ADDISON WESLEY DATA & ANALYTICS SERIES

PRACTICAL Cassandra

A Developer's Approach

RUSSELL BRADBERRY
ERIC LUBOW

Practical Cassandra

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

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

❖

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.

❖

This page intentionally left blank

Contents

Foreword by Jonathon Ellis

I was excited to learn that *Practical Cassandra* would be released right at my five-year anniversary of working on Cassandra. During that time, Cassandra has achieved its goal of offering the world's most 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 administering 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 a 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 Chair

This page intentionally left blank

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 are 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 of Cassandra.

—Paul Dix, Series Editor

This page intentionally left blank

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 for the cluster, model your data, and monitor and troubleshoot any issues. Toward the end of the book, we provide sample code, in-depth detail as to how Cassandra works under the covers, and real-world case 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 book 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 depth 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 the 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 happens 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 while 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/.

This page intentionally left blank

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 it 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 Bryan 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

This page intentionally left blank

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. He 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.

This page intentionally left blank

8

Monitoring

As the old systems adage goes, a service doesn't exist unless it's monitored. In this chapter, we will cover the basics of monitoring Cassandra. These include file-based logging, inspection of the JVM, and monitoring of Cassandra itself.

Logging

Under the covers, Cassandra uses the standard Java log library Log4j. Log4j is another Apache project that enables the capability to control the granularity of log statements using a configuration file. If you want to find out more about what is happening on a particular node than what `nodetool` and JMX MBeans (which we will cover in more detail later in the chapter) are telling you, you can change the logging levels.

As a front end to the Log4j back end, Cassandra uses Simple Logging Façade for Java (SLF4J). The logging levels from least verbose to most verbose are

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`
- `FATAL`

Understanding these logging levels is important not only to help monitor what is going on in the system or on a particular node but also to help troubleshoot problems. In troubleshooting complex systems such as Cassandra, Cassandra's `nodetool`, logging, and even the JMX MBeans can lead to red herrings. So it is necessary to compile as much information pertinent to the problem as possible to help diagnose what might be going on.

Taking a look at a normal healthy Cassandra node's system.log, you will see `INFO` lines that refer to various stages of the system executing their tasks. These include MemTable flushes, HintedHandoffs, and compactions, just to name a few.

Changing Log Levels

If you want to make any changes to the logging schema, you will need to find the log4j-server.properties file. The default logging level for Cassandra and the rootLogger is INFO. This level provides a standard amount of information that is sufficient for understanding the general health of your system. It is definitely helpful to see what your system looks like, so you should do so while logging at the DEBUG level. Be sure not to leave Cassandra in DEBUG mode for production as the entire system will act noticeably slower. To change the standard logging level in Cassandra from INFO to DEBUG, change the line that looks like this:

```
log4j.rootLogger=INFO,stdout,R
```

to this:

```
log4j.rootLogger=DEBUG,stdout,R
```

Now your Cassandra node will be running in DEBUG mode. To change it back, just swap the INFO and DEBUG again. To show less logging, you can change the logging level to WARN, ERROR, or FATAL.

Example Error

It is worth noting that not all problem messages enter the logs as WARNING or higher (higher meaning toward FATAL). Listing 8.1 presents an example of when things start to go south. This is a common set of log messages that you may see with your system set at INFO level. Even with the logging level set to INFO, there is a lot of useful information in the logs. Don't be afraid to keep a regular eye on the logs so you know what patterns of log messages are normal for your system. For example, if things are starting to slow down, you may see something like Listing 8.1.

Listing 8.1 INFO **Messages That Show Mutation and** READ **Messages Dropped**

```
INFO [ScheduledTasks:1]  2012-07-09 20:48:57,290  MessagingService.java (line 607)
3476 MUTATION messages dropped in last5000ms
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,290 MessagingService.java (line 607)
677 READ messages dropped in last 5000ms
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,291 StatusLogger.java (line 50)
Pool Name                      Active                 Pending       Blocked
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,291 StatusLogger.java (line 65)
ReadStage                           32                     621             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,291 StatusLogger.java (line 65)
RequestResponseStage                 0                       0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,291 StatusLogger.java (line 65)
ReadRepairStage                      0                       0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,291 StatusLogger.java (line 65)
MutationStage                       32                    4105             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,291 StatusLogger.java (line 65)
ReplicateOnWriteStage                0                       0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
GossipStage                          0                       0             0
```

```
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
AntiEntropyStage                  0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
MigrationStage                    0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
StreamStage                       0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
MemtablePostFlusher               0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
FlushWriter                       0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
MiscStage                         0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,292 StatusLogger.java (line 65)
InternalResponseStage             0                         0             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,293 StatusLogger.java (line 65)
HintedHandoff                     1                         8             0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,293 StatusLogger.java (line 70)
CompactionManager                 0                         0
 INFO [ScheduledTasks:1] 2012-07-09 20:48:57,293 StatusLogger.java (line 82)
MessagingService                n/a                     0,0
```

When you see messages being dropped, as in the first two lines of the listing, that's a sign that your system is under stress. Depending on your application requirements, some level of dropped messages may be acceptable. But regardless of whether or not your application can tolerate running in a degraded state, the overall health of your cluster (or at the very least this node) is in question. Most applications are capable of handling dropped `READ` requests, but dropped `MUTATION` messages mean that data that should have been written isn't getting written. Depending on the consistency level of the write in question, this could mean the write didn't happen at all, or it could mean the write didn't happen on this node. Also notice that the `ReadStage` and `MutationStage` lines have multiple `Active` and `Pending` messages left to work on. The reason these messages are dropped is that Cassandra wants to do its best to keep up with the volume of work that it is being given.

There are other such common log lines to watch for, which can be done via a log monitor. One method for monitoring the logs programmatically using Nagios will be discussed later in this chapter.

JMX and MBeans

Built into Cassandra and the JVM is the capability to use the JMX, or Java Management Extensions. In other words, using JMX gives you the capability to manage your servers remotely or check into settings programmatically, including the memory, CPU, threads, Gossip, or any other part of the system that has been instrumented in JMX. Instrumentation is what enables the application to provide application-specific

information to be collected by external tools. JMX also gives you the ability to control certain aspects of this information.

JConsole

MBeans, or Managed Beans, are a special type of JavaBean that makes a resource inside the application or the JVM available externally. The standard tool that ships with Java for managing the MBeans is JConsole.

The most common use case for accessing a Cassandra server is that the server will be remote and you probably won't have console access to it. It is also highly recommended that you run JConsole remotely as it is a heavy user of resources on a machine and can steal those resources away from the Cassandra node. If this is the case, you can use SSH tunneling to bring up JConsole. When you SSH, be sure to use the `-X` switch to ensure that X11 forwarding is on. This is what enables you to use JConsole over the network. After SSHing into the machine running Cassandra and finding the JConsole binary, just execute it as you would any normal binary. Assuming everything is configured correctly, you will get a JConsole login window as shown in Figure 8.1.

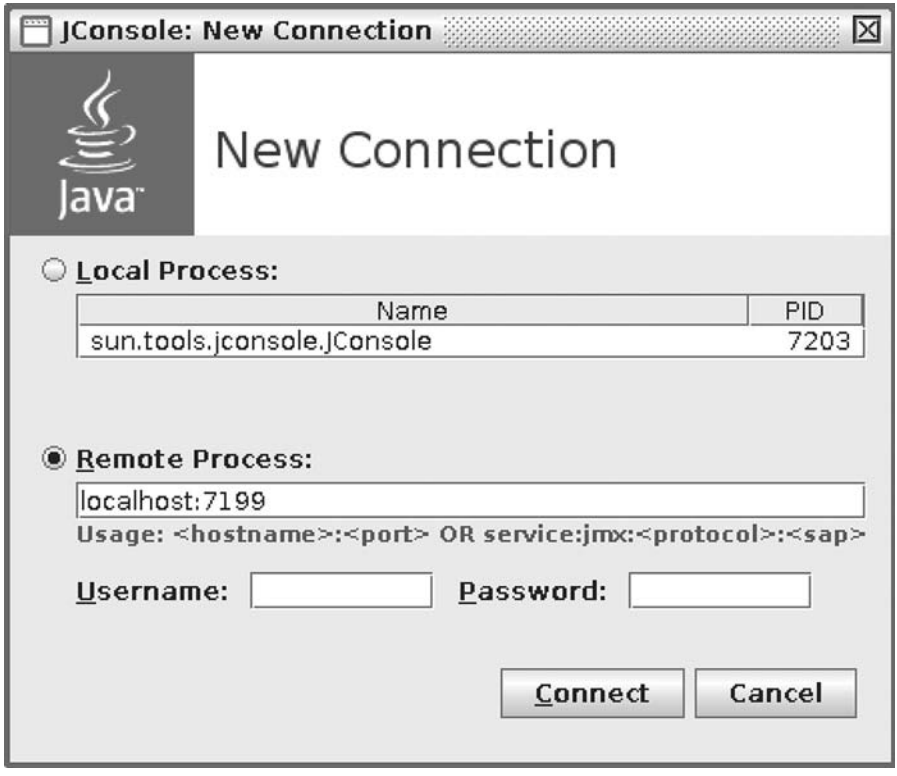

Figure 8.1 JConsole login window when logging in via localhost

Click on the radio button labeled Remote Process and type `localhost:7199`. If you have Cassandra set up with authentication, you will need to put in the username and password as well. Port 7199 is the default JMX port for Cassandra. The first thing you will notice once a connection has been established is that there are multiple tabs that contain information for you to look through. These tabs are Overview, Memory, Threads, Classes, VM Summary, and MBeans.

The Overview, Memory, and Threads tabs are sets of graphs that provide insight into the current state of the system. The Classes graph is simply a graph of how many classes are loaded into the JVM at the current time (or a different time range that you choose). The VM Summary is an overview of what the current view of the Java Virtual Machine is.

Memory

The Memory tab consists of graphs about the current state of the memory (see Figure 8.2). One of the most important memory stats that you want to be aware of is your current heap usage. It is also the default graph that comes up on the Memory tab. As with everything else in your system, it is helpful to know what a good baseline is for heap usage during normal system operations. If you have a 10GB heap set and you find during most of your operations that you are using only 3GB, you can likely reduce your JVM maximum heap size. This will enable you to reduce your memory footprint on the system and possibly even speed up your GCs.

Cassandra also does a lot of work off-heap. Things like bloom filters in version 1.2 and forward are off-heap now. Keeping tabs on what the off-heap memory usage looks like is also very important.

Garbage collection is also one of the metrics that can be viewed from the Memory tab. If needed (though typically not recommended unless you know what you are doing), you can even force a GC from the Memory tab in JConsole. In Cassandra 1.1 and later, there are even helpful bars that display the total memory used versus memory available both on-heap and off-heap.

Threads

The Threads tab in JConsole is dedicated to showing the current and peak usage patterns of various thread stages in Cassandra (see Figure 8.3). These include everything that you would normally see in the logs from things like the CommitLog handler and compaction scheduler all the way to garbage collection, Gossip, and streaming. It is also helpful here to see how many threads your system uses normally as well as under load.

MBeans

The final tab in JConsole is the tab for MBeans (see Figure 8.4). There are a lot of MBeans that are useful for assessing the state of your Cassandra node and your Cassandra cluster. You will notice that there are a few groupings here that can be expanded. Other than the standard Java MBeans that are available to every agent, there are several groupings specific to Cassandra. All of their class paths start with org .apache.cassandra.

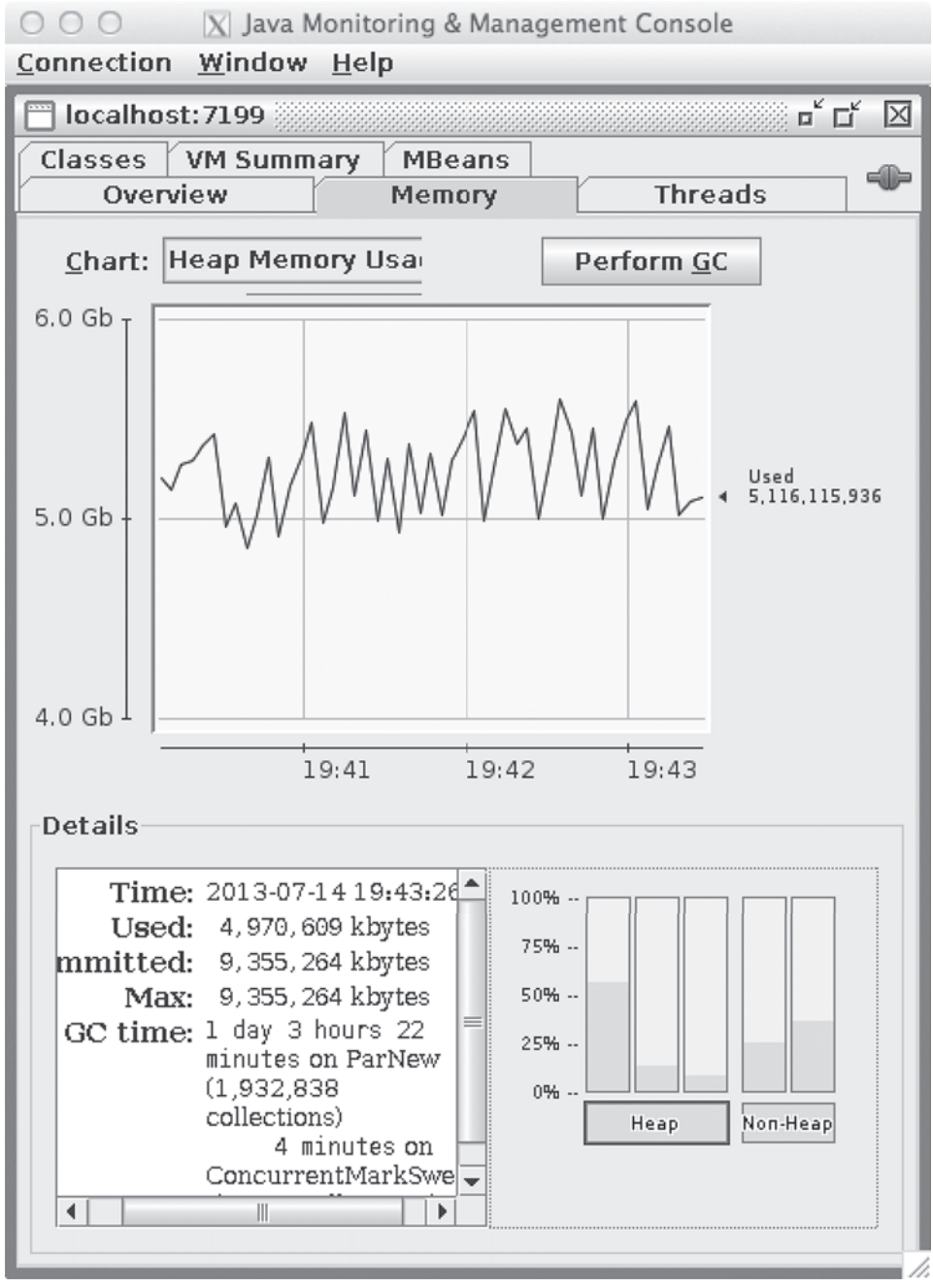

Figure 8.2 JConsole Memory tab displaying heap usage graphs

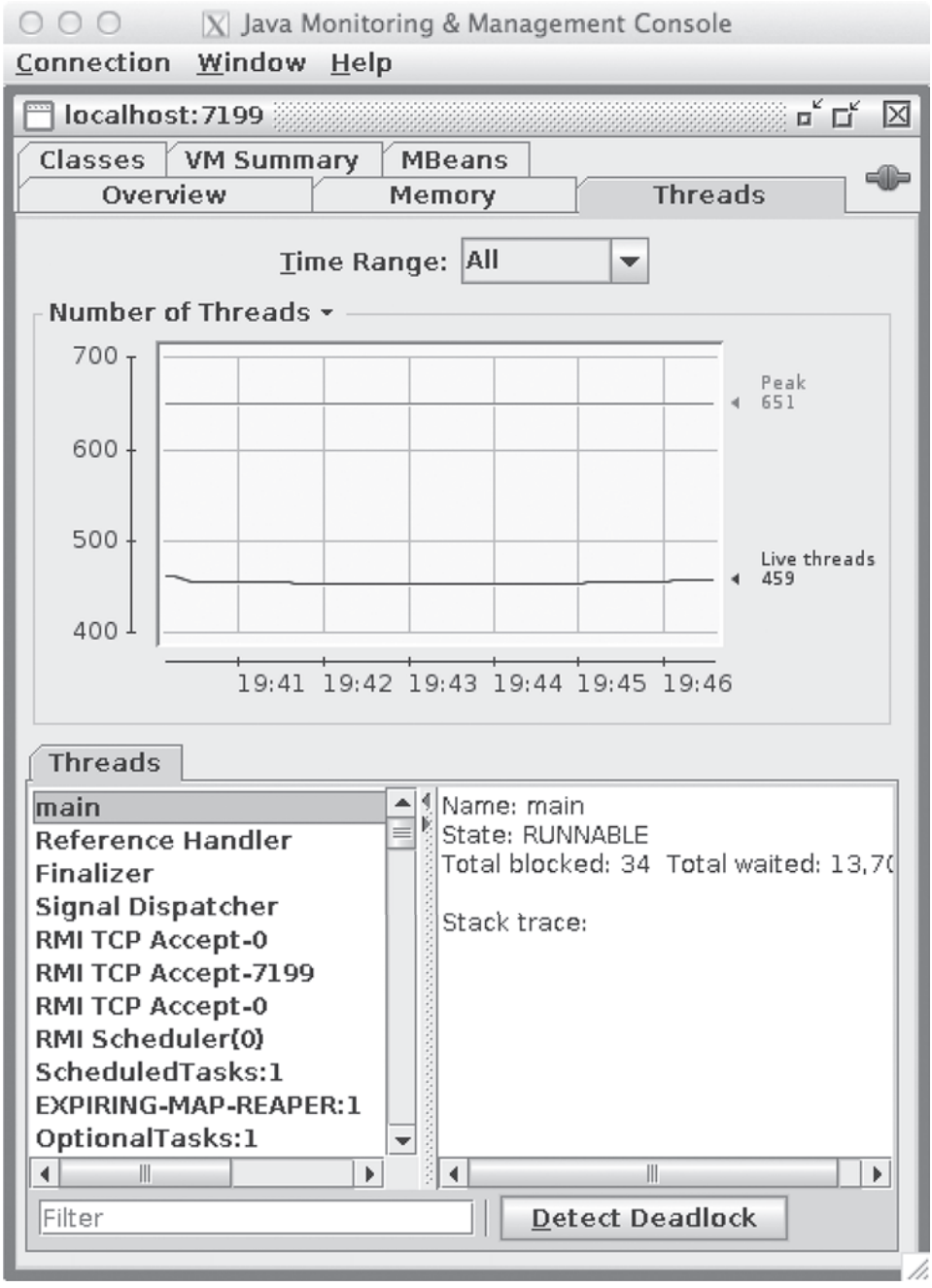

Figure 8.3 JConsole Threads tab displaying the number of threads Cassandra is currently using and general information on the main Cassandra thread

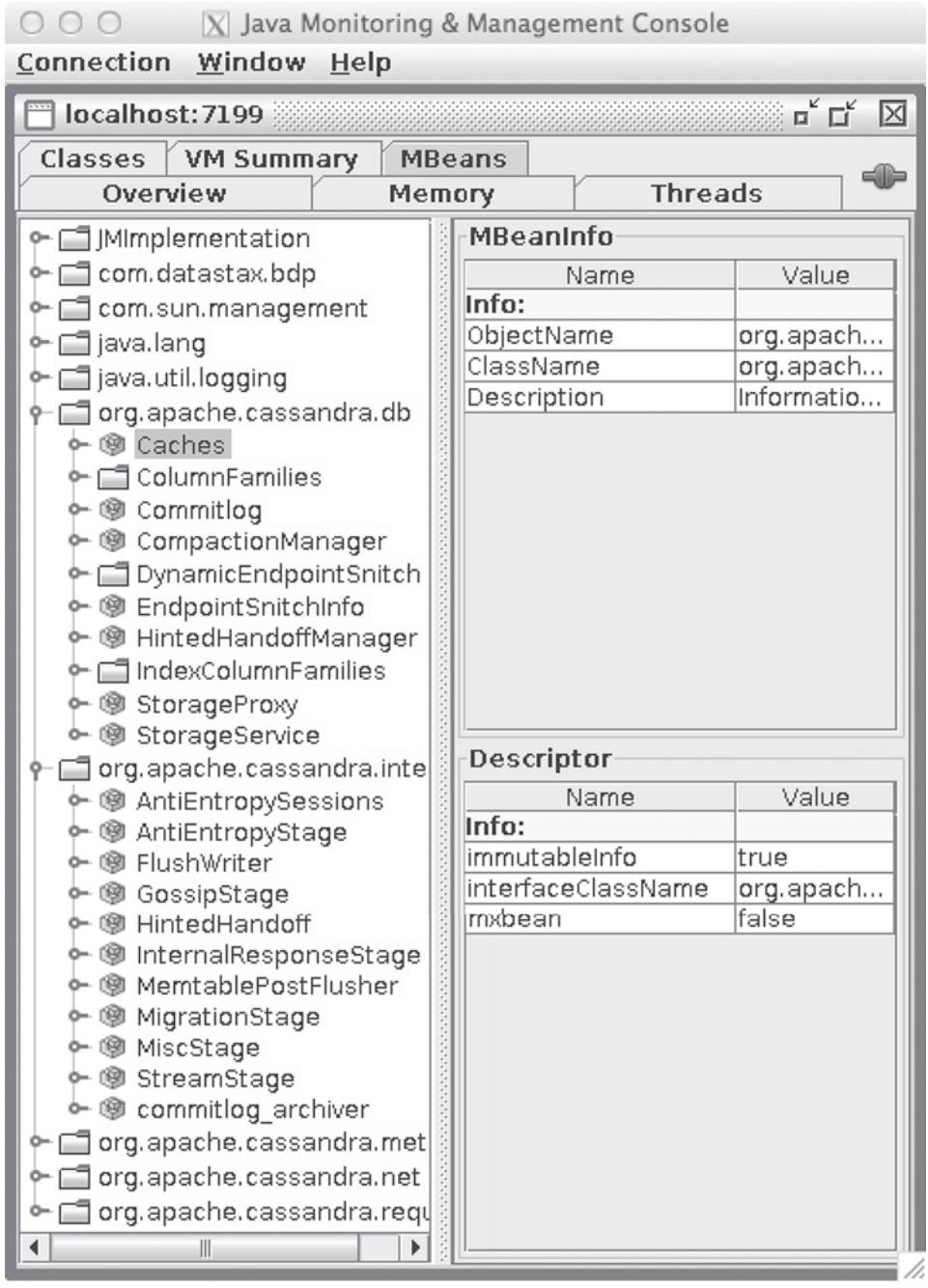

Figure 8.4 JConsole MBeans tab view with the Cassandra DB and Internal trees opened

There are a lot of MBeans provided by an application as complex as Cassandra. There is no need to cover all of them as you can easily explore them on your own using the JConsole interface.

As a high-level overview, they are broken down into the following categories:

- **DB.** The MBeans stored in the DB section cover everything about the actual data storage part of Cassandra. You can view information about the cache and CommitLogs, or even information about the individual ColumnFamilys that you have created in each of the keyspaces. HintedHandoffManager, EndPointSnitchInfo, and CompactionManager information can also be found here.
- **Internal.** In the Internal section, there are MBeans that cover the state and statistics around the staged architecture. More specifically, you can find information about the state of Gossip and HintedHandoff as opposed to just finding information about the managers as in the DB section.
- **Metrics.** The metrics available in this section are ClientRequestMetrics. These are things like read and write timeouts and `Unavailable` errors.
- **Net.** The network section houses information about internode communication. This includes information about the FailureDetector, Gossiper, MessagingService, and StreamingService.
- **Request.** The MBeans in the Request section are about tasks related to read, write, and replication.

When you select any MBean in the tree, its MBeanInfo and MBean descriptor are displayed on the right-hand side of the window. If any additional attributes, operations, or notifications are available, they will appear in the tree as well, below the selected MBean.

Each one of these sections of MBeans provides access to a large amount of information, giving you insight into both the system as a whole and the individual nodes. Familiarizing yourself with what is available to you as an admin will help you when it comes time to instrument JMX-level checks from within your monitoring system.

Health Checks

Using JConsole to monitor your system is tedious and good as a monitoring system only if you are actively staring at the graphs and information all the time. Since that is unrealistic and time-consuming, we recommend that you use other systems for monitoring the general health of your system such as Nagios.

Nagios

Nagios is open-source software dedicated to monitoring computers, networks, hosts, and services and can alert you when things are going wrong or have been resolved. It is extremely versatile and has the capability to monitor many types of services, applications, or parts of an application. Let's start at the bottom of the monitoring chain and work our way up. In order to avoid a complete lesson on monitoring, we will only cover the basics along with what the most common checks should be as they relate to Cassandra and its operation.

There are three primary alerts in Nagios: `WARNING`, `CRITICAL`, and `OK`. They mean exactly what they sound like. A `WARNING` alert is sent if the service in question is starting to show signs of a problem, such as a hard drive nearing capacity. A `CRITICAL` alert is sent if the service in question is down or in a catastrophic state, such as a hard drive that is completely out of space and preventing the applications using that drive from running. An `OK` alert is sent when the service has recovered or become available again, such as when the total space used on the hard drive has dropped below the threshold set to alert for `CRITICAL` or `WARNING`.

OS and Hardware Checks

When monitoring any machine, it's best to start out with the checks at the OS and hardware layer. Even if you are running Cassandra in a virtualized environment such as Amazon or Rackspace, there are still hardware(ish) checks that should be instituted.

Disks and Partitions

The first thing you are going to want to check is the amount of free disk space on data partitions and the CommitLog partitions (assuming they are on separate partitions). Remember that if you are using SizeTieredCompaction, you shouldn't have the alert set for `WARNING` at 80% disk utilization and `CRITICAL` set at 90% disk utilization. The safer approach is to set the `WARNING` threshold to be roughly 35% disk utilization and the `CRITICAL` threshold at 45% disk utilization. SizeTieredCompaction is capable of taking up two times the size of the largest SSTable on disk. And while it is unlikely that a single SSTable would be 50% of the data on disk, it is better to be safe than sorry. Recovering from having too much data on disk is extremely difficult.

This concept of monitoring partitions and drives is also important because of JBoD support in Cassandra 1.2 and later. This means that Cassandra can have a single data directory on multiple disks. You will need to know if one or more of those disks are having an issue or require replacement. By monitoring the utilization and health of all the disks in your system, you will know their state and whether they need replacing or maintenance.

Last, you want to ensure that the drive that contains the log files doesn't fill up. Depending on your log settings, Cassandra has the potential to be very verbose in the log files. If the log files become too large, they can prevent the rest of your system from working if the drive(s) runs out of space.

Swap

Linux divides its physical memory into smaller chunks called pages. Swapping is the process whereby a page of memory is copied from memory to a dedicated space on the hard disk called swap space to free up that page of memory. Although there are cases where it is OK, it is normally not recommended for systems to be in a state where they are swapping memory. Typically, anything more than 5% to 10% of your swap space being used is cause for investigation.

On a Cassandra node, swapping is usually a bad sign, so you will want to monitor the swap partition for usage of nearly any kind. Since you should be able to hold the entire JVM's heap space in memory with at least a little room to spare for the operating system,

getting to the point of swapping out pages of memory means it might be a little too late to recover. One of the reasons Cassandra is able to function so well with regard to writes is the fact that many of the writes occur to the memory-mapped MemTables. Having these MemTables swap to disk would drastically impair the performance of Cassandra and should therefore be avoided when possible.

Clock Drift

Clock drift refers to the phenomenon where one clock does not run at the exact same speed as another clock. It is especially important to be aware of this if you are running in a virtualized environment as drift from the hypervisor can be much more prevalent than on regular iron. The system clock is incredibly important to Cassandra's write and reconciliation architecture. Most writes are serialized by timestamp. In other words, if two writes come in for the same column at almost the same time, the determining factor for which value wins is which timestamp is higher. If the system clocks in the ring are not all in sync, you are probably going to see some really strange behavior.

One of the ways to deal with that is to monitor the clock drift using NTP. NTP, or Network Time Protocol, is the most commonly used time synchronization system on the Internet. It also comes with a binary for telling you the offset (drift) from its synchronizing time server. You obviously want to minimize the amount of drift your system experiences. But there will invariably be some that you have to deal with. Monitoring is the way you know if the NTP daemon isn't doing the job it is supposed to be doing and keeping your clocks in sync. Being alerted to a problem with the clocks in a distributed environment that relies heavily on time for decision making could save a lot of time tracking down weird problems later on.

Ping Times

It is also a good idea to check the ping time responses from each of the Cassandra nodes being monitored. There are any number of reasons that these responses can begin to come back slowly. A few examples include the following:

- A machine that is doing too much work and running short of CPU cycles to respond quickly
- I/O saturation, too high an await (average wait) time, and the machine cannot respond quickly to the request
- Network saturation due to unthrottled streaming on a high-speed network link

Whatever the reason is, it is good to know if there is network congestion of which you should be aware. When a node is slow to receive packets (which is the case with nodes with high ping times), writes can be slow to come in and register, reads and writes will be dropped to keep up with the demand being put on the system, or any number of other weird behaviors may appear. What constitutes a high ping time from your monitoring server depends to a great extent on your network paths. Run a few ping tests from your monitoring server to your Cassandra nodes during regular usage periods to get a feel for what a normal threshold is.

CPU Usage

Cassandra is usually an I/O-bound system. You usually run into problems with disk writes or reads slowing down long before you run into CPU-related slowdown. But just to be safe, as different workloads call for different tools to be used at different times, you should monitor CPU usage. While there are many things you could look for when monitoring CPU usage, such as context switches or interrupt requests, a good place to start is usually watching the system load average. The system load average is an average of the number of processes waiting to get into the system's run queue over a period of time. In the case of the `uptime` command, it's over one, five, and 15 minutes. Keep in mind that in the case of multiprocessor systems, the load is relative to the number of processors and cores on the system.

The common rule for utilization is that you want to have a machine working hard but not overworking. This means that you typically want to have the machine running at about 70% utilization. That leaves you headroom for spikes in work and doesn't leave the machine underutilized during slower periods. So if you have four cores, having the load sit at around 3.00 is usually a safe bet. If you have four cores and the load is 3.5 or higher, you should try to find out what's wrong and fix it before things go from bad to worse.

Cassandra-Specific Health Checks

Once you have the basic system checks in place, it's time to add monitoring that is specific to Cassandra. There are various checks that interact with Cassandra at different levels of the system. Some are superficial such as checking to see if ports are alive and being listened on. Some checks require using a slightly more in-depth toolset to programmatically check the MBeans described earlier.

Ports

There are three primary ports of interest to Cassandra: 7000 (or 7001 if SSL/TLS is enabled), 7199, and 9160. Port 7000/7001 is used by Cassandra for cluster communication. This includes things such as the Gossip protocol and failure detection. Port 7199 is used by JMX. Port 9160 is the Thrift port and is used for client communication. In order for your cluster to function properly, all of these ports should be accessible.

While it is not necessary to specifically monitor these ports, it is a good idea to test them out one way or another. Testing the Thrift port (9160) is just testing whether you can connect to an instance using a Cassandra driver. In terms of monitoring, if you can connect, the check passes. If you can't connect to the server, the check should send off an alert. You can also use a simple TCP check here even though it is less comprehensive.

JMX Checks

Using some of the knowledge we gained from looking at the normal behavior of our system with JConsole, we are going to add some checks using JMX. There are plug-ins for Nagios that enable you to run JMX queries and compare the results against a set of predetermined thresholds. While there are many values that can be monitored through JMX, there are a few that stand out.

The first set of JMX checks to create is for read and write request latency. These values are given in microseconds because they should be that small. These latencies can be measured at the Cassandra application level and/or at the ColumnFamily level. Measuring them at the application level is important as a general health metric. High request latencies can be indicative of a bad disk or that your current read pattern is starting to slow down. If there is a ColumnFamily for which it is particularly important to have extremely low-latency reads and/or writes, it would be a good decision to monitor the performance for that ColumnFamily as well. It is important to note that read latency and write latency are two separate metrics provided by Cassandra, and both are important in their own right depending on your workload.

The next set of JMX metrics to keep tabs on is garbage collection timing. Cassandra will not only tell you how long its last garbage collection took but also how long that last ParNew GC took. A good way to think of ParNew garbage collection is that it is a stop-the-world garbage collection that uses multiple GC threads to complete its job. If you are monitoring the amount of time these take, you can easily set up an alert for when they start to take too long. Cassandra is unavailable during a stop-the-world garbage collection pause. The longer these pauses take, the longer Cassandra will be unavailable.

Another metric that is useful in helping to determine whether or not you need to add capacity to your cluster is PendingTasks under the CompactionManagerMBean. Depending on the speed and volume with which you ingest data, you will need to find a comfortable set of thresholds for your system. Typically, the number of PendingTasks should be relatively low, as in fewer than 50 at any given time. There are certainly acceptable reasons for things to back up, such as forced compactions or cleanup, but it is advisable to watch this metric carefully. If you have an alert set for PendingTasks and find this alert firing regularly, you may need to add more capacity (either more or faster disks or more nodes) to your cluster to keep up with the workload.

The last JMX metrics that should make it onto your first round of monitoring are the amount of on-heap and the amount of off-heap memory used at a time. The amount of on-heap memory used should always be less than the amount of heap that you have allowed the JVM to allocate. Since you know what this value is at start time, you should be able to easily monitor whether or not you are approaching that value. Off-heap memory tracking is a little harder to monitor for sane values. This is a metric where you will once again have to take a look at JConsole and see what regular and peak values are for the system under normal and peak operational loads so you don't send off useless alerts.

Log Monitoring

There is a lot of useful information in the Cassandra logs that can be indicative of a problem. As mentioned earlier in the chapter, you can find `READ` and `WRITE` dropped message counts within the `INFO` log level. There is a Nagios plug-in that can monitor logs and check for specific log messages. Using this plug-in, you can have Nagios alert you not just when there are `READ` and/or `WRITE` messages dropped, but you also can have it alert you when this happens more than *n* times per period. For instance, your application may be tolerant of missing `READ`s and much less tolerant of missing `WRITE`s. So the log monitoring check can alert you with a `CRITICAL` alert if more than 1,000 mutations

have been dropped over a five-minute period and with a `WARNING` alert if more than 1,000 mutations have been dropped over a 15-minute period.

This is just in the case of bad things happening in the `INFO` level. You can also have the log monitoring system alert you if any `FATAL`, `ERROR`, or `WARNING` log messages are put into the logs. Many of these plug-ins are configurable enough to send the log messages (or at least the one that caused the notification) along with the alert.

Cassandra Interactions

Now that we have the OS and system layer monitored and we know Cassandra is up and at least responding, it's time to check a little deeper. The further into the application you monitor, the better you will be able to sleep at night knowing things are functioning the way you want them to. Although it is useful and necessary to have superficial checks like load average and memory, the real value of monitoring systems is realized as you get deeper into the application.

What this means is that you should be checking things that are specific to your application in addition to the Cassandra server. If your application writes to a new ColumnFamily at the beginning of every month, you should have your monitoring system check before the month turnover that the new ColumnFamily exists (and optionally create it if it doesn't).

Another good use of monitoring resources is to check the response time of certain queries. If you are regularly running queries that roll up all the events for an hour, monitor how long that query takes to run and set up an alert if it's outside the normal threshold. In other words, if the query runs too fast, you want to know because it's possible you aren't collecting all the data you expect to be there. If the query takes too long to run, your system could be under heavy load or you may have just hit a point where you need to rethink your query patterns. Either way, that type of instrumentation is useful to measure how your system actually performs compared to how you expect it to perform.

If you run an application at the top of every hour—an extract, transform, load (ETL) process, for example—it might be a good idea to have the application put a "run complete" column somewhere when it's done. At the beginning of every hour, the monitoring system can run a query to check for the existence of the column for the last hour. If the "run complete" column doesn't exist for the last hour, it would be good to know so you can look into why.

Summary

There are many tools available for building monitoring systems. Nagios is just one of the common general-purpose monitoring tools. As long as some application is checking on the health and availability of your system and letting you know when an issue is present, or about to present itself, you will be in good shape. There are also some good examples of how your main application and other parts of your application interact with Cassandra and can be instrumented to give you a feeling of total information awareness and potentially the ability to get a good night's sleep when it is all in production.

In this example, Nagios can act as an early-warning tool. It can give you a heads-up to look at the machine in question and dig deeper into a potential problem before it turns into something more serious such as a full-fledged outage or a completely downed node. Ensuring an intelligently set-up monitoring infrastructure is essential to having a well-designed and architected system.

This page intentionally left blank

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X