of the first CQL drivers.

If you are deploying Cassandra soon, or you inherited a Cassandra cluster to tend, spend some time with the deployment, performance tuning, and maintenance chapters. Some complexity is inherent in distributed system, particularly one designed to push performance limits and scale without compromise; forewarned is, as they say, forearmed. If you are new to Cassandra, I highly recommend the chapters on data modeling and CQL. The Cassandra Query Language represents a major shift in developing against Cassandra and dramatically lowers the learning curve from what you may expect or fear.

Here's to the next five years of progress!

—Jonathon Ellis, Apache Cassandra Chai

Foreword by Paul Dix

Cassandra is quickly becoming one of the backbone components for anyone working with large datasets and real-time analytics. Its ability to scale horizontally to handle hundreds of thousands (or millions) of writes per second makes it a great choice for high-volume systems that must also be highly available. That's why I'm very pleased that this book is the first in the series to cover a key infrastructural component for the Addison-Wesley Data & Analytics Series: the data storage layer.

In 2011, I was making my second foray into working with Cassandra to create a high-volume, scalable time series data store. At the time, Cassandra 0.8 had been released, and the path to 1.0 was fairly clear, but the available literature was lagging sorely behind. This book is exactly what I could have used at the time. It provides a great introduction to setting up and modeling your data in Cassandra. It has coverage of the most recent features, including CQL, sets, maps, and lists. However it doesn't stop with the introductory stuff. There's great material on how to run a cluster in production, how to tune performance, and on general operational concerns.

I can't think of more qualified users of Cassandra to bring this material to you. Eric and Russell ar Datastax Cassandra MVPs and have been working extensively with Cassandra and running it in production for years. Thankfully, they've done a great job of distilling their experience into this book so you won't have to search for insight into how to develop against and run the most current release o Cassandra.

—Paul Dix, Series Edito

Preface

Apache Cassandra is a massively scalable, open-source, NoSQL database. Cassandra is best suited to applications that need to store large amounts of structured, semistructured, and unstructured data. Cassandra offers asynchronous masterless replication to nodes in many data centers. This gives it the capability to have no single point of failure while still offering low latency operations.

When we first embarked on the journey of writing a book, we had one goal in mind: We wanted to keep the book easily digestible by someone just getting started with Cassandra, but also make it a useful reference guide for day-to-day maintenance, tuning, and troubleshooting. We know the pain of scouring the Internet only to find outdated and contrived examples of how to get started with a new technology. We hope that *Practical Cassandra* will be the go-to guide for developers—both new and at an intermediate level—to get up and running with as little friction as possible.

This book describes, in detail, how to go from nothing to a fully functional Cassandra cluster. It shows how to bring up a cluster of Cassandra servers, choose the appropriate configuration options fo the cluster, model your data, and monitor and troubleshoot any issues. Toward the end of the book, w provide sample code, in-depth detail as to how Cassandra works under the covers, and real-world cas studies from prominent users.

What's in This Book?

This book is intended to guide a developer in getting started with Cassandra, from installation to common maintenance tasks to writing an application. If you are just starting with Cassandra, this boo will be most helpful when read from start to finish. If you are familiar with Cassandra, you can skip around the chapters to easily find what you need.

- Chapter 1, Introduction to Cassandra: This chapter gives an introduction to Cassandra and the philosophies and history of the project. It provides an overview of terminology, what Cassandra is best suited for, and, most important what we hope to accomplish with this book.
- Chapter 2, Installation: Chapter 2 is the start-to-finish guide to getting Cassandra up and running. Whether the installation is on a single node or a large cluster, this chapter guides you through the process. In addition to cluster setup, the most important configuration options are outlined.
- Chapter 3, Data Modeling: Data modeling is one of the most important aspects of using Cassandra. Chapter 3 discusses the primary differences between Cassandra and traditional RDBMSs, as well as going in depth into different design patterns, philosophies, and special features that make Cassandra the data store of tomorrow.
- Chapter 4, CQL: CQL is Cassandra's answer to SQL. While not a full implementation of SQL, CQL helps to bridge the gap when transitioning from an RDBMS. This chapter explores in dept the features of CQL and provides several real-world examples of how to use it.
- Chapter 5, Deployment and Provisioning: After you've gotten an overview of installation and querying, this chapter guides you through real-world deployment and resource provisioning. Whether you plan on deploying to the cloud or on bare-metal hardware, this chapter is for you. In addition to outlining provisioning in various types of configurations, it discusses the impact of the different configuration options and what is best for different types of workloads.
- Chapter 6, Performance Tuning: Now that you have a live production cluster deployed, this chapter guides you through tweaking the Cassandra dials to get the most out of your hardware,

operating system, and the Java Virtual Machine (JVM).

- Chapter 7, Maintenance: Just as with everything in life, the key to having a performant and, more important, working Cassandra cluster is to maintain it properly. Chapter 7 describes all th different tools that take the headache out of maintaining the components of your system.
- Chapter 8, Monitoring: Any systems administrator will tell you that a healthy system is a monitored system. Chapter 8 outlines the different types of monitoring options, tools, and what to look out for when administering a Cassandra cluster.
- Chapter 9, Drivers and Sample Code: Now that you have a firm grasp on how to manage and maintain your Cassandra cluster, it is time to get your feet wet. In Chapter 9, we discuss the different drivers and driver features offered in various languages. We then go for the deep dive by presenting a working example application in not only one, but four of the most commonly used languages: Java, C#, Ruby, and Python.
- Chapter 10, Troubleshooting: Now that you have written your sample application, what happen when something doesn't quite work right? Chapter 10 outlines the tools and techniques that can be used to get your application back on the fast track.
- Chapter 11, Architecture: Ever wonder what goes on under the Cassandra "hood"? In this chapter, we discuss how Cassandra works, how it keeps your data safe and accurate, and how it achieves such blazingly fast performance.
- Chapter 12, Case Studies: So who uses Cassandra, and how? Chapter 12 presents three case studies from forward-thinking companies that use Cassandra in unique ways. You will get the perspective straight from the mouths of the developers at Ooyala, Hailo, and eBay.
- Appendix A, Getting Help: Whether you're stuck on a confusing problem or just have a theoretical question, having a place to go for help is paramount. This appendix tells you about the best places to get that help.
- Appendix B, Enterprise Cassandra: There are many reasons to use Cassandra, but sometimes it may be better for you to focus on your organization's core competencies. This appendix describes a few companies that can help you leverage Cassandra efficiently and effectively whi letting you focus on what you do best.

Code Samples

All code samples and more in-depth examples can be found on GitHub at http://devdazed.github.io/practical-cassandra/.

Acknowledgments

We would like to acknowledge everyone involved with Cassandra and the Cassandra community—everyone from the core contributors of Cassandra all the way down to the end users who have made i such a popular platform to work with. Without the community, Cassandra wouldn't be where it is today. Special thanks go to

- Jay Patel for putting together the eBay case study
- Al Tobey and Evan Chan for putting together the case study on Ooyala
- Dominic Wong for putting together the Hailo case study
- All the technical reviewers, including Adam Chalemian, Mark Herschberg, Joe Stein, and Brya Smith, who helped give excellent feedback and ensured technical accuracy where possible
- Paul Dix for setting us up and getting us on the right track with writing

About the Authors

Russell Bradberry (Twitter: @devdazed) is the principal architect at SimpleReach, where he is responsible for designing and building out highly scalable, high-volume, distributed data solutions. H has brought to market a wide range of products, including a real-time bidding ad server, a rich media ad management tool, a content recommendation system, and, most recently, a real-time social intelligence platform. He is a U.S. Navy veteran, a DataStax MVP for Apache Cassandra, and the author of the NodeJS Cassandra driver Helenus.

Eric Lubow (Twitter: @elubow) is currently chief technology officer of SimpleReach, where he builds highly scalable, distributed systems for processing social data. He began his career building secure Linux systems. Since then he has worked on building and administering various types of ad systems, maintaining and deploying large-scale Web applications, and building email delivery and analytics systems. He is also a U.S. Army combat veteran and a DataStax MVP for Apache Cassandra

Eric and Russ are regular speakers about Cassandra and distributed systems, and both live in New York City.

1. Introduction to Cassandra

Apache Cassandra is a powerful and massively scalable NoSQL database. It is architected to handle real-time big-data workloads across multiple data centers with no single point of failure. It works on commodity hardware and can easily be deployed in a cloud-based infrastructure. But before we get into the nitty-gritty of things, here is a quick lesson in Greek mythology.

A Greek Story

In Greek mythology, Cassandra was the beautiful daughter of King Priam and Queen Hecuba of Troy the twin sister of Helenus and younger sister to the great Trojan warrior Hector, and eventually a priestess of Apollo. She was believed to be the second-most beautiful woman in the world. Her beaut was compared to the likes of Aphrodite or Helen of Troy. She had red curly hair, blue eyes, and fair skin and was intelligent, charming, friendly, and very desirable. The other side of Cassandra was that she was generally considered to be insane.

When Apollo first saw Cassandra, he immediately fell in love with her. To show his love, he offere her the gift of prophecy if she would kiss him, and she agreed. But when Apollo went to kiss Cassandra, instead of a kiss, she spat in his mouth. Because Apollo had already granted Cassandra th gift of prophecy, he could not take it away. But he did change it so that even though Cassandra would always know what was going to happen, nobody would ever believe her.

And in fabled fashion, when Cassandra told the people of Troy that the Trojan Horse was bad news they ignored her and Troy was captured. After the Trojans lost the war, a Greek warrior named Ajax took Cassandra prisoner and gave her to King Agamemnon as a slave. She told Agamemnon that his wife, Clytemnestra, was going to kill him. But Apollo's curse did not allow anyone to believe her. After killing her husband, King Agamemnon, Clytemnestra then killed Cassandra.

The reason for telling this story is twofold. First, it shows a little about why the name Cassandra was chosen for this database. She was a repository of knowledge of things that were going to happen. This is similar to the way you can use the Cassandra system to help you build a better product by having a keen understanding of what's going on around you. Second, the names of many of the characters in this and other Greek tragedies are used for the names of many of the applications that play well with Cassandra. These include Helenus (the Node.js driver), Priam (a Cassandra automatio tool), and Hector (the Java driver), just to name a few.

What Is NoSQL?

There is no single definition for NoSQL. To some it stands for "Not Only SQL"; to others it means "No SQL." Either way, it refers to the departure from the traditional relational database technologies that have dominated the development landscape for the past few decades.

What is likely the largest driver of the NoSQL movement is a commonly held belief that relational databases are not well suited to large amounts of data and scale. Whether or not this is true, the emergence of the key/value, graph, document, and "big table" data storage engines shows that a new generation of database technologies is taking center stage.

There is no single database technology that is synonymous with the NoSQL movement. Branding and marketing seem to be mostly what determine how relevant a technology is to the terminology.

There's No Such Thing as "Web Scale"

Another marketing term that gets thrown around quite frequently is "Web scale." It is used quite often when discussing how to determine whether a database system is suitable for a particular Web application's needs and whether it will hold up as the application grows. This is a very subjective ter as everyone's needs are different. A simple SQL setup will achieve most scalability needs. Dependin on the read/write patterns of an application, one may need a specialized database, such as Kyoto Cabinet (previously named Tokyo Cabinet) for key/value or MongoDB as a document store. In a system that needs high write throughput and linear scalability, Cassandra is a great fit and will hold u under some very heavy workloads.

The key point to remember when discussing the idea of Web scale technologies is that nearly everything out there will scale with enough money, hardware, and headaches. The trick is to figure ou which piece of software is best suited for your usage patterns and workloads and will scale out in a way suitable for your application and your organization.

ACID, CAP, and BASE

Before we get too deep into Cassandra, it is important to understand some of the basic concepts tha surround databases so you know what concessions you may have to make when choosing a system. There are three main sets of properties that define what database systems are capable of. Those are ACID, CAP, and BASE. ACID comprises some of the general properties of database systems. CAP covers a little more about distributed systems. BASE is a little newer theory and includes the practica considerations of implementing a distributed system.

Understanding these theories will help you to understand where some of the design decisions come in, not only for Cassandra but also for your application and how it is developed. The idea of building distributed applications and distributed systems often comes down to give and take. You may give up consistency for availability. You may find it's wiser for your application's needs to give a little on availability in favor of consistency. ACID, CAP, and BASE are the driving technical theories behind many of these decisions. It is important to understand the trade-offs made in the design of the underlying systems (Cassandra) so you can ensure that your application performs the way you expect it to perform.

ACID

ACID stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability. In order to understand ACID and how it relates to databases, we need to talk about transactions. When it comes to databases, a transaction is defined as a single logical operation. For example, if you are shopping online, every time you add an item to your shopping cart, that item and its quantity make up the database transaction. Even if you add multiple items or multiple quantities of the same item with a single click that entire shopping cart addition is just a single transaction.

Atomicity means that each transaction either works or it doesn't. This is to say that if any single pa of the transaction fails, the entire transaction fails. This should hold true for every situation related to a transaction that could cause a failure. Network failure, power outage, or even a node outage occurring at transaction time should cause a complete transaction failure in an atomic system.

Consistency ensures that when a transaction is complete, whether it is successful or not, the database is still in a valid state. This means that any data written to the database must also be valid. When writing data to the database, you also need to include any database application-level rules such as constraints, cascades, triggers, or stored procedures. The application of those rules should also

leave the data in a valid state.

Isolation is a property that ensures that all transactions that are run concurrently appear as if they were executed serially (one right after the other). Each transaction must be run in a vacuum (isolation). This is to say that if two transactions are run at the same time, they remain independent o each other during the transaction. Some examples of isolation are locks (table, row, column, etc.), dirty reads, and deadlocks. The reason these are relevant is concurrency. Multiple changes can be attempted on the same data or set of data. Knowing what version of the data is the correct one is important for keeping the entire system in a sane state.

Durability means that after the transaction is complete, it will remain that way. In other words, the data change that is incurred by the transaction is stored permanently, regardless of external events (such as a power failure).

CAP

The CAP theorem, also known as Brewer's theorem, asserts that it is impossible for a distributed system to satisfy all three CAP guarantees. CAP stands for **C**onsistency, **A**vailability, and **P**artition tolerance. The important thing to note about the CAP theorem is that all three parts of it cannot be satisfied at the same time.

Although the *C* in CAP also stands for "consistency" (similar to the *C* in ACID), the meaning is different. *Consistency* means that all nodes in a grouping see the same data at the same time. In other words, any particular query hitting any node in the system will return the same result for that specific query. Consistency also further implies that when a query updates a value in one node, the data will b updated to reflect the new value prior to the next query.

The *availability* of a system speaks to the guarantee that regardless of the success or failure of a request, the requestor will receive a response. This means that system operations will be able to continue even if part of the system is down, whatever the reason. Availability is what lets the softwar attempt to cope with and compensate for externalities such as hardware failures, network outages, power failures, and the like.

Partition tolerance refers to the capability of a distributed system to effectively distribute the load across multiple nodes. The load could be data or queries. This implies that even if a few nodes are down, the system will continue to function. Sharding is a commonly used management technique for distributing load across a cluster. Sharding, which is similar to horizontal partitioning, is a way of splitting data into separate parts and moving them to another server or physical location, generally fo performance improvements.

There are various reasons that all three parts of the theorem cannot be satisfied in distributed systems. Most have to do with the volume of the data and how long it takes to move data around and check to ensure that it is correct. CAP is often used to justify the use of weaker consistency models. Many of the CAP-based ideas have evolved into the idea of BASE.

BASE

Just as in chemistry, BASE is at the opposite end of the spectrum from ACID. BASE stands for **B**asically **A**vailable, **S**oft state, and **E**ventual consistency. The notion of BASE comes in when dealin with a distributed system so large that maintaining the principles of CAP becomes impractical. It is worth noting that the constraints on transactions from ACID are still in play at some level; they just happen at different times with slightly different rules.

Having a system be *basically available* means that the system will respond to any request. The

caveat is that the response may be a failure to get the data or that the data may be in an inconsistent c changing state. This is equivalent in the real world to depositing a check in your bank account and waiting for it to go through the clearinghouse to make the funds available to you.

Using the BASE terminology, we can expand on the idea of banking with checks. If your bank has only one branch, consistency and availability are satisfied. No partitioning is necessary, and every transaction you make will be available and consistent with itself. If your bank has two branches, whe you deposit a check into branch A, branch B will not see the funds instantaneously because the data needs time to become eventually consistent. What if you deposit two checks and one bounces? The entire transaction should not fail because of one check; each check will be processed in isolation. A problem with one check should not cause a problem with the whole system. That would not make for very durable system. If the computers at branch A go down, that shouldn't stop branch B from working completely. That would mean that the system isn't very available, so there are safety nets in place.

The idea of a *soft-state* system means the system is always changing. This is typically due to eventual consistency. It is common for soft-state systems to undergo changes even when there is no additional input to them.

Eventual consistency refers to the concept that once a system stops receiving input, the data will propagate to wherever else it needs to be in the system sooner or later. The beauty of this is that the system does not check for consistency on every transaction as is expected in an ACID-compliant system.

Where Cassandra Fits In

Now that we have a decent idea of the tenets of a distributed system, it's time to take a look at where Cassandra excels. There are a lot of database systems, and nearly all of them were designed to handle a particular problem efficiently and effectively. But the most important thing that you need to know when deciding whether Cassandra is the right tool for the job is the goal of the job. In other words, if you can illustrate what it is you are trying to accomplish, you'll be able to determine if Cassandra is what you need to be successful.

In the context of the Web analytics application that we are building, Cassandra is suitable for a variety of reasons. One of the most common use cases for Cassandra is dealing with time-series data. What this means is that there is a sequence of successive data points that are all related to the same topic. For example, every time a page view happens on your Web site, an entry is made into the logs with the time of the event (page view), including some metadata around that event (IP, browser, URL etc.).

Now let's say your Web site isn't made up of just one or two Web servers, but a whole cluster of Web servers is required to support your traffic. And let's also say that you want to store the resulting Web server data in a database and not just aggregate logs on a log server. How is Cassandra well suited for that? Before you can answer whether or not Cassandra is the right tool to help you solve your problem, we should talk about what Cassandra is and where it came from.

What Is Cassandra?

Cassandra is an open-source distributed database management system. It is designed to handle large amounts of data spread across many commodity servers while remaining highly available. Cassandra is loosely defined as a key/value store where one key can map to one or more values.

Although early in its life Cassandra was just a key/value store, it has evolved into much more. It is

now commonly seen as a hybrid containing common properties of two types of databases: a key/valu store and a row store. Unlike a relational database management system (RDBMS), Cassandra ColumnFamilys (similar to relational tables) do not need to have matching columns within a row. Even rows within a ColumnFamily are not required to always follow the same naming schema. The options are available, but data patterns are not strictly enforced. Data can also be added in very high volumes at very high velocities, and Cassandra will determine the correct version of a piece of data b resolving the timestamp at which it was inserted into the system.

Architecturally, its decentralized nature allows for no single point of failure and ensures that every node in the cluster has the same role. This means that every node in the cluster can serve any request Cassandra also supports replication and multi-data-center replication. Since replication strategies are configurable, you can set up your distribution architecture to be as centralized or spread out, or as redundant or fail-safe, as you would like. Because data is automatically replicated to nodes, downed faulty nodes are easily replaceable. New nodes can be added at will, without downtime, to increase read and write throughput or even just availability. The consistency levels are tunable, which allows you to have the application enforce the amount of resources applied to data assurance at a transaction level.

Cassandra also has an ecosystem being built around it. There are monitoring systems like OpsCenter to help you see the health of your cluster and manage common administration tasks. There are drivers for many of the major languages. Cassandra now comes with integration points for Hadoo and MapReduce support, full text search with Solr, and Apache Pig and Hive support. There is even a SQL-like query language called CQL, or Cassandra Query Language, to help in the data modeling an access patterns.

History of Cassandra

Apache Cassandra was originally developed at Facebook in 2008 to power Facebook's in-box search feature. The original authors were Avinash Lakshman, who also is one of the authors of the Amazon Dynamo paper, and Prashant Malik. After being in production at Facebook for a while, Cassandra wa released as an open-source project on Google Code in July of 2008. In March of 2009, it was accepte to the Apache Foundation as an incubator project. In February of 2010, it became a top-level Apache project.

As of the time of this writing, the most recent version of Apache Cassandra is the 1.2 series. Cassandra has come a long way since the first major release after its graduation to a top-level Apach project. It has picked up support for Hadoop, text search integration through Solr, CQL, zero-downtime upgrades, virtual nodes (vnodes), and self-tuning caches, just to name a few of the major features. Cassandra is still in constant heavy development, and new features are always being added and tested.

Note

The central paper on Cassandra, written by the primary Facebook engineers, is called "Cassandra—A Decentralized Structured Storage System" and is available at www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf.

Schema-less (If You Want)

Cassandra ColumnFamilys are considered schema-less. This means that you do not need to define a schema ahead of time. If you want to add a column, you simply specify the column name at write-tim and the column will be created if it doesn't exist. This lends itself to allowing for extremely wide rows, even rows that have millions of columns. Additionally, rows do not need to contain all or even any of the columns that other rows in the same table contain.

Cassandra does give you the option to create a schema, however. If you know what your data structure looks like, you can add column names and specify default types for those columns. This als enables you to add secondary indexes for the columns that you know about.

Who Uses Cassandra?

Cassandra is in wide use around the world, and usage is growing all the time. Companies like Netflix eBay, Twitter, Reddit, and Ooyala all use Cassandra to power pieces of their architecture, and it is critical to the day-to-day operations of those organizations. To date, the largest publicly known Cassandra cluster by machine count has over 300TB of data spanning 400 machines.

Because of Cassandra's ability to handle high-volume data, it works well for a myriad of applications. This means that it's well suited to handling projects from the high-speed world of advertising technology in real time to the high-volume world of big-data analytics and everything in between. It is important to know your use case before moving forward to ensure things like proper deployment and good schema design.

Is Cassandra Right for Me?

This isn't a very easy question to answer. Using Cassandra requires thinking in a different way about how you store data. While there are rows and columns, Cassandra is, at its base, a key/value store. There is no built-in full text search; there are no B-tree indexes or data manipulation functions.

One of the biggest differences between Cassandra and standard SQL RDBMSs is that there are no manipulation functions. These include `SUM`, `GROUP`, `JOIN`, `MAX`, `MIN`, and any other method you would use to modify the data at query time.

While deciding if Cassandra is a good fit for your use case, know that a lot of data manipulation ca be achieved at write-time rather than read-time. This, of course, means that you will be storing different views of the same data in multiple places. This is not necessarily a bad thing.

One example of this is to use counter columns where you would need aggregation. This is as easy a incrementing a value for each of the different ways you want to see your data. This pattern does require that you know what questions you want to ask ahead of time; if you need ad hoc data analysis in real time, Cassandra may not be the right fit.

Cassandra Terminology

In order to understand Cassandra, a good place to start is the vocabulary.

Cluster

A cluster is two or more Cassandra instances working together. These instances communicate with each other using the Gossip protocol.