O'REILLY®

Compliments of Pivotal®

Data Warehousing with Greenplum

Open Source Massively Parallel Data Analytics

Marshall Presser

Pivotal Greenplum®

The World's First Open Source Massively Parallel Data Warehouse

The Pivotal Greenplum database is both architecturally and functionally different than any other open source data processing project or cloud database. Pivotal Greenplum utilizes massively parallel processing (MPP) technology and an innovative query optimizer to execute complex SQL analytics on very large data sets at speeds multiple times faster than other solutions.

Highly Scalable and Expandable

Available on premise, private cloud or public cloud

Supporting powerful in-database data science and operational analytics

Learn more at **pivotal.io/pivotal-greenplum**
Source code fully downloadable at **greenplum.org**

Pivotal®

Data Warehousing with Greenplum

Open Source Massively Parallel Data Analytics

Marshall Presser

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Data Warehousing with Greenplum
by Marshall Presser

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Shannon Cutt
Production Editor: Kristen Brown
Copyeditor: Octal Publishing, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

May 2017: First Edition

Revision History for the First Edition
2017-05-30: First Release

978-1-491-98350-8

[LSI]

Table of Contents

Foreword

In the mid-1980s, the phrase "data warehouse" was not in use. The concept of collecting data from disparate sources, finding a historical record, and then integrating it all into one repository was barely technically possible. The biggest relational databases in the world did not exceed 50 GB in size. The microprocessor revolution was just getting underway, and two companies stood out: Tandem, who lashed together microprocessors and distributed Online Transaction Processing (OLTP) across the cluster; and Teradata, who clustered microprocessors and distributed data to solve the big data problem. Teradata named the company from the concept of a terabyte of data —1,000 GB—an unimaginable amount of data at the time.

Until the early 2000s Teradata owned the big data space, offering its software on a cluster of proprietary servers that scaled beyond its original 1 TB target. The database market seemed set and stagnant with Teradata at the high end; Oracle and Microsoft's SQL Server product in the OLTP space; and others working to hold on to their diminishing share.

But in 1999, a new competitor, soon to be renamed Netezza, entered the market with a new proprietary hardware design and a new indexing technology, and began to take market share from Teradata.

By 2005, other competitors, encouraged by Netezza's success, entered the market. Two of these entrants are noteworthy. In 2003, Greenplum entered the market with a product based on PostgreSQL, that utilized the larger memory in modern servers to good effect with a data flow architecture, and that reduced costs by deploying on commodity hardware. In 2005, Vertica was founded based on a major reengineering of the columnar architecture first

implemented by Sybase. The database world would never again be stagnant.

This book is about Greenplum, and there are several important characteristics of this technology that are worth pointing out.

The concept of flowing data from one step in the query execution plan to another without writing it to disk was not invented at Greenplum, but it implemented this concept effectively. This resulted in a significant performance advantage.

Just as important, Greenplum elected to deploy on regular, nonproprietary hardware. This provided several advantages. First, Greenplum did not need to spend R&D dollars engineering systems. Next, customers could buy hardware from their favorite providers using any volume purchase agreements that they already might have had in place. In addition, Greenplum could take advantage of the fact that the hardware vendors tended to leapfrog one another in price and performance every four to six months. Greenplum was achieving a 5 to 15 percent price/performance boost several times a year—for free. Finally, the hardware vendors became a sales channel. Big players like IBM, Dell, and HP would push Greenplum over other players if they could make the hardware sale.

Building Greenplum on top of PostgreSQL was also noteworthy. Not only did this allow Greenplum to offer a mature product much sooner, it could use system administration, backup and restore, and other PostgreSQL assets without incurring the cost of building them from scratch. The architecture of PostgreSQL, which was designed for extensibility by a community, provided a foundation from which Greenplum could continuously grow core functionality.

Vertica was proving that a full implementation of a columnar architecture offered a distinct advantage for complex queries against big data, so Greenplum quickly added a sophisticated columnar capability to its product. Other vendors were much slower to react and then could only offer parts of the columnar architecture in response. The ability to extend the core paid off quickly, and Greenplum's implementation of columnar still provides a distinct advantage in price and performance.

Further, Greenplum saw an opportunity to make a very significant advance in the way big data systems optimize queries, and thus the ORCA optimizer was developed and deployed.

During the years following 2006, these advantages paid off and Greenplum's market share grew dramatically until 2010.

In early 2010, the company decided to focus on a part of the data warehouse space for which sophisticated analytics were the key. This strategy was in place when EMC acquired Greenplum in the middle of that year. The EMC/Greenplum match was odd. First, the niche approach toward analytics and away from data warehousing and big data would not scale to the size required by such a large enterprise. Next, the fundamental *shared-nothing* architecture was an odd fit in a company whose primary products were shared storage devices. Despite this, EMC worked diligently to make a fit and it made a significant financial investment to make it go. In 2011, Greenplum implemented a new strategy and went "all-in" on Hadoop. It was no longer "all-in" on the Greenplum Database.

In 2013, EMC spun the Greenplum division into a new company, Pivotal Software. From that time to the present, several important decisions were made with regard to the Greenplum Database. Importantly, the product is now open sourced. Like many open source products, the bulk of the work is done by Pivotal, but a community is growing. The growth is fueled by another important decision: to reemphasize the use of PostgreSQL at the core.

The result of this is a vibrant Greenplum product that retains the aforementioned core value proposition—the product runs on hardware from your favorite supplier; the product is fast and supports both columnar and tabular tables; the product is extensible and Pivotal has an ambitious plan in place that is feasible.

The bottom line is that the Greenplum Database is capable of winning any fair competition and should be considered every time.

I am a fan.

— Rob Klopp
Ex-CIO, United States Social Security Administration
Author of The Database Fog Blog

Preface

Why Are We Writing This Book?

When we at Pivotal decided to open-source the Pivotal Greenplum Database, we decided that an open source software project should have more information than that found in online documentation. As a result, we should provide a nontechnical introduction to Greenplum that does not live in a vendor's website. Many other open source projects, especially those under the Apache Software Foundation, have books, and Greenplum is an important project, so it should, as well. Our goal is to introduce the features and architecture of Greenplum to a wider audience.

Who Are the "We"?

Marshall Presser is the lead author of this book, but many others at Pivotal have contributed content, advice, editing, proofreading, suggestions, topic, and so on. Their names are listed in the Acknowledgments section and, when appropriate, in the sections to which they have written extensively. It might take a village to raise a child, but it turns out that it can take a crowd to source a book.

Who Is the Audience?

Anyone with a background in IT, relational database, big data, or analytics can profit from reading this book. It is not designed for experienced Greenplum users who are interested in the more technical features or those expecting detailed technical discussion of optimal query and loading performance, and so on. We provide

pointers to more detailed information if you're interested in a deeper dive.

What the Book Covers

This book covers the basic features of the Greenplum Database, beginning with an introduction to the Greenplum architecture and then describing data organization and storage; data loading; running queries; and doing analytics in the database, including text analytics. In addition, there is material on monitoring and managing Greenplum, deployment options as well as some other topics.

What It Doesn't Cover

We won't be covering query tuning, memory management, best practices for indexes, adjusting the collection of database parameters (known as GUCs), or converting to Greenplum from other relational database systems. These are all valuable topics. They are covered elsewhere and would bog down this introduction with too much detail.

Where You Can Find More Information

At the end of each chapter, there is a section pointing to more information on the topic.

How to Read This Book

It's been our experience that a good understanding of the Greenplum architecture goes a long way. An understanding of the basic architecture makes the sections on data distribution and data loading seem intuitive. Conversely, a lack of understanding of the architecture will make the rest of the book more difficult to comprehend. We would suggest that you begin with Chapter 1 and Chapter 3 and then peruse the rest of the book as your interests dictate. If you prefer, you're welcome to start at the beginning and work your way in linear order to the end. That works, too!

Acknowledgments

I owe a huge debt to my colleagues at Pivotal who helped explicitly with this work and from whom I've learned so much in my years at Pivotal and Greenplum. I cannot name them all, but you know who you are.

Special callouts to the section contributors (in alphabetical order by last name):

- Oak Barrett for "Greenplum Management Utilities" on page 61
- Kaushik Das for "Data Science on Greenplum with Apache MADlib" on page 39
- John Knapp for Chapter 7, *Integrating with Real-Time Response*
- Frank McQuillan for "Data Science on Greenplum with Apache MADlib" on page 39
- Tim McCoy for "Greenplum Command Center" on page 55
- Venkatesh Raghavan for Chapter 8, *Optimizing Query Response*
- Craig Sylvester and Bharath Sitaraman for "Text Analytics" on page 47
- Bharath Sitaraman and Oz Basarir for "Greenplum Workload Manager" on page 60

Other contributors, reviewers, and colleagues:

- Jim Campbell, Craig Sylvester, Venkatesh Raghavan, and Frank McQuillan for their yeoman work in reading the text and helping improve it no end.
- Cesar Rojas for encouraging Pivotal to back the book project.
- Jacque Istok and Dormain Drewitz for encouraging me to write this book.
- Ivan Novick especially for the Greenplum list of achievements and the Agile development information.
- Elisabeth Hendrickson for her really useful content suggestions.
- Jon Roberts, Scott Kahler, Mike Goddard, Derek Comingore, Louis Mugnano, Rob Eckhardt, Ivan Novick, and Dan Baskette for the questions they answered.

Other commentators:

- Stanley Sung
- Amil Khanzada
- Jianxia Chen
- Kelly Indrieri
- Omer Arap

And, especially, Nancy Sherman, who put up with me while I was writing this book and encouraged me when things weren't going well.

CHAPTER 1

Introducing the Greenplum Database

Problems with the Traditional Data Warehouse

Sometime near the end of the twentieth century, there was a notion in the data community that the traditional relational data warehouse was floundering. As data volumes began to increase in size, the data warehouses of the time were beginning to run out of power and not scaling up in performance. Data loads were struggling to fit in their allotted time slots. More complicated analysis of the data was often pushed to analytic workstations, and the data transfer times were a significant fraction of the total analytic processing times. Furthermore, given the technology of the time, the analytics had to be run in-memory, and memory sizes were often only a fraction of the size of the data. This led to sampling the data, which can work well for many techniques but not for others, such as outlier detection. Ad hoc queries on the data presented performance challenges to the warehouse. The database community sought to provide responses to these challenges.

Responses to the Challenge

One alternative was *NoSQL*. Advocates of this position contended that SQL itself was not scalable and that performing analytics on large datasets required a new computing paradigm. Although the

NoSQL advocates had successes in many use cases, they encountered some difficulties. There are many varieties of NoSQL databases, often with incompatible underlying models. Existing tools had years of experience in speaking to relational systems. There was a smaller community that understood NoSQL better than SQL, and analytics in this environment was still immature. The NoSQL movement morphed into a *Not Only SQL* movement, in which both paradigms were used when appropriate.

Another alternative was Hadoop. Originally a project to index the World Wide Web, Hadoop soon became a more general data analytics platform. MapReduce was its original programming model; this required developers to be skilled in Java and have a fairly good understanding of the underlying architecture to write performant code. Eventually, higher-level constructs emerged that allowed programmers to write code in Pig or even let analysts use SQL on top of Hadoop. However, SQL was never as complete or performant as that in true relational systems.

In recent years, Spark has emerged as an in-memory analytics platform. Its use is rapidly growing as the dramatic drop in price of memory modules makes it feasible to build large memory servers and clusters. Spark is particularly useful in iterative algorithms and large in-memory calculations, and its ecosystem is growing. Spark is still not as mature as older technologies, such as relational systems.

Yet another response was the emergence of clustered relational systems, often called massively parallel processing systems. The first entrant into this world was Teradata in the mid-to-late 1980s. In these systems, the relational data, traditionally housed in a single-system image, is dispersed into many systems. This model owes much to the scientific computing world, which discovered MPP before the relational world. The challenge faced by the MPP relational world was to make the parallel nature transparent to the user community so coding methods did not require change or sophisticated knowledge of the underlying cluster.

A Brief Greenplum History

Greenplum took the MPP approach to deal with the limitations of the traditional data warehouse. Greenplum was originally founded in 2003 by Scott Yara and Luke Lonergan as a merger of two companies, Didera and Metapa. Its purpose was to produce an analytic

data warehouse with three major goals: rapid query response, rapid data loading, and rapid analytics by moving the analytics to the data.

It is important to note that Greenplum is an analytic data warehouse and not a transactional relational database. Although Greenplum does have the notion of a transaction, which is useful for Extract, Transform, and Load (ETL) jobs, you should not use it for transactional purposes like ticket reservation systems, air traffic control, or the like. Successful Greenplum deployments include, but are not limited to the following:

- Fraud analytics
- Financial risk management
- Cyber security
- Customer churn reduction
- Predictive maintenance analytics
- Manufacturing optimization
- Smart cars and Internet of Things (IoT) analytics
- Insurance claims reporting and pricing analytics
- Healthcare claim reporting and treatment evaluations
- Student performance prediction and dropout prevention
- Advertising effectiveness
- Traditional data warehouses and business intelligence (BI)

From the beginning, Greenplum was based on PostgreSQL, the popular and widely used open source database. Greenplum kept in sync with PostgreSQL releases until it forked from the main PostgreSQL line at version 8.2.15.

The first version of this new company arrived in 2005, called BizGres. In the same year, Greenplum and Sun Microsystems formed a partnership to build a 48-disk, 4-CPU appliance-like product, following the success of the Netezza appliance. What distinguishes the two is that Netezza required special hardware, whereas all Greenplum products have always run on commodity servers, never requiring special hardware boost.

2007 saw the first publicly known Greenplum product, version 3.0. Later releases added many new features, most notably mirroring and

High Availability—at a time when the underlying PostgreSQL could not provide any of those.

In 2010, a consolidation began in the MPP database world. Many smaller companies were purchased by larger ones. EMC purchased Greenplum in July 2010, just after the release of version 4.0 of Greenplum. EMC packaged Greenplum into a hardware platform, the Data Computing Appliance (DCA). Although Greenplum began as a pure software play, with customers providing their own hardware platform, the DCA became the most popular platform.

2011 saw the release of the first paper describing Greenplum's approach to in-database machine learning and analytics, MADlib. There is a later chapter in this book describing MADlib in more detail. In 2012, EMC purchased Pivotal Labs, a well-established San Francisco–based company that specialized in application development incorporating pair programming, Agile methods, and involving the customer in the development process. This proved to be important not only for the future development process of Greenplum, but also for giving a name to the 2013 spinoff of Greenplum from EMC. The spinoff was called Pivotal and included assets from EMC as well as from VMware. These included the Java-centric Spring Framework, RabbitMQ, the Platform as a Service (PaaS) Cloud Foundry, and the in-memory data grid Apache Geode, known commercially as GemFire.

In 2015, Pivotal announced that it would adopt an open source strategy for its product set. Pivotal would donate most of the software to the Apache Foundation and the software then would be freely licensed under the Apache rules. However, it maintained a subscription-based enterprise version of the software, which it continues to sell and support.

The Pivotal data products then included the following:

- Greenplum
- HDB/Apache HAWQ, a data warehouse based on Greenplum that runs natively on Hadoop
- Gemfire/Apache Geode
- Apache MADlib (incubating)

Officially, the open source version is known as the Greenplum Database and the commercial version is the Pivotal Greenplum Database. With the exception of some features that are proprietary and available only with the commercial edition, the products are the same.

Greenplum management thought about an open source strategy before 2015 but decided that the industry was not ready. By 2015, many customers were beginning to require open source. Greenplum's adoption of an open source strategy saw Greenplum community contributions to the software as well as involvement of PostgreSQL contributors. Pivotal sees the move to open source as having several advantages:

- Avoidance of vendor lock-in
- Ability to attract talent in Greenplum development
- Faster feature addition to Greenplum with community involvement
- Greater ability to eventually merge Greenplum to current PostgreSQL version
- Many customers demand open source

There are several distinctions between the commercial Pivotal Greenplum and the open source Greenplum. Pivotal Greenplum offers the following:

- 24/7 premium support
- Database installers and production-ready releases
- GP Command Center—GUI management console
- GP Workload Manager—dynamic rule based resource management
- GPText—Apache Solr-based text analytics
- Greenplum GemFire Connector—data transfer between Pivotal Greenplum and Pivotal GemFire low latency in memory data grid
- Quicklz compression
- Open Database Connectivity (ODBC) and Object Linking and Embedding, Database (OLEDB) drivers for Pivotal Greenplum

NOTE

GP Command Center, GP Workload Manager, and GPText are discussed in other sections of this book.

2015 also saw the arrival of the Greenplum development organization's use of an Agile development methodology; in 2016, there were 10 releases of Pivotal Greenplum, which included such features as the release of the GPORCA optimizer, a high-powered, highly parallel cost-based optimizer for big data. In addition, Greenplum added features like a more sophisticated Workload Manager to deal with issues of concurrency and runaway queries, and the adoption of a resilient connection pooling mechanism. The Agile release strategy allows Greenplum to quickly incorporate both customer requests as well as ecosystem features.

With the wider adoption of cloud-based systems in data warehousing, Greenplum added support for Amazon Simple Storage Service (Amazon S3) files for data as well as support for running Pivotal Greenplum in both Amazon Web Services (AWS) as well as Microsoft's Azure. 2016 saw an improved Command Center monitoring and management tool and the release of the second-generation of native text analytics in Pivotal Greenplum. But, perhaps most significant is Pivotal's commitment to reintegrate Greenplum into more modern versions of PostgreSQL, eventually leading to PostgreSQL 9.x support. This is beneficial in many ways. Greenplum will acquire many of the features and performance improvements made in PostgreSQL in the past decade. In return, Pivotal then can contribute back to the community.

Pivotal announced that it expected to release Greenplum 5.0 in the first half of 2017.

In Greenplum 5.0, the development team cleaned up many diversions from main line PostgreSQL, focusing on where the MPP nature of Greenplum matters and where it doesn't. In doing this, the code base is now considerably smaller and thus easier to manage and support.

It will include features such as the following:

- JSON support, which is of interest to those linking Greenplum and MongoDB and translating JSON into a relational format
- XML enhancements, such as an increased set of functions for importing XML data into Greenplum
- PostgreSQL-based Analyze that will be an order of magnitude faster generating table statistics
- Enhanced vacuum performance
- Lazy transactions IDs, which translate into fewer vacuum operations
- Universally unique identifier (UUID) data type
- Raster PostGIS
- User-defined functions (UDF) default parameters

What Is Massively Parallel Processing

To best understand how massively parallel processing (MPP) came to the analytic database world, it's useful to begin with scientific computing.

Stymied by the amount of time required to do complex mathematical calculations, the Cray-1 computer introduced vectorized operations in the early 1970s. In this architecture, the CPU acts on all the elements of the vector simultaneously or in parallel, speeding the computation dramatically. As Cray computers become more expensive and budgets for science were static or shrinking, the scientific community expanded the notion of parallelism by dividing complex problems into small portions and dividing the work on a number of independent, small, inexpensive computers. This group of computers became known as a cluster. Tools to decompose complex problems were originally scarce and much expertise was required to be successful. The original attempts to extend the MPP architecture to data analytics was difficult. However, a number of small companies discovered that it was possible to start with standard SQL relational databases, distribute the data among the servers in the cluster, and transparently parallelize operations. Users could write SQL code without knowing the data distribution. Greenplum was one of the pioneers in this endeavor.

Here's a small example of how MPP works. Suppose that there is a box of 1,200 business cards. The task is to scan all the cards and find the names of all those who work for Acme Widget. If a person can scan one card per second, it would take that one person 20 minutes to find all those people whose card says Acme Widget.

Let's try it again, but this time distribute the cards into 10 equal piles of 120 cards each and recruit 10 people to scan the cards, each one scanning the cards in one pile. If they simultaneously scanned at the rate of 1 card per second, they would all finish in about 2 minutes. This is an increase in speed of 10 times.

This idea of data and workload distribution is at the heart of MPP database technology. In an MPP database, the data is distributed in chunks to all the nodes in the cluster. In the Greenplum database, these chunks of data and the processes that operate on them are known as *segments*. In an MPP database, as in the business card example, the amount of work distributed to each segment should be approximately the same to achieve optimal performance.

Of course, Greenplum is not the only MPP technology or even the only MPP database. Hadoop is a common MPP data storage and analytics tool. Spark also has an MPP architecture. Pivotal GemFire is an in-memory data-grid MPP architecture. These are all very different from Greenplum because they do not natively speak standard SQL.

The Greenplum Database Architecture

The Greenplum Database employs a *shared-nothing* architecture. This means that each server or node in the cluster has its own independent operating system (OS), memory, and storage infrastructure. Its name notwithstanding, in fact there is something shared and that is the network connection between the nodes that allows them to communicate and transfer data as necessary. Figure 1-1 presents an overview of the Greenplum Database architecture.

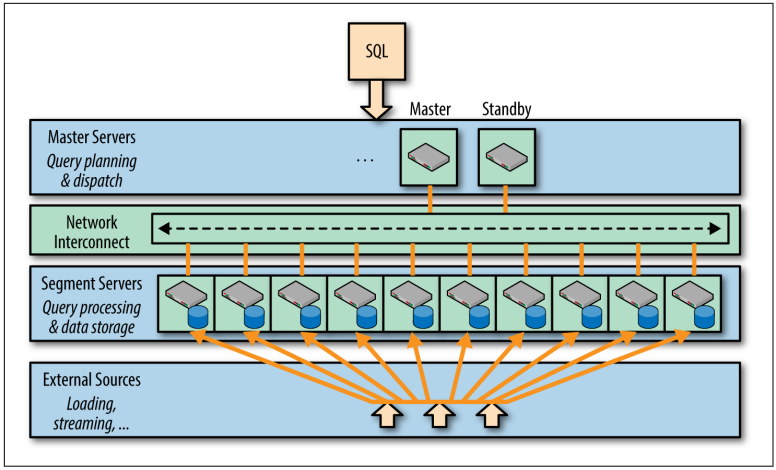

Figure 1-1. The Greenplum MPP architecture

Master and Standby Master

Greenplum uses a master/worker MPP architecture. In this system, users and database administrators (DBAs) connect to a master server, which houses the metadata for the entire system. This metadata is stored in a PostgreSQL database derivative. When the Greenplum instance on the master server receives a SQL statement, it parses it, examines the metadata repository, forms a plan to execute that statement, passes the plan to the workers, and awaits the result. In some circumstances, the master must perform some of the computation.

Only metadata is stored on the master. All the user data is stored on the segment servers, the worker nodes in the cluster. In addition to the master, all production systems should also have a standby server. The standby is a passive member of the cluster, whose job is to receive mirrored copies of changes made to the master's metadata. In case of a master failure, the standby has a copy of the metadata, preventing the master from becoming a single point of failure.

Some Greenplum clusters use the standby as an ETL server because it has unused memory and CPU capacity. This might be satisfactory when the master is working, but in times of a failover to the standby, the standby now is doing the ETL work as well as its role as the master. This can become a choke point in the architecture.

Segments and Segment Hosts

Greenplum distributes user data into what are often known as *shards*, but are called *segments* in Greenplum. A segment host is the server on which the segments resides. Typically, there are several segments running on each segment server. In a Greenplum installation with eight segment servers, each might have six segments for a total of 48 segments. Every user table in Greenplum will have its data distributed in all of the 48 segments. We go into more detail on distributing data later in this book. Unless directed by support, users or DBAs should never connect to the segments themselves except through the master.

A single Greenplum segment server runs multiple segments. Thus, all other things being equal, it will run faster than a single-instance database running on the same server. That said, you should never use a single-instance Greenplum installation for a business-critical process because it provides no high availability or failover in case of hardware, software, or storage error.

Private Interconnect

The master must communicate with the segments and the segments must communicate with one another. They do this on a private Universal Datagram Protocol (UDP) network that is distinct from the public network on which users connect to the master. This is critical. Were the segments to communicate on the public network, user downloads and other heavy loads would greatly affect Greenplum performance. The private network is critical. Greenplum requires a 10 Gb network and strongly urges redundant 10 Gb switches for redundancy.

Other than the master, the standby, and the segment servers, some other servers may be plumbed into the private interconnect network. Greenplum will use these to do fast parallel data loading and unloading. This topic is discussed in the data loading chapter.

Mirror Segments

In addition to the redundancy provided by the standby master, Greenplum strongly urges the creation of *mirror segments*. These are segments that maintain a copy of the data on a primary segment, the one that actually does the work. Should either a primary segment or

the host housing a primary segment fail, the mirrored segment contains all of the data on the primary segment. Of course, the primary and its mirror must reside on different segment hosts. When a segment fails, the system automatically fails-over from the primary to the mirrored segment, but operations in flight fail and must be restarted. DBAs can run a process to recover the failed segment to synchronize it with the current state of the databases.

Learning More

There is a wealth of information about Greenplum available on the Pivotal website. In addition, you can find documentation on the latest version of the Pivotal Greenplum Database General Guide to Pivotal Greenplum Documentation (latest version).

For a discussion on building Greenplum clusters the Greenplum Cluster Concepts Guide is invaluable for understanding cluster issues that pertain to appliances, cloud-based Greenplum or customer-built commodity clusters.

If you're implementing Greenplum, you should read the Best Practices Guide. As this book should make clear, there are a number of things to consider to make good use of the power of the Greenplum Database. If you're accustomed to single-node databases, you are likely to miss some of the important issues that this document helps explain.

The Greenplum YouTube channel has informative content on a variety of technical topics. Most of these are general and do not require any experience with Greenplum to be informative. Others, such as the discussion on the Pivotal Query Optimizer, go into considerable depth.

There is also a Greenplum Users Meetup Group that meets in person in either San Francisco or Palo Alto. It is usually available as a live webcast. If you sign up as a member, you'll be informed about the upcoming events.

A Greenplum database developer's tasks are generally not as complicated as those for a mission-critical OLTP database. Nonetheless, many find it useful to attend the Pivotal Academy Greenplum Developer class.

As part of the Greenplum open source initiative, Pivotal formed two groups tasked with facilitating discussion about Greenplum. The first group deals with user questions about Greenplum. Pivotal data personnel monitor the group and provide answers in a timely fashion. The second group is a conversation vehicle for the Greenplum development community. In the spirit of transparency, Pivotal invites interested parties to listen in and learn about the development process and potentially contribute.

Members of the Greenplum engineering, field services, and data science groups blog about Greenplum, providing insights and use cases not found in many of the other locations. These tend to be more technical in nature. The Pivotal Greenplum Knowledge Base houses general questions, their answers, and some detailed discussions on deeper topics.

Pivotal's Jon Roberts has a PivotalGuru website that discusses issues and techniques that many Greenplum users find valuable. Although Jon is a long time Pivotal employee, the content on PivotalGuru is his own and is not supported by Pivotal.

Internet searches on Greenplum return many hits. It's wise to remember that not everything on the internet is accurate. In particular, as Greenplum evolves over time, comments made in years past might no longer reflect the current state.

In addition to the general material, there are some interesting discussions about the history of Greenplum that warrant examination:

- Greenplum founders Luke Lonergan and Scott Yara discuss Greenplum's origin
- Jacque Istok on Greenplum in the Wild
- How We Made Greenplum Open Source

CHAPTER 2

Deploying Greenplum

As the world continues to evolve, Greenplum is embracing change and now includes four deployment options. With the trend toward cloud-based computing, Greenplum has added public and private cloud to its list of deployment options. This gives Greenplum users four options for production deployment.

- Customer-built clusters
- Appliance
- Public cloud
- Private cloud

Custom(er)-Built Clusters

Initially, Greenplum was a software-only company. It provided a cluster-aware installer but assumed that the customer had correctly built the cluster. This strategy provided a certain amount of flexibility. For example, customers could configure exactly the number of segment hosts they required and could add hosts when needed. They had no restrictions on which network gear to use, how much memory per node, nor the number or size of the disks. On the other hand, building a cluster is considerably more difficult than configuring a single server. To this end, Greenplum has a number of facilities that assist customers in building clusters.

Today, there is much greater experience and understanding in building MPP clusters, but a decade ago, this was much less true and some early deployments were sub-optimal due to poor cluster design and its effect on performance. The *gpcheckperf* utility checks disk I/O bandwidth, network performance, and memory bandwidth. Assuring a healthy cluster before Greenplum deployment goes a long way to having a performant cluster.

Customer-built clusters should be constructed with the following principles in mind:

- Greenplum wants consistent high performance read/write throughput. This almost always means servers with internal disks. Some customers have built clusters using large shared storage-area network (SAN) devices. Even though input/output operations per second (IOPS) numbers can be impressive for these devices, that's not what is important for Greenplum, which does large sequential reads and writes. A single SAN device attached to a large number of segment servers often falls short in terms of how much concurrent sequential I/O it can support.
- Greenplum wants consistently high network performance. A 10 GB network is required.
- Greenplum, and virtually every other database, likes to have plenty of memory. Greenplum recommends at least 256 GB of RAM per segment server. All other things being equal, the more memory, the greater concurrency is possible. That is, more analytic queries can be run simultaneously.

Some customers doing extensive mathematical analytics find that they are CPU limited. Given the Greenplum segment model, more cores would benefit these customers. More cores will mean more segments and thus more processing power. When performance is less than expected, it's important to know the gating factor. In general, it is memory or I/O rather than CPU.

The Pivotal Clustering Concepts Guide should be required reading if you are deploying Greenplum on your own hardware.

Appliance

After Greenplum was purchased by EMC, the company provided an appliance called the Data Computing Appliance (DCA), which soon

became the predominant deployment option. Many early customers did not have the kind of expertise to build and manage a cluster. They did not want to deal with assuring that the OS version and patches on their servers were in accordance with the database version. They did not want to upgrade and patch the OS and Greenplum. For them, the DCA was a very good solution. The current DCA v3 is an EMC/Dell hardware and software combination that includes the servers, disks, network, control software, and also the Greenplum Database. It also offers support and service from Dell EMC for the hardware and OS and Pivotal for the Greenplum software.

There are advantages of the DCA over a customer-built cluster.

The DCA comes with all the relevant software installed and tested. All that is required at the customer site is a physical installation and site-specific configuration such as external IP address setup. In general, enterprises find this faster than buying hardware, building a cluster, and installing Greenplum. It also ensures that all known security vulnerabilities have been identified and fixed. If enabled, the DCA will "call home" when its monitoring capabilities detect unusual conditions.

The DCA has some limitations. The number of segment hosts must be a multiple of four. No other software can be placed on the nodes without vendor approval. It comes with fixed memory and disk configurations.

Public Cloud

Many organizations have decided to move much of their IT environment away from their own datacenters to the cloud. Public cloud deployment offers the quickest time to deployment. You can configure a functioning Greenplum cluster in less than an hour.

Pivotal now has Greenplum available, tested, and certified on AWS and Microsoft Azure and plans to have an implementation on Google Cloud Platform (GCP) by the first half of 2017. In Amazon, the Cloud Formation Scripts define the nodes in the cluster and the software that resides there. Pivotal takes a very opinionated view of the configurations it will support in the public cloud marketplaces; for example:

- Only certain numbers of segment hosts are supported using the standard scripts for cluster builds.
- Only certain kinds of nodes will be offered for the master, standby, and segment servers.
- Only high-performance 10 Gb interconnects are offered. For performance and support reasons, there are only some Amazon Machine Images (AMIs) that are available.

These are not arbitrary choices. Greenplum has tested these configurations and found them suitable for efficient operation in the cloud. That being said, customers have built their own clusters without using the Greenplum-specific deployment options on both Azure and AWS. Although they are useful for QA and development needs, these configurations might not be optimal for use in a production environment.

Private Cloud

Some enterprises have legal or policy restrictions that prevent data from moving into a public cloud environment. These organizations can deploy Greenplum on private clouds, mostly running VMware infrastructure. Details are available in the Greenplum documentation, but some important principles apply to all virtual deployments:

- There must be adequate network bandwidth between the virtual hosts.
- Shared-disk contention is a performance hindrance.
- Automigration must be turned off.
- As in real hardware systems, primary and mirror segments must be separated, not only on different virtual servers, but also on physical infrastructure.
- Adequate memory to prevent swapping is a necessity.

If these shared-resource conditions are met, Greenplum will perform well in a virtual environment.

Choosing a Greenplum Deployment

The choice of a deployment depends upon a number of factors that must be balanced:

Security
: Is a public cloud possible or does it require security vetting?

Cost
: The cost of a public cloud deployment can be large if Greenplum needs to run 24 hours a day, 365 days a year. This is an operational expense that must be balanced against the capital expense of purchasing a DCA or hardware for a custom-built cluster. The operational chargeback cost of running either a private cloud, DCA, or customer-build cluster varies widely among datacenters. Moving data to and from a site in the enterprise to the cloud can be costly.

Time-to-usability
: A public cloud cluster can be built in an hour or so. A private cloud configuration in a day. A customer-built cluster takes much longer. A DCA requires some lead time in ordering and installing.

Greenplum Sandbox

Although not for production use, Greenplum distributes a sandbox in both VirtualBox and VMware format. This VM contains a small configuration of Greenplum Database with some sample data and scripts that illustrate the principles of Greenplum. It is freely available at PivNet. The sandbox also is in AWS. This blog post provides more detail.

The sandbox has no high availability features. There is no standby master nor does it have mirror segments. It is built to demonstrate the features of Greenplum, not for database performance. It's of invaluable help in learning about Greenplum.

Learning More

You can find a more detailed introduction to the DCA, the Greenplum appliance, at the EMC-Dell Greenplum DCA. The Getting Started Guide contains much more details about architecture, site planning, and administration. A search for "DCA" at the EMC-Dell website will yield a list of additional documentation. To subset the information to the most recent version of the DCA, click the radio button "Within the Last Year" under "Last Updated."

Building a cluster for the first time is likely to be a challenge. To minimize the time to deployment, Greenplum has published two very helpful documents on clustering: the Clustering Concepts Guide, mentioned in the introductory chapter of this book and a website devoted to clustering material. Both should be mandatory reading before you undertake building a custom configuration. Pivotal provides advice for building Greenplum in virtual environments.

Pivotal's Andreas Scherbaum has produced Ansible scripts for deploying Greenplum. These is not an officially supported endeavor and requires basic knowledge of Ansible.

Those interested in running Greenplum in the public cloud should consult the offerings on AWS and Microsoft Azure. There is a brief video on generating an Azure deployment that walks through the steps to create a Greenplum cluster.

CHAPTER 3
Organizing Data in Greenplum

To make effective use of Greenplum, architects, designers, developers, and users must be aware of the various methods by which data can be stored because it will affect performance in loading, querying, and analyzing datasets. A simple "lift and shift" from a transactional data model is almost always suboptimal. Data warehouses generally prefer a data model that is flatter than a normalized transactional model. Data model aside, Greenplum offers a wide variety of choice in how the data is organized. These choices include the following:

Distribution
: Determines into which segment table rows are assigned

Partitioning
: Determines how the data is stored on each of the segments

Orientation
: Determines whether the data is stored by rows or by columns

Compression
: Used to minimize data table storage in the disk system

Append-optimized tables
: Used to enhance performance for data that is rarely changed

External tables
: Provide a method for accessing data outside Greenplum

Indexing
: Used to speed lookups of individual rows in a table

Distributing Data

One of the most important methods for achieving good query performance from Greenplum is the proper distribution of data. All other things being equal, having roughly the same number of rows in each segment of a database is a huge benefit. In Greenplum, the data distribution policy is determined at table creation time. Greenplum adds a distribution clause to the Data Definition Language (DDL) for a CREATE TABLE statement. There are two distribution methods. One is random, in which each row is randomly assigned a segment when the row is initially inserted. The other is by the use of a hash function computed on the values of some columns in the table.

Here are some examples:

```
CREATE TABLE bar
(id INT, stuff TEXT dt DATE) DISTRIBUTED BY (id);

CREATE TABLE foo
(id INT, more_stuff TEXT) DISTRIBUTED RANDOMLY;

CREATE TABLE gruz
(id INT, still_more_stuff TEXT, gender CHAR(1));
```

In the case of the bar table, Greenplum will compute a hash value on the `id` column of the table when the row is created and will use that value to determine into which segment the row should reside.

The `foo` table will have rows distributed randomly among the segments. Although this will generate a good distribution with almost no skew, it might not be useful when colocated joins will help performance. Random distribution is fine for small-dimension tables and lookup tables but probably not for large fact tables.

In the case of the `gruz` table, there is no distribution clause. Greenplum will use a set of default rules. If a column is declared to be a primary key, Greenplum will hash on this column for distribution. Otherwise, it will choose the first column in the text of the table definition if it is an eligible data type. If the first column is a user-defined type or a geometric type, the distribution policy will be random. It's a good idea to explicitly distribute the data and not rely

on the set of default rules; they might change in a future major release.

Currently an update on the distributed column(s) is allowed only with the use of the Pivotal Query Optimizer (discussed in a later section). With the legacy query planner, updates on the distribution column were not permitted, because this would require moving the data to a different segment database.

A poorly chosen distribution key can result in suboptimal performance. Consider a database with 64 segments and the table `gruz` had it been distributed by the gender column. Let's suppose that there are three values, "M," "F," and NULL for unknown. It's likely that roughly 50 percent of the values will be "M"; roughly 50 percent "F"; and a small proportion of NULLs. This would mean that two of the 64 segments would be doing useful work and others would have nothing to do. Instead of achieving an increase in speed of 64 times over a nonparallel database, Greenplum could generate an increase of only two times, or, in worst case, no increase in speed at all if the two segments with the "F" and "M" data live on the same segment host.

The distribution key can be a set of columns, but experience has shown that one or two columns usually is sufficient. Rarely are more than two required. Distribution columns should always have high cardinality, although that in itself will not guarantee good distribution. That's why it is important to check the distribution. To determine how the data actually is distributed, Greenplum provides a simple SELECT statement that shows the data distribution:

```
SELECT gp_segment_id, count(*) FROM foo GROUP BY 1;
 gp_segment_id | count
---------------+--------
             1 | 781632
             0 | 749898
             2 | 761375
             3 | 753063
```

`gp_segment_id` is a pseudocolumn, one that is not explicitly defined by the create table statement but maintained by Greenplum.

A best practice is to run this query after the initial data load to see that the chosen distribution policy in generating equi-distribution. You also should run it occasionally in the life of the data warehouse to see if changing conditions suggest a different distribution policy.

Another guiding principle in data distribution is colocating joins. If there are queries that occur frequently and consume large amounts of resources in the cluster, it is useful to distribute the tables on the same columns.

For example, if queries frequently join tables on the `id` column, there might be performance gains if both tables are distributed on the `id` column because the rows to be joined will be in the same segment. This is less important if one of the tables is very small, perhaps a small dimension table in a data warehouse. In that case, Greenplum will automatically broadcast or redistribute this table to all the segments at a relatively small cost completely transparently to the user.

Here's an example to illustrate colocation:

```
SELECT f.dx, f.item, f.product_name f.store_id s.store_address
FROM fact_table f, store_table s
WHERE f.store_id = s.id AND f.dx = 23
ORDER BY f.product_name;
```

To colocate the join, use the following distribution clause for the `fact_table`:

```
DISTRIBUTED BY (store_id)
```

For the `store_table` use this:

```
DISTRIBUTED BY (id)
```

Of course, some other queries will use different join conditions, so it's not possible to have every join colocated. Some experience with the use case will go a long way to find the best distribution policy. It's important that the join columns have the same data type. The integer 1 and the character "1" are not the same and will hash differently.

If there is no prior knowledge of how the tables are to be used in select statements, random distribution can be effective. If further usage determines that another distribution pattern is preferable, it's possible to redistribute the table. This is often done by a "CTAS," or "create table as select." To redistribute a table, use the following commands:

```
CREATE TABLE new_foo AS SELECT * from foo
   DISTRIBUTED BY (some_other_coummn);
DROP TABLE foo;
ALTER TABLE new_foo RENAME TO foo;
```

Or, more simply, use the following:

```
ALTER TABLE foo SET DISTRIBUTED BY (some_other_column);
```

Both of these methods will require resource usage and should be done at relatively quiet times. In addition, Greenplum requires ample disk space for both versions of the table during the reorganization.

Polymorphic Storage

Greenplum employs the concept of *polymorphic storage*. That is, there are a variety of methods by which Greenplum can store data in a persistent manner. This can involve partitioning data, organizing the storage by row or column orientation, various compression options, and even storing it externally to the Greenplum Database. There is no single best way that suits all or even the majority of use cases. The storage method is completely transparent to user queries, and thus users do not do coding specific to the storage method. Utilizing the storage options appropriately can enhance performance.

Partitioning Data

Partitioning a table is a common technique in many databases, particularly in Greenplum. It involves separating table rows for efficiency in both querying and archiving data. You partition a table when you create it by using a clause in the DDL for table creation. Here is an example of a table partitioned by range of the timestamp column into partitions for each week:

```
CREATE TABLE foo (fid INT, ftype TEXT, fdate DATE)
DISTRIBUTED by (fid)
PARTITION BY RANGE(fdate)
(
    PARTITION week START ('2015-11-01'::DATE)
                 END ('2016-01-31'::DATE)
                 EVERY ('1 week'::INTERVAL)
);
```

This example creates 13 partitions of the table, one for each week from 2015-11-01 to 2016-01-31.

What is happening here? As rows enter the system, they are distributed into segments based upon the hash values for the `fid` column. Partitioning works at the storage level in each and every

segment. All values whose `fdate` values are in the same week will be stored in a separate file in the segment host's filesystem. Why is this a good thing? Because Greenplum usually does full-table scans rather than index scans, if a WHERE clause in the query can limit the date range, the optimizer can eliminate reading the files for all dates outside that date range.

If our query was

```
SELECT * FROM foo WHERE fdate > '2016-01-14'::DATE;
```

the optimizer would know that it had to read data from only the three latest partitions, which eliminates about 77 percent of the scans and would likely reduce query time by a factor of four. This is known as *partition elimination*.

In addition to range partitioning, there is also list partitioning, in which the partitions are defined by discrete values, as demonstrated here:

```
CREATE TABLE bar (bid integer, bloodtype text, bdate date)
DISTRIBUTED by (bid)
PARTITION BY LIST(bloodtype)
(PARTITION a values('A+', 'A', 'A-'),
 PARTITION b values ('B+', 'B-', 'B'),
 PARTITION ab values ('AB+', 'AB-', 'AB'),
 PARTITION o values ('O+', 'O-', 'O')
 DEFAULT PARTITION unknown);
```

This example also exhibits the use of a default partition, into which rows will go if they do not match any of the values of the `a`, `b`, `ab`, or `o` partitions. Without a default partition, an attempt to insert a row that does not map to a partition will result in an error. The default partition is handy for catching data errors, but has a performance hit. The default partition will always be scanned even if explicit values are given in the WHERE clause that map to specific partitions. Also, if the default partition is used to catch errors and it is not pruned periodically, it can grow to be quite large.

In this example, the partitions are named. In the range partitioning example, they are not. Greenplum will assign names to the partitions if they are unnamed in the DDL.

Greenplum allows subpartitioning, the creation of partitions within partitions of a different column than the major partition column as the subpartitions. This might sound appealing, but it can lead to a huge number of small partitions, many of which have little or no

data, but need to be opened and closed by the filesystem during table scans. In this example, there are 52 × 5 = 260 partitions. If this were column-oriented, we would get 52 × 5 × (*number of columns*) partitions. In Greenplum, as in Postgres, each partition is actually a table.

```
CREATE TABLE gruz (bid INT, bloodtype TEXT, bdate DATE)
DISTRIBUTED by (bid)
PARTITION BY RANGE (bdate)
SUBPARTITION BY LIST(bloodtype)
SUBPARTITION TEMPLATE
(SUBPARTITION a VALUES('A+', 'A', 'A-'),
 SUBPARTITION b VALUES ('B+', 'B-', 'B'),
 SUBPARTITION ab VALUES ('AB+', 'AB-', 'AB'),
 SUBPARTITION o VALUES ('O+', 'O-', 'O'),
 DEFAULT SUBPARTITION unknown)
(START (DATE '2016-01-01')
 END (DATE '2016-12-25')
   EVERY (INTERVAL '1 week'));
```

Often, new users to Greenplum confuse partitioning data with distributing data. All data must be distributed. This is how Greenplum works. But partitioning is optional. Partitioning data is orthogonal to distribution. There is no notion of randomly partitioning data as it would make no sense. The optimizer would not be able to do partition elimination because it could not use the WHERE clause to determine where data resided on the filesystem. Distribution works to separate data across the segments, and partitioning works to separate data within segments. You can think of this two-dimensional "striping" of data as "plaiding" because we can envision it as stripes running in perpendicular directions. There is a further confusion. PARTITION is a keyword in the definition of Window Functions. This use of partition has nothing to do with table partitioning.

Just as Greenplum did not allow update of the distribution columns, it also forbade the update of the partition column until the use of the GPORCA optimizer.

In many analytic data warehouses, more recent data is more important than older data, and it is useful to archive older data. This older data may might be required for occasional use, but might be best archived out of the database. The external table mechanism is ideal for this use.

It is possible to add, drop, and split partitions. The Greenplum Database documentation covers this in detail.

Here are some best practices on partitioning:

- It makes little sense to partition small lookup tables and the like.
- Never partition on columns that are part of the distribution clause.
- In column-oriented tables, the number of files for heavily partitioned tables can be very large as each column will have its own file. A table with 50 columns, partitioned hourly with 10 subpartitions over two years will have $50 \times 24 \times 365 \times 2 \times 10 = 8{,}760{,}000$ files per segment. With a segment host having eight segments this would be more than 70 million files that the server must open and close to do a full-table scan. This will have a huge performance impact.

To obtain more detail on the partitions, the following query works nicely on range partitioned tables. A WHERE clause for particular schemas and tables will shorten the output.

```
SELECT
       schemaname || '.' || tablename AS "Schema.Table"
      ,partitiontablename AS "PartTblName"
      ,partitiontype AS "Type"
      ,split_part(partitionrangestart, '::', 1) AS "Start"
      ,split_part(partitionrangeend, '::', 1) AS "End"
      ,partitionendinclusive AS "End Inclusive?"
FROM
     pg_partitions;
```

Compression

Compression is normally considered a method to decrease the storage requirements for data, but in Greenplum, it has an additional benefit. When doing analytics with large data sets, queries often read the entire table from disk. If the table is compressed, the smaller size can result in many fewer disk reads, enhancing query performance. Small tables need not be compressed because it will yield neither significant performance gain nor storage gains. Compression and expansion require CPU resources. Systems already constrained by CPU need to experiment to determine if compression will be helpful in query performance.

Like distribution, you specify compression when you create a table. Greenplum offers many compression options but only for append-

optimized or column-oriented tables. The first statement in the following example creates a table, the second generates an error:

```
CREATE TABLE foobar (fid INT, ft TEXT)
WITH (compresstype = quicklz,orientation=column,appendonly=true)
DISTRIBUTED by (fid);

CREATE TABLE gruz (gid INT, gt TEXT)
WITH (compresstype = quicklz,orientation=row)
DISTRIBUTED by (gid);
ERROR: invalid option "compresstype" for base relation.
Only valid for Append Only relations
```

You can use Zlib and quicklz to compress row-oriented tables. You can use run-length encoding (rle) to compress column-oriented tables. Run-length encoding can be very effective when used on columns with sorted data.

A column-oriented table also can have different compression strategies for different columns. In the following example, each of the four columns is compressed differently, although this should not be interpreted as a best practice:

```
create table comp1
   ( cid int ENCODING(compresstype=quicklz),
     ct text ENCODING(compresstype=zlib, compresslevel =  5),
     cd date ENCODING(compresstype=rle_type),
     cx smallint)
with (appendonly=true, orientation=column)
distributed by (cid);
```

You can run Zlib compression at many levels. It's tempting to try the higher levels, but they will take considerably longer to compress and might not provide any significant advantage in compression ratio or in improving query speed.

Append-Optimized Tables

An append-optimized table also stores table data by row, but in a smaller format. This smaller format excludes information that makes updates faster, so you should use it only for data that rarely changes. Prior to Greenplum 4.3, Greenplum used append-only tables, which allowed rows to be appended but not updated, as shown here:

```
CREATE TABLE foobar (fid INT, ft TEXT)
WITH (APPENDONLY = TRUE) DISTRIBUTED BY (fid);
```

Append-optimized tables are particularly useful in denormalized-fact tables in a data warehouse where new data is loaded into staging or ETL tables, cleansed and corrected if necessary, and then placed in the production tables.

External Tables

The Greenplum concept of an external table is data that is stored outside the database proper but can be accessed via SQL statements, as though it were a normal database table. The external table is useful as a mechanism for loading data as well as a method for archiving data. Because no data in an external table lives within a Greenplum database, an external table is a metadata object that instructs Greenplum how to access the data when required in a query.

External tables can be either readable or writeable and they can refer to files or, in the case of external web tables, can be defined in terms of a URL or a process that generates data.

Creating a readable external table is much like creating a normal table. Data manipulation language (DML) operations, such as update and delete, are not permitted; only those involved in reading the table are allowed. It does not have a `DISTRIBUTED BY` clause, but does have a `LOCATION` clause that informs Greenplum where the data can be found and which protocol is used to access it. Following are the current protocols:

gpfdist
: A parallel transport protocol used to transfer flat files

gpfdists
: A secure version of gpfdist

gphdfs
: Used to access files in Hadoop Distributed File System (HDFS)

s3
: Used to access files in Amazon S3 system

file
: A single-threaded transport suitable for small files

To learn more about the gpfdist protocol, read Chapter 4.

A recent addition to Greenplum is the ability to have readable and writable external tables as files in the Amazon S3 storage tier. This has many useful features. It allows hybrid queries in which some of the data is natively stored in Greenplum with other data living in S3. It also allows Greenplum to use S3 as an archival location for older, less frequently used data. Should the need arise to access the older data, you can access it transparently in place without moving it into Greenplum. You also can use this process to have some table partitions archived to Amazon S3 storage. For tables partitioned by date, this is a very useful feature.

In the following example, raw data to be loaded into Greenplum is located into an existing Greenplum table. The flat file has comma-separated values (CSV) with a header line and is stored on Amazon S3 storage. The first few lines could look like this:

```
123,Fido, /09/09/2010
456, Rover, /01/21/2014
789 ,Bonzo, 04/15/2004
```

The table to be loaded would be defined as follows:

```
CREATE TABLE dogs
(did int, dname text, bday date) distributed randomly;
```

The external table definition could be this:

```
CREATE READABLE EXTERNAL TABLE dogs_ext like(dogs)
LOCATION
('s3://s3-us-west-2.amazonaws.com/s3test.foo.com/ normal/)
FORMAT 'csv' (header)
LOG ERRORS SEGMENT REJECT LIMIT 50 rows;
```

Assuming the Amazon S3 URI is valid and the file is accessible, the following statement will show three rows from the CSV file on Amazon S3 as though it were an internal Greenplum table. Of course, performance will not be as good as if the data were stored internally.

```
SELECT * FROM dogs_ext limit 3;
```

Just as a readable external table is for importing data, a writable external table is for exporting data from Greenplum.

The gphdfs protocol allows for reading and writing data to the HDFS. For Greenplum used in conjunction with Hadoop as a landing area for data, this provides easy access to data ingested into Hadoop and possibly cleaned there with native Hadoop tools.

Greenplum also encompasses the concept of an external web table, both readable and writeable. There are two kinds of web tables: those accessed by a pure HTTP call, and those accessed via an operating system (OS) command. In general, these web tables will access data that is changing on a regular basis and can be used to ingest data from an external source.

The United States Geologic Service produces a set of CSV files that describe global earthquake activity. You can access this data on a regular basis, as shown in the following example:

```
CREATE TABLE earthquake (
  xtime text, latitude float,longitude float,
  depth float,mag float,magType text,nst int,
  gap float,dmin float,rms float,
  net text,id text,updated text,
  place text,type text,
  horError float,depError float,magError float,
  magNst int,status text,locSource text,magSource text)
DISTRIBUTED BY (id);

CREATE EXTERNAL WEB TABLE earthquake_ext (like earthquake)
// Only a superuser can create an external web table
EXECUTE
'wget -qO - http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_day.csv'
ON MASTER
FORMAT 'CSV' (HEADER) ;

INSERT INTO earthquakes SELECT * from earthequakes_ext;
```

The following example illustrates using an OS command that would run on each segment's output, assuming the script reads from a pipe into stdin (more details are available in the Pivotal documentation):

```
CREATE EXTERNAL WEB TABLE error_check (
   edate date,
   euser text,
   etype text,
   emsg text)
EXECUTE 'ssh /usr/local/scripts/error_script.sh'
FORMAT 'CSV' ;
```

To create an external table, users must explicitly be granted this privilege in the following way by gpdamin, the Greenplum superuser. The following command grants gpuser the ability to create external tables using the HTTP protocol:

```
ALTER ROLE gpuser CREATEEXTTABLE;
```

The goal in the first example is to archive older data to somewhere in the filesystem, but still available for query use. First, define an external table like the table to be archived. Next, dump the data from the partition to be archived to the external table. Then,

exchange partitions, making the external table an external partition to the table and thus eliminating the old partition for the week from table:

```
CREATE WRITABLE EXTERNAL TABLE foo_ext
    (LIKE foo)
    ( LOCATION ('gpfdist://somehost:8082/2016_week1')
    FORMAT 'text' (DELIMITER ',');

INSERT into foo_ext
   SELECT * from foo_1_prt_week_1;

CREATE READABLE EXTERNAL TABLE foo_ext_2016_week1
    (LIKE foo)
    ( LOCATION ('gpfdist://somehost:8082/2016_week1')
    FORMAT 'text' (DELIMITER ',');

ALTER TABLE foo
   EXCHANGE PARTITION for ('2016-01-01'::date)
   WITH TABLE foo_ext_2016_week1
   WITHOUT VALIDATION;
```

Greenplum allows the use of Amazon S3 storage as a place to hold these archived partitions. What would change in the preceding example is only the external table definition:

```
CREATE EXTERNAL TABLE foo_ext
    (LIKE foo)
    LOCATION ('s3://s3.amazonaws.com/test/my_data
  config=/home/gpuser/s3.conf')
    FORMAT 'text' (DELIMITER ',');
```

There is a great deal more to this than merely changing the location clause. This is described in the Greenplum Database documentation.

Indexing

In transaction systems, indexes are used very effectively to access a single row or a small number of rows. In analytic data warehouses, this sort of access is relatively rare. Most of the time, the queries access all or many of the rows of the table and a full table scan gives better performance in an MPP architecture than an index scan. In general, Greenplum favors full-table scans over index scans in most cases.

For that reason, when moving data warehouses to Greenplum, best practice is to drop all the indexes and add them on an as-needed basis for particular commonly occurring queries. This has two

added benefits. It saves considerable disk space and time. Maintaining indexes during large data loads degrades performance. Best practice is to drop the indexes on a table during a load and re-create them thereafter. Even though this is a parallel operation, it does consume computing resources.

This is not to suggest that you shouldn't use indexes in Greenplum, but you should use them only for specific situations for which there is a measurable benefit.

Learning More

The Pivotal Greenplum Administrator's Guide discusses table creation and storage options in this section. It's well worth reading before beginning any Greenplum project.

Jon Roberts discusses Greenplum storage is a variety of posts at his PivotalGuru website. There is a section on append-optimized tables, notes on tuning that rely heavily on data organization in Greenplum, and a section on External Storage with Amazon S3.

As mentioned earlier, although Greenplum does not require extensive indexing, the documentation includes a detailed discussion of indexing strategies.

This blog post provides a detailed explanation of the use of Amazon S3 external tables with Greenplum.

CHAPTER 4

Loading Data

Before running queries or using analytic tools, Greenplum needs to ingest data. There are multiple ways to move data into Greenplum. In the sections that follow, we'll explore each of them.

INSERT Statements

The simplest way of inserting data is to use INSERT SQL statement that facilitates inserting a few rows of data. However, because the insert is done via the master node of Greenplum Database, it cannot be parallelized.

An insert like the one that follows is fine for populating the values in a small lookup table.

```
INSERT INTO faa.d_cancellation_codes
  VALUES ('A', 'Carrier'),
         ('B', 'Weather'),
         ('C', 'NAS'),
         ('D', 'Security'),
         ('', 'none');
```

There are set-based insert statements that can be parallelized. They are discussed later in this chapter.

\COPY command

The \COPY command is both a psql command as well as a SQL command in Greenplum, with minor differences, one of which is that COPY can be used only by gpadmin, the Greenplum superuser.

You can run \COPY as psql scripts. psql is the command-line tool for accessing Greenplum as well as PostgreSQL. A Greenplum version is available as part of the normal installation process.

\COPY is efficient for loading smallish datasets—a few thousand rows or so—but because it is single-threaded through the master server, it is less useful for loads of millions of rows or more.

Let's consider a table called `dogs`, defined as follows:

```
CREATE TABLE dogs
(did int, dname text, bday date) distributed randomly;
```

Now, let's look at a data file on the master server containing a header line and three rows of data:

```
id,name,bday
123,Fido, /09/09/2010
456, Rover, /01/21/2014
789 ,Bonzo, 04/15/2004
```

Here's the SQL statement that copies the three rows of data to the table `dogs`:

```
\COPY dogs  FROM '/home/gpuser/Exercises/dogs.csv'
CSV HEADER LOG ERRORS SEGMENT REJECT LIMIT 50 ROWS;
```

Raw data is often filled with errors. Without the REJECT clause, the \COPY statement would fail if there were errors. In this example, the REJECT clause allows the script to continue loading until there are 50 errors. The LOG ERRORS clause will place the errors in a log file. This is explained in the Greenplum SQL Reference manual.

You also can use \COPY to unload small amounts of data:

```
\COPY dogs  TO '/home/gpuser/Exercises/dogs_out.csv'  CSV HEADER;
```

The gpfdist Tool

For large datasets, neither INSERT nor \COPY will be nearly as performant as Greenplum's parallel loading techniques. This makes use of external tables and the gpfdist tool. gpfdist is a file server that runs on the server on which the raw data resides. It sits there passively until a Greenplum SQL statement implicitly calls it to request data from an external table.

There is an external file with data that looks like the following:

```
id,name,bday
123,Fido, /09/09/2010
456, Rover, /01/21/2014
789 ,Bonzo, 04/15/2004
```

The table to be loaded would be defined like this:

```
CREATE TABLE dogs
(did int, dname text, bday date) distributed randomly;
```

The external table definition could be as shown here:

```
CREATE READABLE EXTERNAL TABLE dogs_ext like(dogs)
LOCATION ('gpfdist://10.0.0.99:8081/dogs.csv')
FORMAT 'csv' (header)
LOG ERRORS SEGMENT REJECT LIMIT 50 rows;
```

After the gpfdist process has been started on the host housing the comma-separated values (CSV) file, the data can easily be imported by using the following statement:

```
INSERT INTO dogs AS SELECT * FROM dogs_ext;
```

When the INSERT statement is executed, all of the segments engaged in the INSERT statement will issue requests to the gpfdist process running on the server with address 10.0.0.99 for chunks of data. They will parse each row, and if the row should belong to the segment that imports it, it will be stored there. If not, the row will be shipped across the private interconnect to the segment to which it belongs and it will be stored there. This process is known as *Scatter-Gather*. The mechanics of this are completely transparent to the user community.

The Greenplum documentation describes a number of methods of deploying gpfdist. In particular, the number of gpfdist processes per external server can have a large impact on load speeds. Figure 4-1 shows one example in which one gpfdist process is running on the external ETL server.

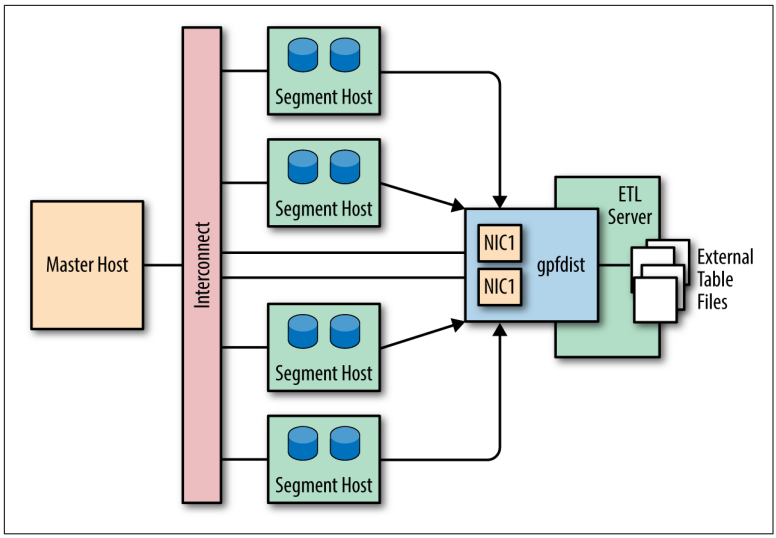

Figure 4-1. External table using single gpfdist instance with multiple network interface cards

The gpload Tool

Many Greenplum users employ the gpload tool. This is a binary command distributed as part of the Greenplum distribution. It uses a YAML configuration file to orchestrate the loading of data. In many organizations, there are dedicated data loading teams who might not actually have much SQL experience or might not even have database accounts. For them, gpload is an ideal tool.

Here's what gpload does behind the scenes:

- Creates a user session in a database
- Creates an external table that describes the data to be loaded
- Starts the gpfdist program on the host where the data is located
- Performs a parallel load of the data from the source to the target table

For this to happen, the configuration file or the command-line argument or relevant environment variable must specify the database user, the target database and table, the location of the target data and the Greenplum master host of the cluster.

Following is a very simple YAML configuration file for our dog table example:

```
VERSION: 1.0.0.1
DATABASE: dogpound
USER: gpuser
HOST: mdw-1
PORT: 5432
GPLOAD:
   INPUT:
    - SOURCE:
         LOCAL_HOSTNAME:
           - data_host.foobar.com
         PORT: 8081
         FILE:
           - /home/gpuser/data/*
       - FORMAT: csv
    - DELIMITER: ','
   OUTPUT:
    - TABLE: public.dogs
    - MODE: INSERT
```

Notice that there are two PORT fields in the YAML file. The first is the Greenplum listener port on which the user session exists. The second is the port that gpfdist and Greenplum uses to transfer data.

There are many useful optional features in gpload:

- Logging load errors for potential correction.
- SQL commands that you can run before and after the load operation. You can use these to add audit commands or check the error files.
- Ability to truncate the target table before loading. This is useful when loading into staging tables on a regular basis.
- Mapping makes it possible for you to apply functions or expressions to data as part of the ingest process. You could use this to encrypt data or transform it in other ways.

You can achieve the same effect of gpload manually.

On `data_host.foobar.com`, run the following command at the Linux prompt:

```
gpfdist -d /home/gpuser/data/ -p 8081 > gpfdist.log 2>&1 &
```

Then, in a psql session, type the following:

```
CREATE TABLE dogs
(did int, dname text, bday date) distributed randomly;

CREATE READABLE EXTERNAL TABLE dogs_ext (like dogs)
LOCATION ('gpfdist://data_host.foobar.com:8081//dogs.csv')
FORMAT 'CSV' (HEADER)
LOG ERRORS SEGMENT REJECT LIMIT 50;

INSERT INTO public.dogs SELECT * from dogs_ext;
```

You can find more information about the YAML configuration file in the next section.

Learning More

There is a thorough discussion of loading and unloading data in the Greenplum documentation.

Pivotal's Jon Roberts has written Outsourcer, a tool that does Change Data Capture from Oracle and SQLServer into Greenplum. This is not a Pivotal product and thus not supported by Pivotal, but Jon makes every effort to maintain, support, and enhance the product.

CHAPTER 5

Gaining Analytic Insight

It's all well and good to efficiently organize and load data, but the purpose of an analytic data warehouse is to gain such insights using a wide variety of tools. Initially, data warehousing used business intelligence tools to investigate previous history of the enterprise. These were embodied in Key Performance Indicators (KPIs) and some limited "What-If" scenarios. Greenplum was more ambitious. It wanted users to be able to do predictive analytics and process optimization using data in the warehouse. To that end, it employed the model of bringing the analytics to the data rather than the data to the analytics. To assist customers, Greenplum formed an internal team of experienced experts in data science and developed analytic tools to work within Greenplum. This chapter explores the ways in which users can gain analytic insight using the Pivotal Greenplum Database.

Data Science on Greenplum with Apache MADlib

Kaushik Das and Frank McQuillan

What Is Data Science and Why Is It Important?

Data science is moving with gusto to the enterprise. The potential for business value in the form of better products and customer experiences as well as mounting competitive pressures is driving this growth. Interest is understandably high in many industries on how

to build and run the appropriate predictive analytics models on the pertinent data to realize this business value.

Much of the interest is due to the proliferation of data generated by the Internet of Things (IoT) as well as advances in memory, CPU, and network capabilities. How can a business use this data to its greatest benefit? The first step is to use the data for retrospective analysis. It is important to see what has happened in the past and be able to slice and dice that data to fully understand what's going on. This is commonly called business intelligence (BI), which makes use of descriptive statistics. The next step is to understand patterns in the data and infer causality from that. This inference can predict what's going on and that can benefit an organization in many ways. This process is called *data science*. The algorithms come from the fields of Artificial Intelligence (AI) and machine learning as well as the more established branches of engineering and statistics. But you need big data technology to make all of this work. Greenplum has been developed with the goal of enabling data science, and in this section, we explore how it does that.

Common Data Science Use Cases

Data science provides value to an organization when it is operationalized to facilitate actions taken by the organization. Data science has two different phases: the *learning phase* and the *application phase*. In the learning phase, data scientists use existing data to form models. In the application phase, we use the models to drive decisions. The operationalization can be of three types based on the latency requirements for each of these phases:

Batch models
: This is when the learning and application both happen in batch mode. An example is a cybersecurity application for quick detection of Advanced Persistent Threats (APTs) through detection of anomalies in lateral movements. The idea is quite simple in concept. We load all the relevant data (like Windows Active Directory data) into a Greenplum Database and use that data to build a graph of all the connections between users and the machines they access within the firewall of a network. The learning consists of understanding the patterns in the graph for each user over a period of time and classifying them. The application consists of detecting anomalies over a period of time by comparing the activity in the recent period with the preexisting

pattern and flagging the anomalies. You can perform this application on a schedule; for example, on a daily basis (you can find more details in "A Data Science Approach to Detecting Insider Security Threats").

For this type of application, you can carry out the entire process in Greenplum. The learning phase involves investigating the data, building features/variables from that data, and building a mathematical model to bring out the relationships (correlations) between the features (for more detail on data science methodology, read "The Eightfold Path of Data Science"). This will require the advanced features of Greenplum, which can handle different types of data such as text. And we can build features and perform statistical tests on large amounts of data very quickly due to the MPP capability of Greenplum.

We also have access to a lot of statistical and machine learning functions through open source packages from Apache MADlib, R, and Python. This makes the entire process of building this sort of data science application much easier than using the traditional methods of moving the data from one place to another and loading it into different tools for data investigation, feature building, and modeling. Data movement is minimized and the computation happens where the data is stored. And this goes for the application (or scoring) of the models on new data, as well.

Event-driven models

This use case involves learning in the batch mode and application in real time. A good example of this is preventive maintenance. The IoT enables us to collect data from machines within a factory or from the field (like oil wells and windmills). It usually costs a lot when these machines fail not just in terms of the repair itself but also in terms of hours of operation lost. In the worst case, there could even be a catastrophic failure leading to loss of life and damage to the environment. We can get a lot of value by using data science to provide early warning of an impending failure. The learning phase involves collecting data from sensors on the machine and its environment and then looking for variations in that data and finding out how they correlate with failures that have happened in the past. This is done very efficiently within Greenplum. The application part—which requires real-time processing of streaming data and applying the model to that data to determine the probability of failure—

requires a microservices-based architecture incorporating in-memory tools like Gemfire (you can find more details in "Operationalizing Data Science Models on the Pivotal Stack"). Greenplum fits very naturally in such a microservices-based architecture.

Streaming models
: This type involves learning and applying in real time. This covers a small subset of data science use cases. Greenplum is not required for this type of model. But, in general, an organization would need all three types of models to truly exploit the potential of all the data available today; Greenplum would be invaluable in the data architecture that could support all these models.

It is useful to summarize the different aspects of Greenplum that make it so attractive for data science.

Tools for Data Science

Let's take a look at some of the tools that make Greenplum such a powerful partner in the data sciente realm:

MPP
: The MPP capability of Greenplum enables the investigation of data and the learning of models very quickly over very large and varied datasets. The text analytics (GPText) and spatio-temporal features (PostGIS) modules of Greenplum are particularly useful in this regard.

Compatibility with older tools
: Most companies have many legacy tools for analyzing data and building models on small data. It is difficult to suddenly switch off all the existing systems and suddenly transition to a modern platform based on Greenplum. Greenplum makes it very easy to undertake such a transition because of its compatibility with many legacy tools. For example, SAS users gain the advantage of big data by loading their data in Greenplum and doing the Data Step; that is, investigating the data and building different features within Greenplum. Then, they can load the features into SAS by using a module called SAS Access. They can build models on samples of data using the SAS functions with which they are familiar. They then can export the models using PMML or SAS Scoring Accelerator and apply the models on large datasets

in Greenplum during the transition period. The final phase of the transition would be moving the model into Greenplum as well when the user has gained familiarity with it.

Capabilities for modern microservices-based architectures
: A modern data platform incorporates different cloud and on-premises storage systems and needs to deploy applications that work in real time. They also need to take advantage of microservices to enable data scientists and developers to have access to data on an automated as-needed basis. Greenplum is very useful to have in such a platform. For example, if you choose to orchestrate your microservices using Spring Cloud Data Flow, you can continue to store the bulk of the data in Greenplum. The learning can happen in Greenplum and the application can be done in real time using Gemfire, taking advantage of the Greenplum–Gemfire Connector, which is discussed later in this chapter.

Apache MADlib (incubating)

Apache MADlib (incubating) is a SQL-based open source library for scalable in-database analytics. It provides parallel implementations of mathematical, statistical, and machine learning methods for structured and unstructured data.

MADlib is a comprehensive library that has been developed over several years. It comprises more than 40 principle algorithms along with many additional utilities and helper functions, as illustrated in Figure 5-1.

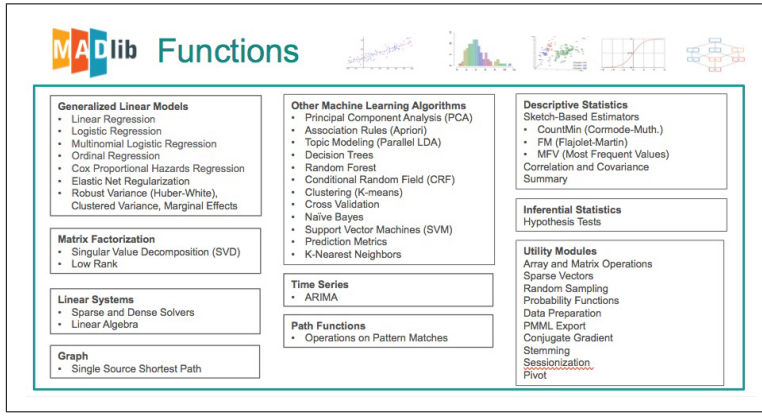

Figure 5-1. List of MADlib functions (as of January 2017)

MADlib's origin dates to 2008 when researchers from academia and the database industry began thinking about how to realize advanced statistical methods in SQL. The main participants were from University of California, Berkeley; the University of Wisconsin; the University of Florida; and Greenplum (which later became part of Pivotal). The software framework that came out of that thinking lead to MADlib, which formalized how to develop methods for the shared-nothing, scale-out capability of modern parallel database engines like Greenplum Database. MADlib also runs on PostgreSQL and Apache HAWQ (incubating) Hadoop-native SQL database.

MADlib has always been an open source project, but in September 2015, it became an incubating project under the Apache Software Foundation.

Scale and performance

Moving large amounts of data between execution engines is costly and time consuming. A core principle of MADlib is to bring the computation to where the data resides in the database, to avoid data transfer costs. Additionally, enterprises do not need to procure and maintain separate infrastructure for data warehousing and machine learning workloads, which would be costly and operationally difficult.

The primary advantage of MADlib is machine learning on very large datasets. There are many open source and commercial tools in the marketplace that offer diverse machine learning libraries, but many of them are restricted to operate on data sizes that fit in memory on a single node.

For example, R and the popular Python-based library scikit-learn are widely used by data scientists. The challenge is that many datasets in the enterprise today are too large to operate on as-is using these tools because they will run out of memory. This means that data scientists can run predictive models, they must reduce dataset size by sampling, reducing granularity, or other means. In so doing, the power of the data is diminished and the accuracy of models is compromised.

MADlib does not have this limitation. It is designed to run on MPP architectures so that datasets generally do not need to be reduced in size. This is achieved in the design by a focus on *scalability* and *performance*.

Scalability in this context means that data scientists do not need to make software changes when working with different data sizes or different cluster sizes. Using MADlib, for example, a data scientist can design and test a predictive model on a small dataset on her laptop with PostgreSQL, and then deploy *the exact same software* to a 50-node production Greenplum cluster and run the model on 1 TB of data.

The performance benefit derives from the fact that MADlib takes advantage of the MPP architecture of Greenplum Database in order to efficiently execute the iterative algorithms typical in modern machine learning methods.

Familiar SQL interface

Suppose that we want to train a linear regression model with house price as the dependent variable, with property tax, number of bathrooms, and floor size as features. Here is the SQL statement:

```
SELECT madlib.linregr_train(
           'houses',                       -- Input table
           'houses_model',                 -- Model output
           'price',                        -- Variable to predict
           'ARRAY[1, tax, bath, size]',    -- Features
           'zip'                           -- Group by
            );
```

Note that we group by zip code, meaning that we want to build separate regression models by zip code, because house prices can vary significantly from one region to the next. This is convenient because it saves calling the model multiple times for the case in which a single model for all data is not desired.

Next, we use the model we just created to predict house prices for new data:

```
SELECT madlib.linregr_predict(
           'ARRAY[1, tax, bath, size]',    -- Features
            model.coeff                    -- Model coefficients
            ) as predicted _price
FROM houses_test, houses_model model;
```

In this example, `houses_test` is the table in which the new data for which we want to predict house price resides.

R Interface

R is one of the most commonly used languages by data scientists. Although it provides a breadth of capability and is a mature library, it is not designed for working with datasets that exceed the memory of the client machine.

This is where PivotalR comes in. PivotalR is a CRAN package that provides an R interface to MADlib. The goal is to harness the familiarity and usability of R, and at the same time take advantage of the scalability and performance benefits of in-database computation that come with Greenplum.

PivotalR translates R code into SQL, which is sent to the database for execution. The database could be local running on a laptop, or it could be a separate Greenplum database cluster. Figure 5-2 demonstrates the flow.

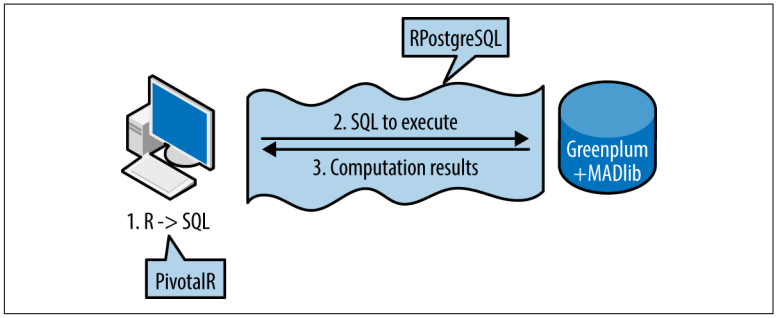

Figure 5-2. Pivotal R flow

Here what's happening in Figure 5-2:

1. R code is translated into corresponding SQL statements on the R client.
2. SQL statements are transmitted to the database where the workload is executed.
3. The result set is returned to the R client.

Here are the advantages you realize:

- Can call MADlib's in-database machine learning functions directly from R.
- Syntax is analogous to native R.

- Data does not need to leave the database.
- All heavy lifting, including model estimation and computation, is done in the database.

The principal design philosophy behind PivotalR is to not compromise the "R-ness" of the user experience. For example, the PivotalR function for linear regression, `madlib.lm()`, is very similar in look-and-feel to R's native `lm()` function.

Text Analytics

Craig Sylvester and Bharath Sitaraman

Traditional data warehouses function mostly on standard relational data types: numbers, dates, and short character strings. Understanding and using free-form text is a complex problem and is mostly done outside of a relational database. Text comes from a variety of sources, such as social media feeds, email databases, financial, healthcare and insurance documents, and from many, many more. It also comes in a variety of formats: raw text, PDF, Word, spreadsheets, and so on.

GPText is a combination of Apache Solr and Greenplum. Solr is a popular open source search engine server for enterprises. GPText 2.0, released in late 2016, is a major overhaul of previous versions and provides much-needed improvements in high availability (HA), scalability, and performance.

GPText takes the flexibility and configurability of Solr and merges it with the scalability and easy-to-use SQL interface of Greenplum. The result is a tool that enables organizations to process and analyze mass quantities of raw text data and combine it with relational data to provide powerful analytic insights.

Greenplum users can index tables filled with raw text columns into Solr indexes. This means that they can use the familiar SQL interface (SQL user-defined functions) to search quickly and efficiently through raw text data and filter via the other structured columns in their tables. Further, users can pipe the results of their searches into Apache MADlib's analytical libraries for clustering, classification, sentiment analysis, and other advanced analytics capabilities.

Brief Overview of the Solr/GPText Architecture

There are two ways that you can think about GPText architecture: physical and logical.

From a physical architecture perspective, GPText allows you to run a number of Solr[1] processes that are independent from the number of Greenplum segment processes running on the system (see Figure 5-3). This allows flexibility in terms of performance as well as memory management. If you are looking to improve indexing and search performance, you can simply increase the number of nodes running on each host. If you are looking to be more conservative in resource consumption, you can decrease your usage while maintaining parallelism.

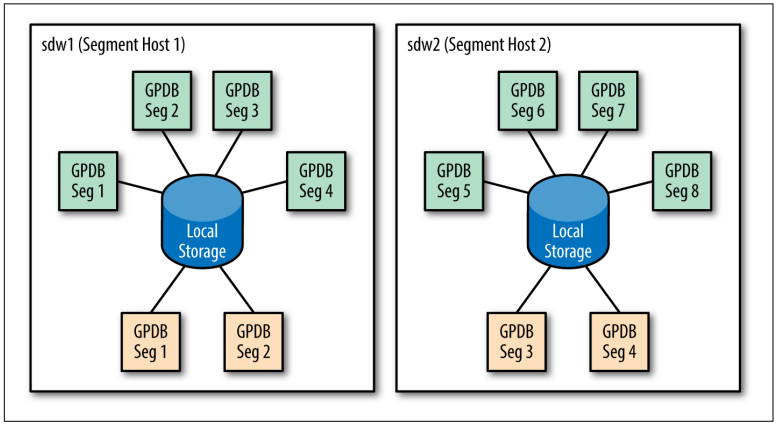

Figure 5-3. GPText physical architecture—Greenplum and Solr processes run independently

An Apache ZooKeeper service (not shown in Figure 5-3) manages the SolrCloud cluster, and you can configure it to run within the Greenplum cluster or on remote servers. ZooKeeper is installed and configured as part of the GPText installation for customers who either don't have an existing ZooKeeper service or choose not to use it. Greenplum Database users access SolrCloud services via GPText

1 Apache Solr is a server providing access to Apache Lucene full-text indexes. Apache SolrCloud is an HA, fault-tolerant cluster of Apache Solr servers. The term *GPText cluster* is another way to refer to a SolrCloud cluster deployed by GPText for use with a Greenplum cluster.

user-defined functions installed in Greenplum Databases and command-line utilities.

From a logical perspective, GPText shards indexes with the same distribution as the original table. Each shard contains a leader replica (similar to a Greenplum primary segment) and a configurable number of follower replicas (similar to Greenplum mirror segments), as depicted in Figure 5-4.

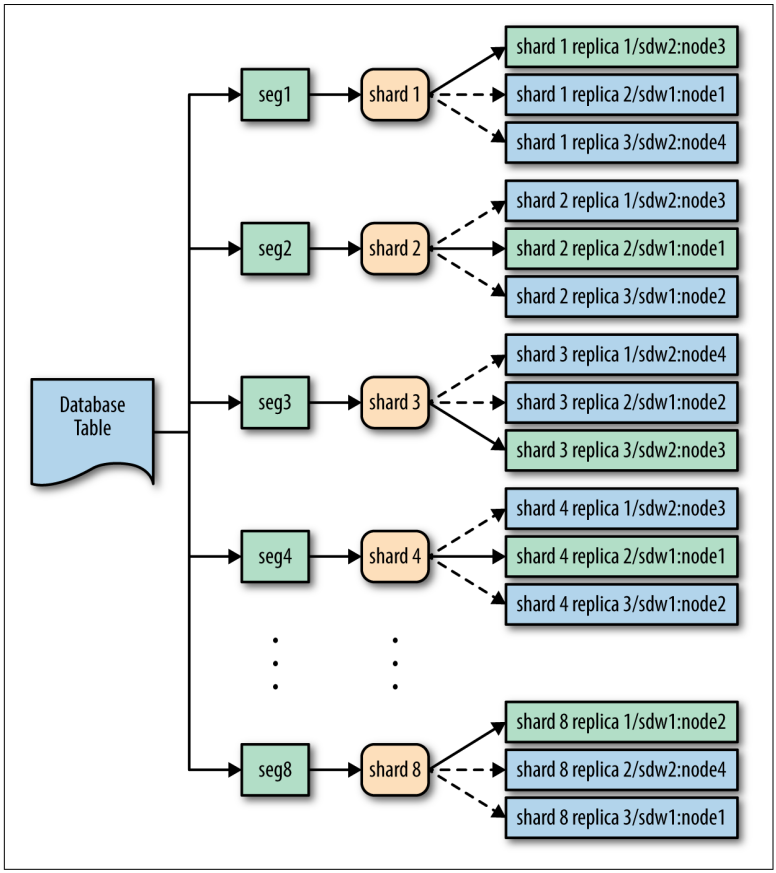

Figure 5-4. GPText logical architecture—indexes sharded with same distribution as original table

Configuring Solr/GPText

In addition to these architectural enhancements, Pivotal essentially ported Solr so that GPText provides the full flexibility you get with a standalone SolrCloud installation. Activities like configuring/

managing memory or performance tuning are fully supported. Index and search configurations such as adding/removing stop words, synonyms, protected words, stemming, and so on are also fully supported (with a provided utility to aid in these operations).

In addition to the analyzers and filters available through Solr, GPText adds two additional text processing analyzers for parsing international text (multiple languages) and social media text (including #hashtags, URLs, emojis, and @mentions).

Defining Your Analysis and Performing Text Searches

Text analysis chains determine how Solr indexes text test fields in a document. An analyzer chain is a set of Java classes that tokenize and filter the content before it is added to the index. You can define different analysis chains for indexing and querying operations.

Field analysis begins with a tokenizer that divides the contents of a field into tokens. In Latin-based text documents, the tokens are words (also called terms). In Chinese, Japanese, and Korean (CJK) documents, the tokens are the individual characters.

The tokenizer can be followed by one or more filters executed in succession. Filters restrict the query results; for example, by removing unnecessary terms (e.g., "a," "an," "the"), converting term formats, or by performing other actions to ensure that only important, relevant terms appear in the result set. Each filter operates on the output of the tokenizer or filter that precedes it. Solr includes many tokenizers and filters that allow analyzer chains to process different character sets, languages, and transformations.

When a query is submitted, Solr processes the query by using a query parser. In addition to the Standard and Lucene parsers, Solr ships with several specialized query parsers with different capabilities. For example, the Complex phrase parser can parse wildcards, and the Surround parser supports span queries: finding words in the vicinity of a search word in a document. Building queries that utilize several query parsers is possible but adds quite a bit of complexity.

To simplify the task for Greenplum users, GPText has developed a Unified Query Parser (UQP) that can process queries involving Boolean operators, complex regular expressions, and proximity searches in a single query. This provides users a clean interface to

write extremely complex queries in order to search their data effectively. Take a look at the following query:

```
SELECT
   q.id,
   q.score
FROM content t,
   gptext.search('demo.public.content',
   '{!gptextqp}"love OR hate" 2W "pic* okra"',
   null) q
WHERE
   t.id = q.id;
```

This query searches for documents containing the words "love" or "hate" within two words of the term `pic* okra` (which would match "pickled okra" or "picking okra" or misspelling like "picling okra"). However, if a specialized search requires a specific Solr query parser, that can be accomplished by specifying the parser at query time. For example, the search option `defType=dismax` will target the user of the Solr DisMax query parser.

The capability to perform faceted searches and queries (for data categorization or segmenting), highlighting search terms, and generating term vectors are all present in GPText.

One of the powers of Greenplum is combining advanced text searching with the parallel enabled statistical and machine learning algorithms available in MADlib. For example, topic modeling through the Latent Dirichlet Allocation (LDA) modeling or k-means cluster analysis is facilitated through GPText.

Administering GPText

Pivotal provides a full range of options for administering and monitoring the GPText cluster.

Configuration parameters used with GPText (such as various timeout values, HA parameters, indexing variables) are built in to GPText with default values. You can change the values for these parameters by setting the new values in a Greenplum Database session. The new values are stored in ZooKeeper. GPText indexes use the values of configuration parameters when the index is created.

Regarding access rights, security in GPText is based on Greenplum security. The privileges needed to execute GPText functions depend

on the privileges of the database table that is the source for the index.

There are command-line tools provided by Pivotal for monitoring and managing the ZooKeeper service and for monitoring the state of the SolrCloud cluster. The zkManager tool provides a means to view the state of the ZooKeeper processes and, if using the included ZooKeeper install, provides options to start and stop the ZooKeeper service. You can view status information for the SolrCloud cluster by using the gptext-state tool.

As mentioned earlier in this section, the primary focus areas of the GPText 2.0 release are HA and scalability. To meet those goals, several important features have been added:

- One major highlight is HA on the same order as Greenplum. GPText now handles cases in which the primary Greenplum segment fails, the primary Solr instance fails, or even both.
- Along those same lines, GPText now provides the capability to recover failed GPText nodes; for example, after a failed Greenplum host has been recovered.
- Backup and restore of Solr indexes has been added.
- GPText now has the ability to expand in the same manner as Greenplum, either by increasing the number of GPText nodes on existing/new hosts or increasing the number of shards to keep synchronized with additional Greenplum segments after an expansion (reindex required).

Learning More

The Pivotal Data Science group is an integral part of the Greenplum community and frequently publishes blogs about their work. You can also learn more at *https://content.pivotal.io/data-science*.

The Apache MADlib site has a wealth of information, including documentation with good examples, free downloads of the code, and the actual source code itself.

There is a MADlib user forum on which MADlib users and interested parties can submit questions, browse the archives, and suggest new features. There is a similar forum for MADlib developers that deals with technical issues relating to the actual code base. You can

subscribe to the either forum by using the link at the MADlib community page.

For a historical view of MADlib, the original Berkeley MADlib papers spell out the goals of the project:

- *MAD Skills: New Analysis Practices for Big Data*
- *The MADlib Analytics Library*

The Pivotal GPText Documentation has sections on architecture, best practices, and HA, as well as a complete specification of the functions and management utilities.

Much of the material is also covered on a YouTube video on the Greenplum channel.

CHAPTER 6

Monitoring and Managing Greenplum

Managing an MPP database bears some similarities to managing a PostgreSQL database, yet introduces other challenges that single-instance databases do not pose. Greenplum provides a set of tools that facilitate its management. Many management tasks begin with monitoring. What's the current state of Greenplum? Users complained of slow performance yesterday at 2 p.m. What was happening? Is the storage filling up? What's the gating factor that's affecting performance? Is there sufficient memory to handle concurrent loads and analytics? Are runaway ad hoc queries hogging the system and preventing other users from getting their work done?

These are the kinds of questions addressed by Greenplum Command Center, a utility bundled with Pivotal Greenplum, the commercial version of the Greenplum Database.

Greenplum Command Center

Tim McCoy

A major task for database administrators (DBAs) is workload management. To do this efficiently requires tools that accurately and quickly answer questions about current and past system performance. A common occurrence is the user who tells the DBA that a query that has run well in the past ran poorly yesterday at 2 p.m. Another is a query that seems to be running forever and consuming

resources that are impeding the performance of concurrently running queries. Yet another common situation is a query that seems to have hanged. The aforementioned use cases require a monitoring tool and a method of controlling resource consumption, either automatically or by DBA active intervention.

Greenplum Command Center (GPCC) is an Enterprise Greenplum tool that presents DBAs with a wealth of information about server and database performance in a web-based interface. The current version (GPCC v3) is a complete redesign of the previous version. Its GUI interface presents information about the state of the database system to the DBAs that will enable them to spot anomalies in performance. The v3 redesign enables GPCC to spot such anomalies even under changing conditions and increases in data as well as system usage.

GPCC displays clear, crisp metrics and, perhaps most important, allows DBAs to interactively query back over time to examine the state of the Greenplum cluster yesterday at 2 p.m. when the user complained of poor performance. To facilitate this kind of exploration, Greenplum runs agents on the segment hosts that monitor and report on resource consumption and database activity. It stores the collected information in the gpperfmon database and various logs on the master and segment hosts

Figure 6-1 shows the GPCC dashboard, which provides a quick overview of the Greenplum cluster.

In the pane on the left side of the dashboard, click the Cluster Metrics tab to see a more detailed view of cluster activity, as illustrated in Figure 6-2.

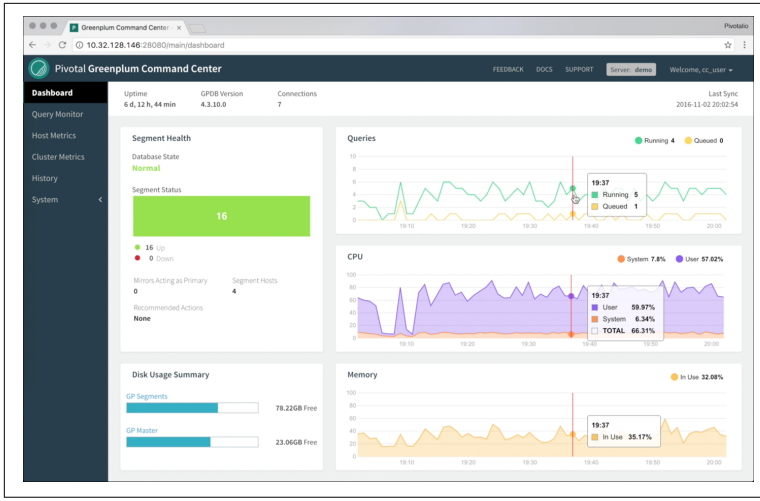

Figure 6-1. The Greenplum Command Center's dashboard view

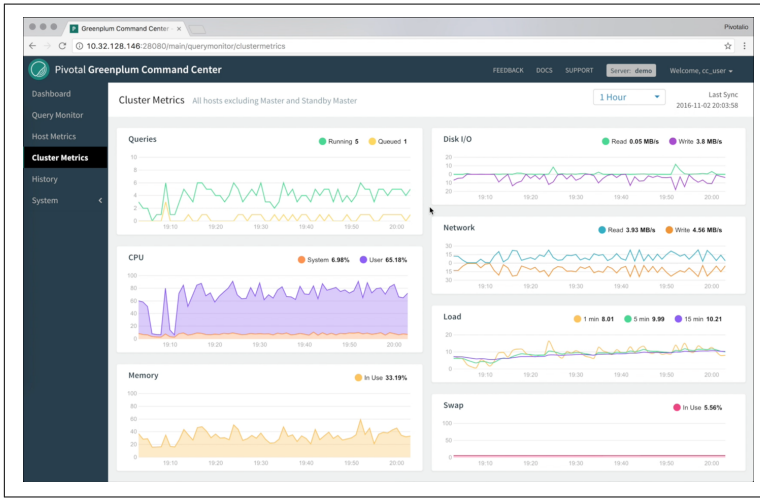

Figure 6-2. Cluster Metrics view

From a high-level view of the cluster health, DBAs can examine individual spikes in the performance visualization for more detail and to determine which queries were running at the time, as depicted in Figure 6-3. Often, heavy system load causes these variations in performance.

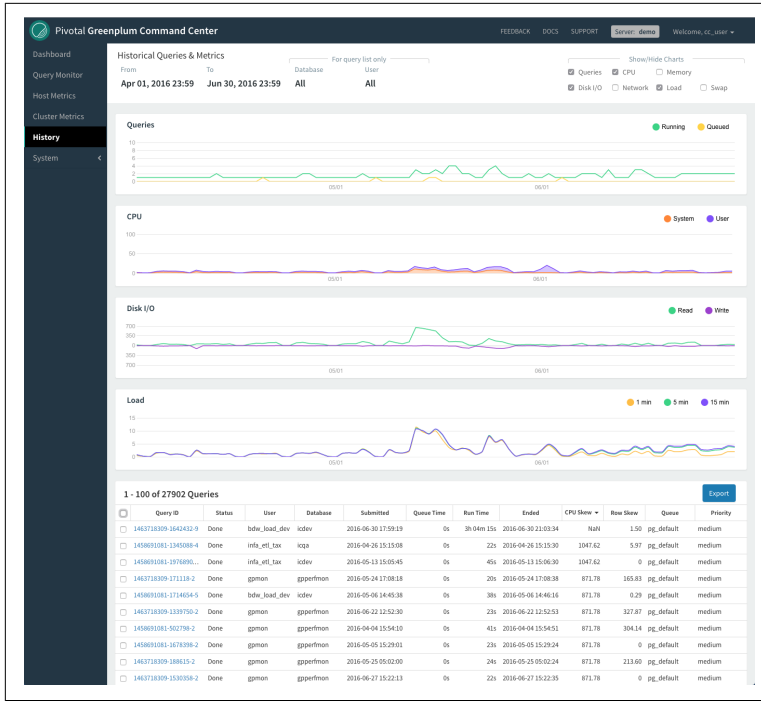

Figure 6-3. Query history view

GPCC, like many of the Greenplum tools, provides new releases independent of the release of the database itself. For the remainder of 2017, planned Command Center releases will focus on security enhancements, closer integration with workload management, interface improvement, and integration of common DBA tasks.

Resource Queues

To ensure the data warehouse is running smoothly, it's important to control resource utilization and concurrency. Data warehouse queries can consume large amounts of CPU, memory, and I/O and too many concurrent queries can completely swamp the system. To control resource usages, Greenplum, like PostgreSQL, encompasses the concept of *resource queues*. Each user or role is associated with a resource queue. Each queue defines the limits on resources that its associated roles can use.

For example, the following two statements will create a resource queue named `adhoc` and associate the role `group2` with the queue:

```
CREATE RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=3);

ALTER ROLE group2 RESOURCE QUEUE adhoc;
```

The `ACTIVE_STATEMENTS` parameter specifies that at most three statements submitted to the server from logins associated with this role can be executed at any one time. If a fourth request is submitted while the previous three are still running, the new request will be held in a queue until a slot is available for it to be run.

In addition to the active statement limit, the queue definition can contain limits on memory utilization and/or maximum estimated query cost.

Estimated cost is a bit of an amorphous metric and can vary from the use of the legacy query optimizer in comparison to the Pivotal Query Optimizer (aka GPORCA). Given this background, we suggest that estimated cost of a plan not be used.

```
CREATE RESOURCE QUEUE etl
  WITH (ACTIVE_STATEMENTS=3, MEMORY_LIMIT = '500MB');
```

In addition to these queue controls, several GUCs (or system configuration parameters) control memory utilization for the system

CPU usage for queues is managed by the priority setting. It can be `LOW`, `MEDIUM`, `HIGH`, or `MAX`. Compared to active statements or memory, the priority setting is less well defined. As a general rule, the higher the setting, the more CPU.

```
CREATE RESOURCE QUEUE poweruser
  WITH (ACTIVE_STATEMENTS=3, MEMORY_LIMIT = '1500MB',
  PRIORITY = HIGH);
```

The system sets roles that have no assigned resource queue to the `pg_default` resource queue. As with many defaults, it's best practice to not use them, but to create resource queues and assign roles to them as appropriate. A typical data warehouse implementation might have a resource queue for ETL jobs; another for ad hoc queries; another for standard reports; and yet another for short, high-priority queries. Only certain statements are actually affected by the resource queue mechanism. By default, these include `SELECT`, `SELECT INTO`, and `CREATE TABLE AS SELECT`. In an analytic data warehouse, these are the statements that are most likely to consume large amounts of resources.

The following query reveals the status of the queues:

```
select * from gp_toolkit.gp_resqueue_status;

-[ RECORD 1 ]--+------------
queueid        | 228978
rsqname        | power
rsqcountlimit  | 3
rsqcountvalue  | 0
rsqcostlimit   | -1
rsqcostvalue   | 0
rsqmemorylimit | 1.57286e+09
rsqmemoryvalue | 0
rsqwaiters     | 0
rsqholders     | 0
-[ RECORD 2 ]--+------------
queueid        | 6055
rsqname        | pg_default
rsqcountlimit  | 20
rsqcountvalue  | 0
rsqcostlimit   | -1
rsqcostvalue   | 0
rsqmemorylimit | -1
rsqmemoryvalue | 0
rsqwaiters     | 0
rsqholders     | 0
```

A fuller description of the resource queues is available in the Pivotal documentation.

Resource queues do an important job in proactively managing workload but they do not respond to issues with dynamic resource consumption. For example, after a query leaves a queue and begins executing, the queue mechanism does nothing to further constrain its behavior. Poorly written SQL can easily overwhelm a system.

Many DBAs have a variety of self-written scripts that solve the problems of monitoring and altering running queries. In 2016, Greenplum introduced two important tools that make this aspect of database administration much easier and more automated.

Greenplum Workload Manager

Bharath Sitaraman and Oz Basarir

The goal of a workload manager is to increase throughput, performance predictability, and stability of a Greenplum cluster. One important role is to prevent runaway queries and deal with other

unpredictable events that cannot be prevented by the resource queue mechanism. One bad query can dominate the cluster, not just in Greenplum, but in all databases. Because this cannot be detected by the resource queue proactively, DBAs can use Greenplum Workload Manager (WLM) to detect and resolve such problems.

WLM uses rules to determine its actions. A rule consists of a few parts: a condition and an action to take when the condition applies. The condition looks at a statistic and a limit. For example, here's the rule to log a query that runs for more than five minutes:

```
gpdb_record(message='query runtime >5 min')
   when session_id:host:pid:runtime > 300
```

Following is the rule to kill a query that runs for more than 10 minutes, consumes more than 67 percent CPU, and does a `SELECT` on the `foobar` table:

```
host:pg_cancel_backend()when session_id:host:total_cpu > 67
  and session_id:host:pid:runtime > 600
  and session_id:host:pid:current_query =~ /select.*foobar/
```

There is an action to throttle a query; that is, limit its CPU resource, which should be a first step before killing queries. A more drastic action is to use `pg_terminate_backend()`, which would terminate the user session. It's a best practice to log actions for a bit to get a feel for activity in Greenplum before taking actions that might affect the user community.

There are dozens of different statistics available for formulating rules. They are broadly separated into categories such as I/O statistics, CPU utilization, memory utilization, time of day and day of week, and data and memory skew. With these datums and Boolean logic, DBAs can create simple and complex rules to allocate resources on a Greenplum cluster. You can view the full list of datums in the WLM documentation.

Greenplum Management Utilities

Oak Barrett

Administering a massively parallel database system requires DBAs to have a variety of tools for the various situations the cluster will encounter. Although Greenplum supports many of the PostgreSQL utilities, these fall short in the MPP world. To fill this void, Green-

plum provides a set of command-line tools to aid administrators not only in their day-to-day work, but with the more specialized operations that they might occasionally face. These tools are provided as part of the Greenplum's Management Utilities suite. Almost all of these tools should be restricted to the DBA and other management staff.

In addition to the management tools, there are a variety of user utilities. In reality, these are almost always run by the DBA, as per best practices. They usually are command-line covers for operations that can be done in SQL and include such activities as creating a database, creating a language, and creating a user.

Below is a brief introduction to several of the common and more specialized tools available in the suite.

Common Tasks: Performed on a Regular Basis

Following are some tasks that you must carry out on a regular basis:

Starting the cluster: `gpstart`
: Orchestrates the parallel startup of all the individual PostgreSQL instances across the master, primary, and mirror segments in the Greenplum.

Stopping the cluster: `gpstop`
: A flexible utility allowing DBAs to stop, restart, or, most often, reload runtime settings found in the *pg_hba.conf* and *postgresql.conf* files without the need to take the database offline.

Monitoring the state of the cluster: `gpstate`
: Quickly provides information regarding the state of your Greenplum cluster, including the validity of the master, standby master, primary, and mirror segments.

Backing up the database: `gpcrondump`
: This backup utility provides options for full or partial backups, which are filtered at the database, schema or table levels. Greenplum also supports Incremental backups at the partition level for append-optimized tables. Local, mounted, and popular third-party backup systems such Dell-EMC DataDomain and Veritas NetBackup are supported.

Transferring data across clusters: `gptransfer`
: Moving data and objects from one cluster to another is a common task This utility provides users a flexible, high-speed option that uses gpfdist and external tables. This is commonly used to migrate full databases to new clusters or populate a subset of data into development/sandbox environments.

Analyzing system performance: `gpcheckperf`
: Provides tooling to check underlying performance of the cluster on which Greenplum is installed. Includes disk I/O, memory bandwidth, and network performance tests. Ideally, DBAs should run this before installing the Greenplum software. It often indicates hardware or system problems in the cluster.

Performing system catalog maintenance: `gpcheckcat`
: Checks the Greenplum catalog for inconsistencies across a variety of scenarios including Access Control Lists (ACLs), duplicate entries, foreign keys, mirroring, ownership, partition integrity, and distribution policy.

Interacting with the segment servers: `gpssh` *and* `gpscp`
: Keeping configuration files consistent across large multinode systems can be difficult. These two tools allow for administrators to run commands and copy files to all nodes via the master server, without the need to directly log in to each individual segment host.

Specialized Tasks: Performed as Needed

You need to perform the tasks that follow as the need arises:

Initializing a new cluster: `gpinitsystem`
: Along with configuration files, this utility defines the structure of the system, including a standby master, the number of data segments, mirroring, data directories, port ranges, and cluster name.

Expanding an existing cluster: `gpexpand`
: There comes a time in many MPP systems life when additional disk and CPU is needed to handle the big data workloads. To easily allow users to add segments to their Greenplum cluster, this tool provides the mechanism to allow the system to be expanded as well as tables and data to be redistributed onto the new nodes.

Adding mirrors to an existing cluster: `gpaddmirrors`
: In some instances, such as a development sandbox or test system, the Greenplum cluster is initialized without mirrors. Recommended best practice for production systems is to enable mirroring. If this is not done during the initialization process, you can invoke this utility after-the-fact to create mirrors of the primary data segments.

Restoring the database: `gprestore`
: You can restore Greenplum objects and data on the original cluster or a different one from which the backup was taken. Similar to backup, restore allows for selective filtering at the database, schema and table levels and supports Dell-EMC DataDomain and Veritas NetBackup.

Recover from a primary segment failure: `gprecoverseg`
: One of Greenplum's features is high availability. This is partly accomplished by mirroring the primary data segments in case of hardware failures (disk, power, etc.). When a primary segment failsover to the mirror, the cluster continues functioning. There comes a time when the original primary segment needs to be reintroduced to the cluster. This utility synchronizes the primary with the mirror based on the transactions that have occurred since the primary failed. Recovery is performed online, while the system is available to end users.

This is a high-level overview of the 35-plus management utilities the Greenplum Database provides. You can find other utilities, such as gpactivatestandby, gpmigrator, gpdeletesystem and gpfilespace, in the Management Utility Reference Guide found in the Greenplum documentation.

Learning More

A good starting point for learning more about the Greenplum Command Center is Tim McCoy's YouTube video.

Following that, the Pivotal Greenplum Command Center documentation has both release notes as well as detailed information on installation, supported platforms, access, and upgrades from previous versions.

Workload management in Greenplum begins with resource queues, an understanding of which is critical to good concurrency.

A place to begin more in depth on Workload Manager is Oz Basarir's YouTube Tutorial.

There is a roundtable discussion on workload management on a Greenplum Chat YouTube video. Be aware, though, that it's older than the tutorial and might not reflect the newer features.

The Pivotal Greenplum Workload Manager documentation has the most detail.

Greenplum memory management is critical. It takes two forms: the Linux OS memory management and the internal Greenplum memory controls. The importance of memory management cannot be overstated and the following articles provide much useful information.

Jon Roberts discusses them both in this PivotalGuru post, as does the Pivotal discussion article.

Managing a large Greenplum database is generally not as complicated as managing a mission-critical OLTP database. Nonetheless, many find it useful to attend the Pivotal Academy Greenplum Administrator class.

More detail on the tools to manage Greenplum is available in the Utility Guide.

CHAPTER 7

Integrating with Real-Time Response

John Knapp

GemFire-Greenplum Connector

We designed Greenplum to provide analytic insights into large amounts of data. We did not design it for real-time response. Yet, many real-world problems require a system that does both. At Pivotal, we use GemFire for real-time requirements and the GemFire-Greenplum Connector to integrate the two.

Problem Scenario: Fraud Detection

As more businesses interact with their customers digitally, ensuring trustworthiness takes on a critical role. More than 17 million Americans were victims of identity theft in 2014, the latest year for which statistics are available. Fraudulent transactions stemming from identity theft—fraudulent credit card purchases, insurance claims, tax refunds, telecom services, and so on—cost businesses and consumers more than $15 billion that year, according to the Department of Justice's Bureau of Justice Statistics.

Detecting and stopping fraudulent transactions related to identity theft is a top priority for many banks, credit card companies, insurers, tax authorities, as well as digital businesses across a variety of industries. Building these systems typically relies on a multistep process, including the difficult steps of moving data in multiple formats between analytical systems, which are used to build and run predic-

tive models, and transactional systems, where the incoming transactions are scored for the likelihood of fraud. Analytical systems and transactional systems serve different purposes and, not surprisingly, often store data in different formats fit for purpose. This makes sharing data between systems a challenge for data architects and engineers—an unavoidable trade-off, given that trying to use a single system to perform two very different tasks at scale is often a poor design choice.

Supporting the Fraud Detection Process

Greenplum's horizontal scalability and rich analytics library (MADlib, PL/R, etc.), help teams quickly iterate on anomaly detection models against massive datasets. Using those models to catch fraud in real time, however, requires using them in an application. Depending on the velocity of data ingested through that application, a "fast data" solution might be required to classify the transaction as fraudulent or not in a timely manner. This activity involves a small dataset and real-time response which needs to be informed by the deep analytics performed in Greenplum. This is where Pivotal GemFire, a Java-based transactional in-memory data grid, supports fraud detection efforts, as well as other use cases like risk management.

Problem Scenario: Internet of Things Monitoring and Failure Prevention

Increasingly, automobiles, manufacturing processes, and heavy duty machinery are instrumented with a profusion of sensors. At Pivotal, the data science team has worked with customers to use historical sensor data to build failure prediction and avoidance models in the Greenplum database. As well tuned as these models are, Greenplum is not built to quickly ingest new data and respond in subsecond time to sensor data that suggests, for example, that certain combinations of pressure and temperature and observed faults are predicting conditions are going awry in a manufacturing process and that operator or automated intervention must be quickly performed to prevent serious loss of material, property, or even life.

For situations like these, disk-centric technologies are simply too slow; in-memory techniques are the only option that can deliver the required performance. Pivotal solves this problem with GemFire, an in-memory data grid.

What Is GemFire?

GemFire is an in-memory data grid based on the open source Apache Geode project. Java objects are stored in memory spread across a cluster of servers so that data can be ingested and retrieved at in-memory speeds, several orders of magnitude faster than disk-based storage and retrieval. GemFire is designed for very-high-speed transactional workloads. Greenplum is not designed for that kind of workload, and thus the two in tandem solve business problems that combine the need for both deep analytics and low-latency response times.

The GemFire-Greenplum Connector

GemFire-Greenplum Connector (GGC) is an extension package built on top of GemFire that maps rows in Greenplum tables to plain old Java objects (POJOs) in GemFire regions. With the GGC, the contents of Greenplum tables now can be easily loaded into GemFire, and entire GemFire regions likewise can be easily consumed by Greenplum. The upshot is that data architects no longer need to spend time hacking together and maintaining custom code to connect the two systems.

GGC functions as a bridge for bidirectionally loading data between Greenplum and GemFire, allowing architects to take advantage of the power of two independently scalable MPP data platforms while greatly simplifying their integration. GGC uses Greenplum's external table mechanisms (described in Chapter 3) to transfer data between all segments in the Greenplum cluster to all of the GemFire servers in parallel, preventing any single-point bottleneck in the process.

In fraud-detection scenarios, this means that it is now seamless to move the results of predictive models from Greenplum to GemFire via an API. After the scores are applied to incoming transactions, those transactions deemed most likely to be fraudulent can be presented to investigators for further review. When cases are resolved, the results—whether the transaction or claim was fraudulent—can be easily moved back to Greenplum from GemFire to continuously improve the accuracy of the predictive models.

In the Internet of Things use case, sensor data flows into GemFire where it is scored according to the model produced by the data science team in Greenplum. In addition, GemFire pushes the newer

data back to Greenplum where it can be used to further refine the analytic processes. There is much similarity between fraud analytics and failure prevention. In both cases, data is quickly absorbed into GemFire, decisions are made in subsecond time, and the data is then integrated back into Greenplum to refine the analysis.

Learning More

The product has evolved since this introductory talk.

There is more detailed information about GGC in the Pivotal documentation.

GemFire is very different from Greenplum. A brief tutorial is a good place to begin learning about it.

CHAPTER 8

Optimizing Query Response

Venkatesh Raghavan

Fast Query Response Explained

When processing large amounts of data in a distributed environment, a naive query plan might take orders of magnitude more time than the optimal plan. In some cases, the query execution will not complete, even after several hours, as shown in our experimental study.[1] Pivotal's Query Optimizer (PQO) is designed to find the optimal way to execute user queries in distributed environments such as Pivotal's Greenplum Database and HAWQ. The open source version of PQO is called GPORCA. To generate the fastest plan, GPORCA considers thousands of alternative query execution plans and makes a cost-based decision.

As with most commercial and scientific database systems, user queries are submitted to the database engine via SQL. SQL is a declarative language that is used to define, manage and query the data that is stored in relational/stream data management systems.

Declarative languages describe the desired result, not the logic required to produce it. The responsibility for generating an optimal execution plan lies solely with the query optimizer employed in the database management system. To understand how query processing

1 "New Benchmark Results: Pivotal Query Optimizer Speeds Up Big Data Queries Up To 1000x"

works in Greenplum, there is an excellent description in the documentation.

GPORCA is a top-down query optimizer based on the Cascades optimization framework,[2] which is not tightly coupled with the host system. This unique feature enables GPORCA to run as a standalone service outside the database system. Therefore, GPORCA supports products with different computing architectures (e.g., MPP and Hadoop) using a single optimizer. It also takes advantage of the extensive legacy of relational optimization in different query processing paradigms like Hadoop. For a given user query, there can be a significantly large number of ways to produce the desired result set, some much more efficient than others. While searching for an optimal query execution plan, the optimizer must examine as many plans as possible and follow heuristics and best practices for choosing and ignoring alternative plans. Statistics capturing the characteristics of the stored data, such as skew, number of distinct values, histograms, and percentage of null values, as well as the cost model for the various operations are crucial ingredients that the query optimizer relies on when navigating the space of all viable execution plans. For GPORCA to be most effective, it is crucial for DBAs to maintain up-to-date statistics for the queried data.

Until recently, Greenplum used what is referred to as the legacy query optimizer (LQO). This is a derivative of the original PostgreSQL planner that was adapted to the Greenplum code base initially. The PostgreSQL planner was originally built for single-node PostgreSQL optimized for OLTP queries. In contrast, an MPP engine is built for long running Online Analytical Processing (OLAP) queries. For this reason, the PostgreSQL planner was not built with an MPP database in mind. Although features like join ordering were carefully thought out, the architecture and design choices make maintenance and adding new features increasingly difficult.

At the end of 2010, Greenplum began an internal effort to produce a modern query optimizer, which made its first appearance in Greenplum version 4.3.5. as GPORCA.

2 G. Graefe. 1995. "The Cascades Framework for Query Optimization." *IEEE Data Eng Bull*, 18(3).

What makes GPORCA particularly useful is its ability to generate more efficient code for some of the complex situations that commonly arise in analytic data warehouses, including the following:

- Smarter partition elimination
- Subquery unnesting
- Common table expressions (CTE)
- Multilevel partitioning
- Improved join ordering
- Join aggregate reordering
- Sort order optimization
- Skew awareness

Previously, the legacy query optimizer was set as the default, but as of Greenplum 5.0, GPORCA is the default query optimizer (see Figure 8-1). You can change the default at the database level or the session level by setting the GUC parameter `optimizer = on`. When enabling GPORCA, we request that users or DBAs ensure that statistics have been collected on the root partition of a partitioned table. This is because, unlike the legacy planner, GPORCA uses the statistics at the root partitions rather than using statistics of individual leaf partitions.

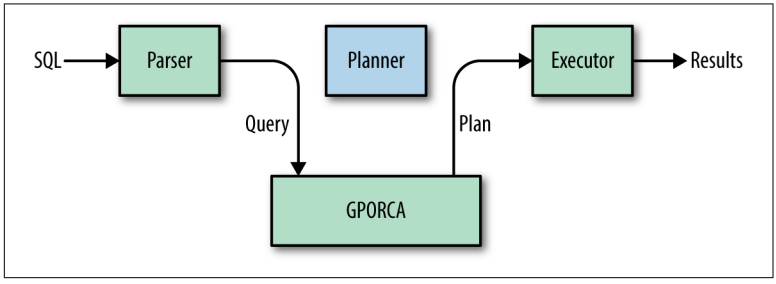

Figure 8-1. Query flow when GPORCA is enabled

Let's look at an example. Following is the schema for the table `part` from the TPC-H benchmark:

```
CREATE TABLE part (
    p_partkey integer NOT NULL,
    p_name character varying(55) NOT NULL,
    p_mfgr character(25) NOT NULL,
    p_brand character(10) NOT NULL,
```

```
    p_type character varying(25) NOT NULL,
    p_size integer NOT NULL,
    p_container character(10) NOT NULL,
    p_retailprice numeric(15,2) NOT NULL,
    p_comment character varying(23) NOT NULL
) distributed by (p_partkey);
```

Consider the correlated query shown in Figure 8-2, which fetches all parts with size greater than 40 or retail price greater than the average price of all parts that have the same brand.

```
tpch=# explain SELECT *  FROM part p1 WHERE p_size > 40 OR
p_retailprice > (SELECT avg(p_retailprice) FROM part p2 WHERE p2.p_brand=p1.p_brand);
```

Figure 8-2. Correlated subquery on the part table

Figure 8-3 presents the explain plan produced by GPORCA, the optimizer status denotes the version of GPORCA used to generate the plan.

```
                                                      QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 3:1  (slice3; segments: 3)  (cost=0.00..96490.33 rows=98133504 width=133)
   ->  Result  (cost=0.00..47850.78 rows=32711168 width=133)
         Filter: tpch500gb.part.p_size > 40 OR tpch500gb.part.p_retailprice > (pg_catalog.avg((avg(tpch500gb.part.p_retailprice))))
         ->  Hash Left Join  (cost=0.00..44622.19 rows=32711168 width=141)
               Hash Cond: tpch500gb.part.p_brand = tpch500gb.part.p_brand
               ->  Table Scan on part  (cost=0.00..3363.56 rows=32711168 width=133)
               ->  Hash  (cost=8688.86..8688.86 rows=25 width=19)
                     ->  Broadcast Motion 3:3  (slice2; segments: 3)  (cost=0.00..8688.86 rows=25 width=19)
                           ->  Result  (cost=0.00..8688.85 rows=9 width=19)
                                 ->  HashAggregate  (cost=0.00..8688.85 rows=9 width=19)
                                       Group By: tpch500gb.part.p_brand
                                       ->  Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..8688.85 rows=9 width=19)
                                             Hash Key: tpch500gb.part.p_brand
                                             ->  Result  (cost=0.00..8688.85 rows=9 width=19)
                                                   ->  HashAggregate  (cost=0.00..8688.85 rows=9 width=19)
                                                         Group By: tpch500gb.part.p_brand
                                                         ->  Table Scan on part  (cost=0.00..3363.56 rows=32711168 width=19)
 Settings:  optimizer=on
 Optimizer status: PQO version 2.13.0
(19 rows)
```

Figure 8-3. GPORCA plan for a correlated subquery on the part table

In comparison, Figure 8-4 shows an LQO plan that employs a correlated execution strategy.

```
                                                      QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)  (cost=0.00..187279528517187.19 rows=47850882 width=133)
   ->  Seq Scan on part p1  (cost=0.00..187279528517187.19 rows=15950294 width=133)
         Filter: p_size > 40 OR p_retailprice > ((subplan))
         SubPlan 1
           ->  Aggregate  (cost=1908415.76..1908415.77 rows=1 width=32)
                 ->  Result  (cost=1749348.14..1792435.54 rows=1308447 width=8)
                       Filter: p2.p_brand = $0
                       ->  Materialize  (cost=1749348.14..1792435.54 rows=1308447 width=8)
                             ->  Broadcast Motion 3:3  (slice1; segments: 3)  (cost=0.00..1741588.80 rows=1308447 width=8)
                                   ->  Seq Scan on part p2  (cost=0.00..1741588.80 rows=1308447 width=8)
 Settings:  optimizer=off
 Optimizer status: legacy query optimizer
```

Figure 8-4. Legacy query optimizer plan for a correlated subquery on the part table

NOTE

The cost models used by the two optimizers are different. For instance, the top node for the GPORCA plan has the cost of 98133504, whereas that of the legacy query optimizer is 187279528517187. These numbers make sense within a particular optimizer, but they are not comparable between the two different optimizers.

GPORCA excels on partitioned tables. By comparison, the LQO can only eliminate partitions statically. For example, if a table is partitioned by date, a WHERE clause that limits the date range would eliminate any partitions in which the limited date range could not occur. However, it cannot handle dynamic conditions in which the WHERE clause has a subquery that determines the range. Furthermore, many large fact tables in a data warehouse might have a significantly large number of partitions. The legacy planner could encounter Out Of Memory (OOM) errors in cases for which GPORCA would not.

Modern data analytics and business intelligence (BI) often produce SQL with correlated subqueries, where the inner subquery requires knowledge of the outer query. Consider the preceding example that fetches parts with size `> 40` or retail price greater than the average price of all parts that have the same brand. In the plan shown in Figure 8-4 generated by the LQO, for the tuple in the outer part table `p1`, the plan executes a subplan that computes the average part price of all parts having the same brand as the tuple from table part `p1`. This computed intermediate value is used to determine whether that tuple in `p1` will be in the query result or not. Because the legacy query optimizer plan repeatedly executes the subplan for each tuple in the part table `p1`, the plan is considered a correlated execution. Such a correlated plan is suboptimal because it does extraneous work that could be avoided. In the worst case, if all the parts belong to the same brand, we will be computing the average price one too many times.

In contrast, GPORCA generates a de-correlated plan in which it first computes average price for each brand. This is done only once. The intermediate results then are joined with the parts table to generate a list of parts that meets the user's criteria.

Un-nesting correlated queries is very important in analytic data warehouses due to the way that BI tools are built. They are also common in handwritten SQL code. This is evident by the fact that

20 and 40 percent of the workloads in the TPC-DS and TPC-H benchmarks, respectively, have correlated subqueries.

With these and other optimizations, SQL optimized by GPORCA can achieve increases in speed of a factor of 10 or more. There are other queries, albeit a small number, for which GPORCA has not yet produced an improvement in performance. As more capabilities are added to GPORCA over time, it will be the rare case for which the LQO provides better performance.

Learning More

To read more about query optimization, go to the following sites:

- *https://sites.google.com/a/pivotal.io/data-engineering/home/query-processing/wiki-articles*
- *http://engineering.pivotal.io/post/orca-profiling/*
- *https://gpdb.docs.pivotal.io/latest/admin_guide/query/topics/query-piv-optimizer.html*

CHAPTER 9

Learning More About Greenplum

This book is but the first step in learning about Greenplum. There are many other sources available from which you can gather information.

Greenplum Sandbox

Pivotal produces a single-node virtual machine for learning and experimenting with Greenplum. Although it's not for production purposes, it's a full-featured version of Greenplum, but of a limited scale. There are only two segments, no standby master, and no segment mirroring. You can download it for free, and it comes with a set of exercises that demonstrate many of the principles described in this book. As of this writing, you can find the latest version at *http://bit.ly/2qFSRb1*.

Greenplum Documentation

The Greenplum documentation answers many questions that both new and experienced users have. You can download up-to-date information, free of charge, at *http://bit.ly/2pTycmR*.

Pivotal Guru (formerly Greenplum Guru)

Jon Roberts, a long-standing member of the Pivotal team, keeps an independent website with many tips and techniques for using the Greenplum database. You can find it at *http://www.pivotalguru.com/*.

Greenplum Best Practices Guide

Part of the Greenplum documentation set is the *Best Practices Guide*. No one should build a production cluster without reading it and following its suggestions.

Greenplum Blogs

https://content.pivotal.io/blogs

Greenplum YouTube Channel

There are frequent updates to content on the YouTube channel with videos of interesting meetups, MADlib use cases, new features, and so forth:

- *http://bit.ly/gpdbvideos*
- *http://bit.ly/2r0zfkX*

Greenplum Knowledge Base

Some of the topics in the knowledge base can get a bit esoteric. This is probably not a place to start, but more suited to more experienced users:

- *https://discuss.pivotal.io/hc/en-us/categories/200072608-Pivotal-Greenplum-DB-Knowledge-Base*
- *https://discuss.pivotal.io/hc/en-us/categories/200072608*

greenplum.org

The Greenplum open source community maintains a website that contains mailing lists, events, a wiki, and other topics you will find helpful on your Greenplum voyage.

About the Author

Marshall Presser is an Advisory Data Engineer for Pivotal and is based in Washington DC. In addition to helping customers solve complex analytic problems with the Greenplum Database, he also works on development projects (currently, Greenplum in the Azure Marketplace).

Prior to coming to Pivotal (formerly Greenplum), he spent 12 years at Oracle, specializing in high availability, business continuity, clustering, parallel database technology, disaster recovery, and large scale database systems. Marshall has also worked for a number of hardware vendors implementing clusters and other parallel architectures. His background includes parallel computation and operating system and compiler development, as well as private consulting for organizations in healthcare, financial services, and federal and state governments.

Marshall holds a BA in Mathematics and an MA in Economics and Statistics from the University of Pennsylvania, and a MSc in Computing from Imperial College, London.