White Paper

MapR, Hive, and Pig on Google Compute Engine

Bring Your MapR-based Infrastructure to Google Compute Engine

MapR Technologies, Inc.
www.mapr.com

MapR, Hive, and Pig on Google Compute Engine

Bring Your MapR-based Infrastructure to Google Compute Engine

Introduction

In 2004, Google published a paper on MapReduce, a programming model that lets us rapidly process massive amounts of data on commodity hardware. In 2005, the Apache™ Hadoop® project began. Today, it is the best-known open source implementation of the MapReduce framework.

The growth in popularity of the MapReduce model has led to a revolution in Big Data[1] processing. Additional tools and languages have been developed to make it easier for data analysts, statisticians, and others to perform Big Data analysis.

In adopting Hadoop for data processing, many organizations start with the traditional model of keeping compute hardware on-site. They build out their own compute clusters, with sizable capital and operations costs. Many find the continual need to grow and maintain their compute clusters too expensive and a drain on resources.

Cloud computing now provides a compelling alternative. Using Google Compute Engine, you can move many of your existing services, including your MapR infrastructure, to the Google Cloud Platform.

There is an explosion in tools and products across the industry and on the Google Cloud Platform for storing and processing Big Data. Managed cloud-based software products and specialized components offer ease of use along with reduced usage and maintenance costs. But there will always be organizations interested in deploying self-managed components on compute nodes in the cloud.

This paper describes how you can take advantage of Google Compute Engine, with support from Google Cloud Storage, and run a self-managed MapR cluster with Apache Hive and Apache Pig as part of a Big Data processing solution.

While this paper is primarily intended for software engineers and architects with expertise in Hadoop, it is also valuable for anyone in the organization involved with making a Big Data project a success. For background on the Hadoop technologies, please see:
Appendix A: What are MapReduce, Hive, and Pig?

Scenario

Imagine that your company receives periodic data uploads from your new product in the field that initially aggregates to a few hundred megabytes. The data might be from hardware sensors built into the product, providing critical information to engineering and manufacturing to ensure its long-term reliability.

The data analyst logs in daily to the MapR compute cluster, examines the quality and applicability of data processing results, updates Hive or Pig scripts, and starts new analysis jobs.

1 Big Data commonly refers to data sets so large that they are impractical to process using traditional database- management and data-processing tools

Scenario

Your data analyst and information technology teams say that during development and the managed roll-out of the product, the original investment in your private compute cluster was insufficient. When the sales team had pushed for the beta program to expand from 20 to 40 potential customers, the team had to scramble to squeeze every last byte and CPU cycle from the "production" cluster, including minimizing, rescheduling, or removing development jobs.

Your team is now trying to forecast computing needs for the public launch. One flexible solution is to use Google Compute Engine clusters. You can add compute resources to the production cluster as customers and products are added and sensor data grows into terabytes. Development teams can bring up Compute Engine clusters during the day, with less-expensive instance types than production and then bring them down at night to save R&D budget. The release team, at last, can create a staging cluster to test new software deployments before they are pushed to production. These staging clusters could then be torn down shortly after roll-out, once production stability is ensured.

Solution Overview

Our solution entails launching a cluster of Google Compute Engine instances, installing MapR, Hive, and Pig software, and configuring each instance appropriately. Google Cloud Storage supports the software installation and the input and output of target data.

If you are not already familiar with how MapR, Pig, and Hive fit together, please see :
Appendix B: MapR Software Components.

Core MapR services are typically installed across all nodes of a cluster and managed centrally by an administrator. Hive and Pig, as client tools, may be installed once per user with different people running different versions of each. We use two Google Compute Engine operating system user ids in this solution to represent a typical installation:

- **mapr:** Userid that owns the MapR software, processes, and MapR File System (MapR-FS). An administrator may log in as this user
- **queryuser**: User id that owns the Hive and Pig query tools. A data analyst logs in as this user

A traditional MapR installation includes the following services:

- **Container Location Database (CLDB):** The metadata service that manages the file system namespace and regulates access to files by clients; multiple nodes in the MapR cluster can be configured to host the CLDB service and thus provide high availability (HA)
- **MapReduce JobTracker:** Service that manages scheduling, monitoring, and restarting job component tasks on compute nodes; MapR clusters can be configured with multiple JobTracker nodes to provide high availability (HA)
- **MapReduce TaskTracker(s):** Execute tasks as directed by the JobTracker

© 2014 MapR, Inc. All Rights Reserved.

Solution Overview

The MapR cluster on the Compute Engine sample application referenced in this paper creates one management services node and a user-specified number of worker nodes. The management services node runs the Container Location Database Service (CLDB), MapR FileServer, and MapReduce Job-Tracker. Each worker node runs an instance of the MapR FileServer and MapReduce TaskTracker. A fully HA cluster would deploy the CLDB and Jobtracker services on multiple nodes.

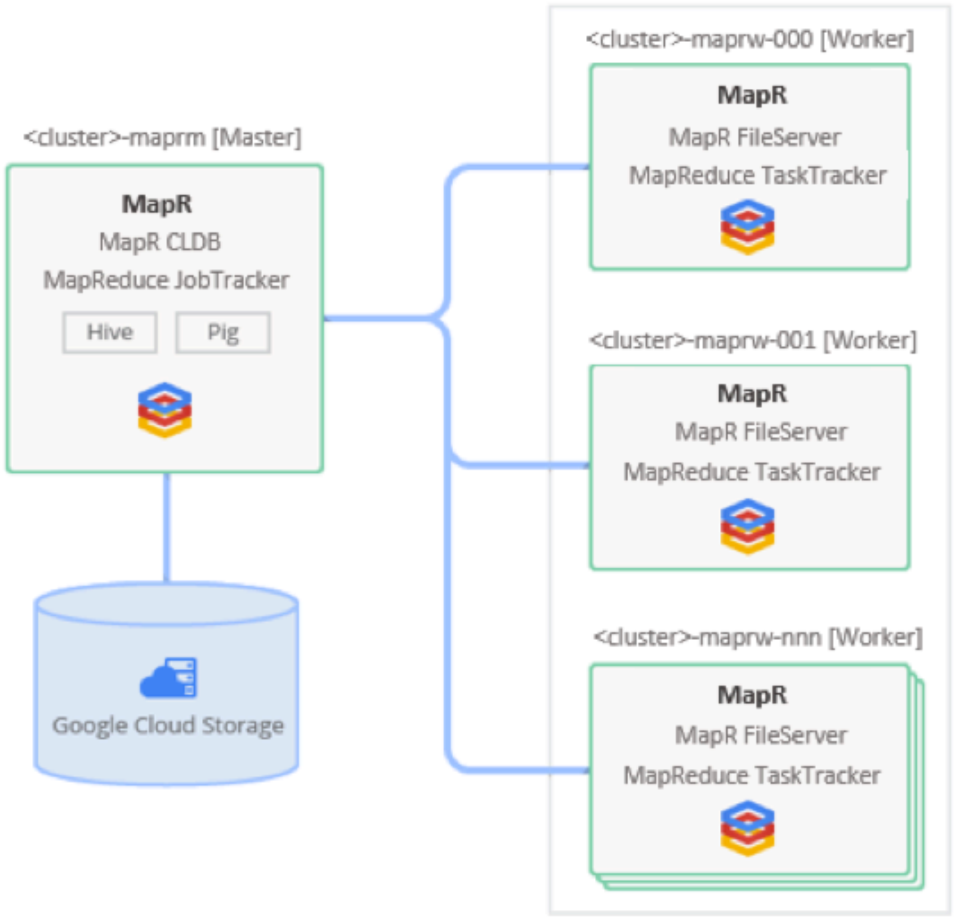
The Apache Hive and Pig sample application referenced in this paper installs Hive and/or Pig on the cluster and configures appropriate access to directories in the MapR-FS for the queryuser.

Google Cloud Storage plays two roles in the solution:

- Stages the software for installation on Google Compute Engine instances. When an instance starts, it downloads the appropriate software package from Google Cloud Storage
- Provides the durable storage for data. Data is brought onto the Google Compute Engine cluster and pushed into MapR-FS for processing. Results data is then copied from MapR-FS into Google Cloud Storage

The following figure illustrated the high-level architectural components:

Figure 1: Architectural components

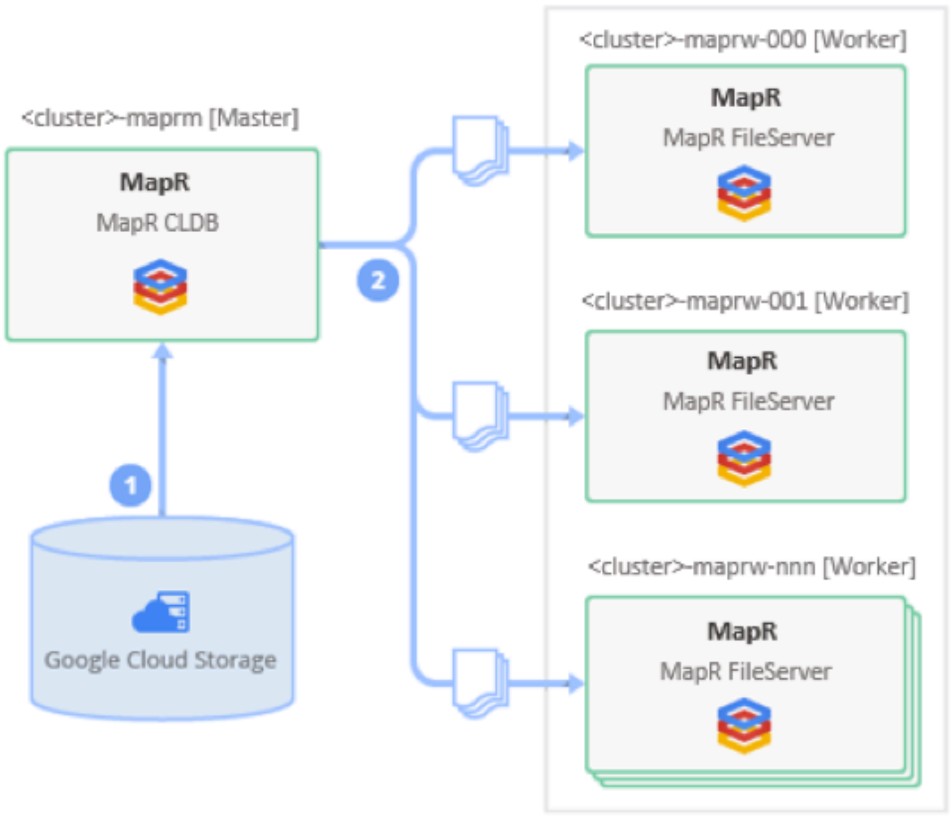


© 2014 MapR, Inc. All Rights Reserved.

Solution Overview

The following figure illustrates the flow of key operations through these components:

Figure 2: Input data flow from Google Cloud Storage to MapR-FS



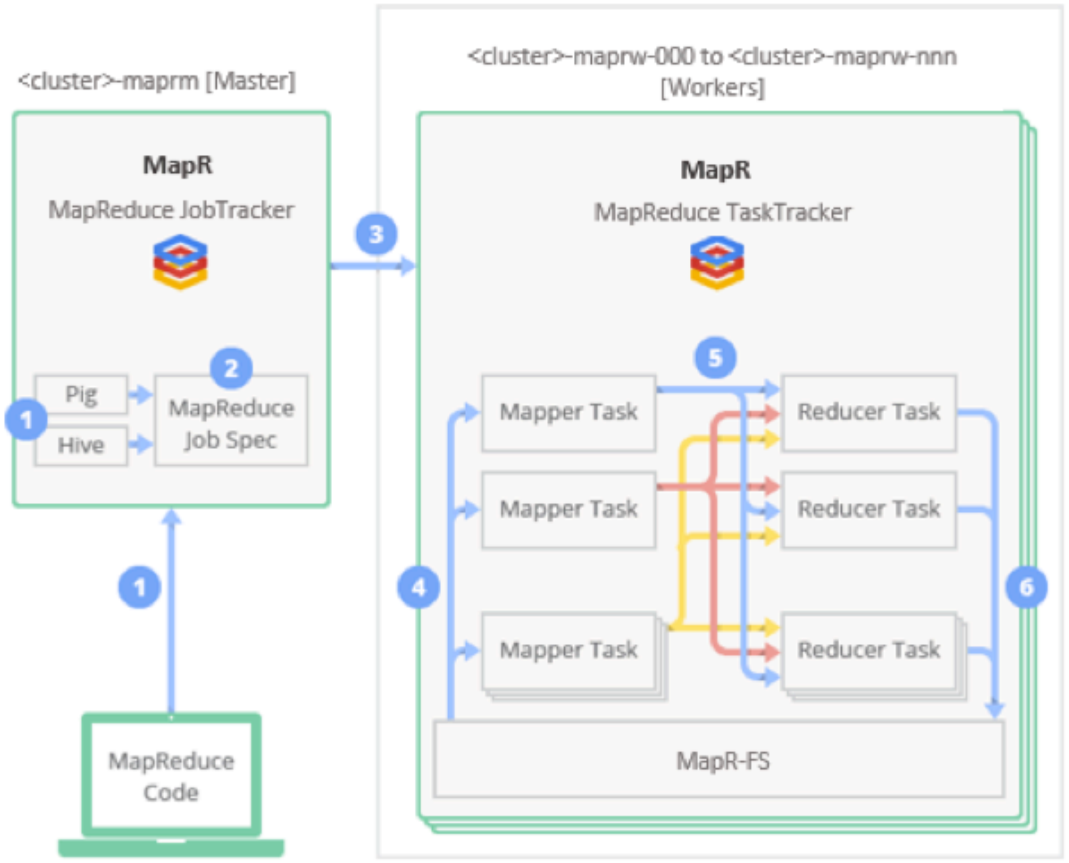
Because MapReduce tasks compute over files stored in MapR-FS, the first step is to insert data into MapR-FS. Figure 2 illustrates a simple example of a single file being transferred from Google Cloud Storage to MapR-FS.

1. On the management services node, a user (or automated task) downloads a data file from Google Cloud Storage to be pushed into MapR-FS. If data has been stored encrypted in Google Cloud Storage, it can be decrypted at this step. Data stored at rest on all Google Compute Engine disks is transparently encrypted.
2. A unique feature of MapR clusters is support of NFS for data ingest. It is possible to simply use the gsutil command to copy data from Google Storage to the NFS end-point that is the MapR file system. Parallel NFS transfers can speed the process if necessary. Alternatively, the gsutil commands can be integrated with a MapReduce job as described below.
3. The Container Location Database (CLDB) is queried as to how to distribute the file to the MapR-FS.

© 2014 MapR, Inc. All Rights Reserved.

The transfer of large datasets can take significant time for data processing. To remedy this, download and insert of large datasets can be done in parallel across Google Compute Engine instances when input data is split across multiple files in Google Cloud Storage. We give an example of this procedure after introduction of the MapReduce workflow.

Figure 3: MapReduce process flow



Once stored in MapR-FS, data is ready for processing with MapReduce. Figure 3 illustrates a MapReduce process flow:

1. A MapReduce job is created to be sent to the MapReduce JobTracker on the management services node. The job could be one of the following:

- A hand-coded MapReduce job submitted from outside the cluster, from an on-premise developer workstation
- The result of Hive or Pig code being interpreted and turned into a MapReduce job

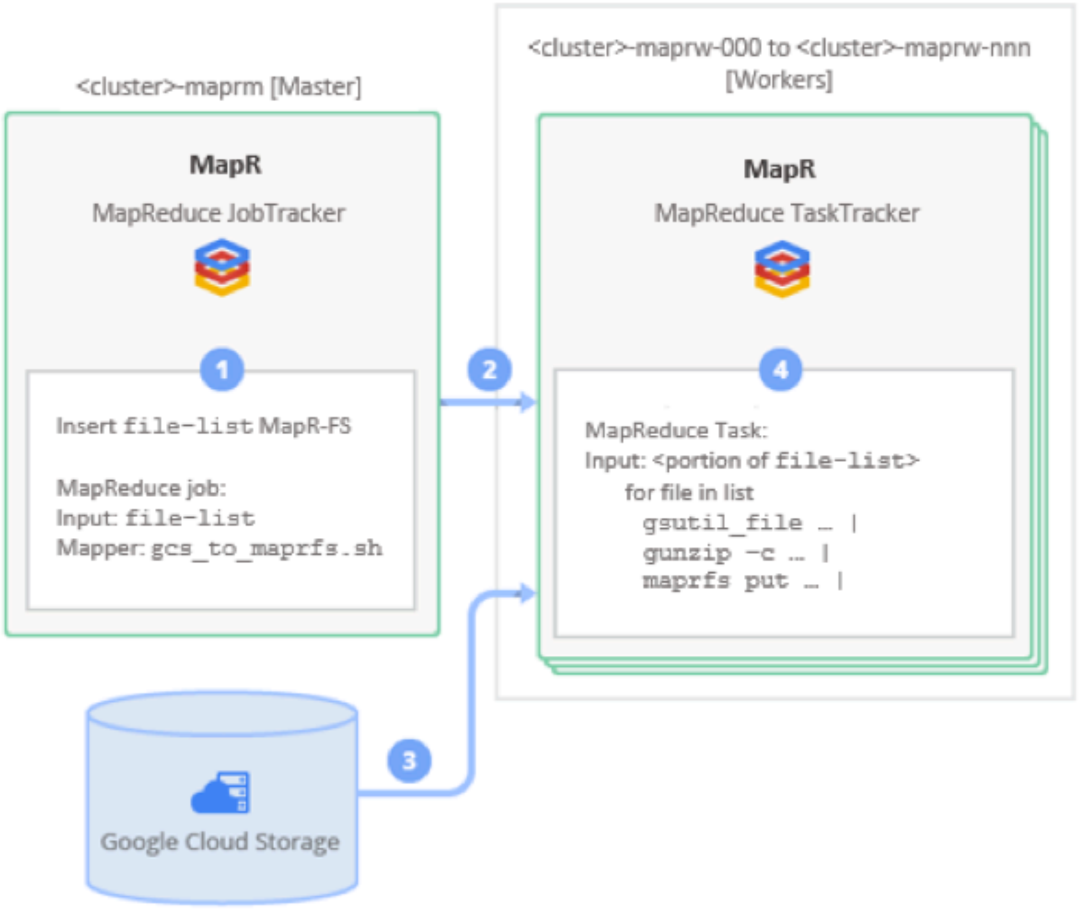
2. The MapReduce code and MapR-FS path to input data are passed to the MapReduce JobTracker.

© 2014 MapR, Inc. All Rights Reserved.

3. The MapReduce JobTracker divides the job into tasks, requesting that TaskTracker instances on different worker nodes create Map Tasks.
4. Map Tasks run the map function in parallel across the cluster. Various nodes across the cluster process each chunk of data at the same time and generate lists of key: value output.
5. Map Task output is grouped by key and passed to Reducer Tasks, with each Reducer Task being responsible for all Mapper output for a given key.
6. Reducer output is written to MapR-FS

A common post-process task is to upload the data from MapR-FS to Google Cloud Storage; following the reverse process illustrated in Figure 2.

Figure 4: Parallel data transfer, Google Cloud Storage to MapR-FS



Solution Overview

Figure 4 shows how you can use the MapReduce infrastructure to distribute data transfer from Google Cloud Storage to MapR-FS across Google Compute Engine instances.

1. A list of files to be transferred is generated and inserted into MapR-FS.
2. A MapReduce job is started with only Mappers, no Reducers. Each Map task receives a portion of the file list to act on.
3. For each file in the list, the script does the following:
   a) Downloads the file from Google Cloud Storage
   b) Uncompress it
   c) Inserts the file into MapR-FS

Distributing transfer from Google Cloud Storage across nodes distributes bandwidth usage across the network. Distributing decompression across nodes distributes CPU usage.

When files are inserted into MapR-FS, the Container Location Database (CLDB) need only be consulted for information on where to replicate data containers.

© 2014 MapR, Inc. All Rights Reserved.

Implementation Considerations

Google Compute Engine is built on virtual machines based on popular Linux distributions, and on networking infrastructure that uses industry-standard protocols. Popular tools such as Hive and Pig run on Google Compute Engine without change and with little additional consideration.

Google Compute Engine also lets you build the right network composed of the right machine types. When constructing a MapR cluster, you must consider elements such as CPU, memory, disks, and network configuration.

This section highlights key implementation options on Google Compute engine for running MapR, Hive, and Pig.

Instance Machine Types

Google Compute Engine offers a range of hardware resources, such as the number of CPU cores and gigabytes of memory, for each instance. For development and other non-production clusters, you can select low-powered instances (2 CPU, 7.5 GB). For staging and production clusters, you can select high-powered instances (8 CPU, 52 GB). You can measure organizational usage and periodically refresh clusters to increase or decrease capabilities, aligning costs with needs.

For more information on Google Compute Engine machine types, please see:
https://cloud.google.com/pricing/compute-engine.

Cluster Network Firewalls

Google Compute Engine lets you create named networks, into which instances can be added and secured. You can create firewall rules for each network, which opens different ports to hosts within the network versus outside the network.

When creating a Google Compute Engine cluster, the easiest way to make nodes accessible from your private network is to give each instance an external IP address. Each external IP address is publicly accessible to all hosts on the Internet. A more secure MapR cluster consists of a network of instances with only private IP addresses, plus a gateway instance with a public address.

Google Compute Engine Advanced Routing lets you create such a Virtual Private Network. For more information, please see:
https://developers.google.com/compute/docs/instances_and_network.

© 2014 MapR, Inc. All Rights Reserved.

Advanced Routing and Virtual Private Networks

When creating a Google Compute Engine cluster, the easiest way to make nodes accessible from your private network is to give each instance an external IP address. Each external IP address is publicly accessible to all hosts on the Internet. A more secure MapR cluster consists of a network of instances with only private IP addresses, plus a gateway instance with a public address.

Google Compute Engine Advanced Routing lets you create such a Virtual Private Network. For more information, please see:
https://developers.google.com/compute/docs/networking#settingupvpn.

Ephemeral Versus Persistent Disks

Google Compute Engine offers both ephemeral disks, in which data is lost if an instance terminates, and persistent disks, in which data remains available if an instance terminates. For more information on Google Compute Engine disks please see:
https://developers.google.com/compute/docs/disks.

Thanks to the replication built into MapR-FS and to the reliability of Google Compute Engine nodes, you can run a very low-cost cluster of instances using ephemeral disks. Such a configuration is appropriate in many cases. However, there are limitations and risks to using only ephemeral disks for production clusters.

Per-Instance Disk Size

Ephemeral disks provide a maximum of 3.45 terabytes (TB) per Google Compute Engine instance. Persistent disks provide 10 TB. You can always add more instances to attach more storage for the cluster, but this reduces the cost advantage of ephemeral disks.

Persistent disks have built in replication, which reduces but doesn't eliminate the benefits of MapR-FS replication. MapR- FS replication still has value in that all data is still available in the event of a single node failure, and replication gives the MapReduce scheduler flexibility in distributing work. By reducing the MapR-FS replication factor from the default of 3 to 2, you can store 50 percent more data with the same investment in disk storage.

© 2014 MapR, Inc. All Rights Reserved.

Instance Failure

Ephemeral disks provide no resilience in node failure. When selecting ephemeral or persistent disks for different instances, consider the following scenarios:

Loss of the CLDB Node: MapR eliminated the NameNode from its architecture and instead replicates and distributes metadata across the cluster to ensure data protection. MapR-FS provides high availability (HA) at the cluster and job levels. If a node running the CLDB service in a MapR cluster fails, a new CLDB node can continue where the failed node left off. For M5 and M7 clusters, it is common to configure multiple nodes with the CLDB service so that no manual intervention is necessary when a single node failure occurs.

Loss of a Worker Node: MapR-FS automatically replicates data by distributing copies of data containers across multiple nodes in the cluster. If a single worker node goes down, no MapR-FS data is lost. The node can be completely reconstructed when it restarts. With a persistent disk, a restarted node needs far less data-block reconciliation and rejoins a cluster more quickly than a node backed by an ephemeral disk.

Loss of a Multiple Instances or an Entire Zone: Except for scheduled data center maintenance, simultaneous loss of multiple nodes in a cluster is unlikely. If results data is being promptly copied off the MapR cluster, you can minimize the cost of such an outage by repopulating MapR-FS and restarting lost MapReduce jobs.

If the cost of such an outage is too high for you, persistent disks offer the greatest safeguard against data loss. Using persistent disks for all nodes also lets you move an entire cluster to another data center zone with the gcutil moveinstances operation.

gcutil moveinstances shuts down all instances passed to the command, takes a snapshot of their persistent disks, deletes the persistent disks, recreates the persistent disks in the new zone, and restarts the instances in the new zone. The following command moves a cluster consisting of one MapR management services node (mapr-maprm) and 10 MapR worker nodes (mapr-maprw-000, mapr-maprw-001, … mapr-maprw-009) from us-central2-a to us-central1-a:

```
gcutil --project=<project> moveinstances mapr-maprm mapr-maprw-00\\d+
--source _ zone=us-central2-a --destination _ zone=us-central1-a
```

For more information on the gcutil moveinstances, please see:
https://developers.google.com/compute/docs/gcutil/tips#moving.

© 2014 MapR, Inc. All Rights Reserved.

Implementation Considerations

User System Privileges

If the end user associated with the queryuser system account should have project-level administrative privileges, you can add him or her as a team member in the Google Compute Engine Console. That person can then connect to their management services instance via the gcutil ssh command.

People who need only have SSH access should not be added as team members, and cannot use gcutil to connect to the MapR management services node. gcutil is a tool for project owners and editors. Enabling SSH access requires the addition of the user's public key to the .ssh/authorized_keys file on the MapR management services node. This is assumed in our sample application and details can be found there.

For more information on connecting to Google Compute Engine, please see:
Http://developers.google.com/compute/docs/instances#sshing.

Hive Metastore

The Hive Metastore stores all Hive metadata in an embedded Apache Derby database in MapR-FS. The Apache Derby database only allows one connection at a time. If you want multiple concurrent Hive sessions, you can use MySQL for the Hive Metastore. MySQL can be installed on a Google Compute Engine instance, typically either on the management services or a standalone node. You can run the Hive Metastore on any machine that is accessible from Hive.

If you keep the metastore on an ephemeral disk, using either MySQL or the default Derby database, you should maintain regular backups or scripts to recreate it. An alternative is to use Google Cloud SQL, which is a highly available, fully managed MySQL service. Cloud SQL provides per-use pricing, automatic replication, and out-of-the-box secure accessibility from clients outside of the Google Compute Engine network.

The sample application includes instructions for either MySQL configuration.

© 2014 MapR, Inc. All Rights Reserved.

Sample Applications

We have two sample applications that accompany this paper:

- solutions-google-compute-engine-cluster-for-hadoop: Hadoop 1.0 installation on Google Compute Engine with ephemeral disks
- solutions-apache-hive-and-pig-on-google-compute-engine: Hive and Pig installation with Hadoop 1.0

You can download them at https://github.com/googlecloudplatform/.

Conclusion

Google's Cloud Platform provides many components for building new cloud-based applications. Google Compute Engine, however, lets you quickly migrate existing applications from costly and comparatively fixed private clusters to flexible clusters of compute nodes.

If you have your own MapR-based clusters and tools such as Hive and Pig, and face ever-increasing demands on your compute infrastructure, you can seamlessly move your applications to Google Compute Engine.

Additional Resources

MapR: http://www.mapr.com

Hadoop MapReduce Tutorial: http://hadoop.apache.org/docs/stable/mapred_tutorial.html

Hive Getting Started: https://cwiki.apache.org/confluence/display/Hive/GettingStarted

Pig Getting Started: http://pig.apache.org/docs/r0.10.0/start.html

© 2014 MapR, Inc. All Rights Reserved.

Appendix A: What are MapReduce, Hive, and Pig?

To understand the value and the components of our solution, you have to understand the value and design of MapReduce. MapReduce is a simple programming paradigm that is surprisingly powerful and has a wide variety of applications. We can show you the power of MapReduce with a real-world example.

Imagine a store with eight cash registers. At the end of the day, the manager needs a tally of all the $1, $5, $10, and $20 bills. Here are two possible solutions for accomplishing this task:

Solution 1: One Employee. In the first solution, a single employee walks from one cash register to the next, and for each bill type updates a simple ledger. This solution takes one employee and a sheet of paper with four lines on it. It is very resource efficient, but can take a long time.

Solution 2: Four Employees. In the second solution, four employees are sent to the odd-numbered registers. For each bill type, each employee writes down the value and the count. After finishing at the odd numbered register, each employee goes to the next even- numbered register. For each bill type, each employee writes down the value and count.

The four employees now exchange pieces of paper, so that one employee has all of the $1 tallies, another has all of the $5 tallies, and so on. Each employee counts the total number of one denomination.

This solution is comparatively resource costly and requires four employees and 40 lines of paper. It is also much faster. When the four employees are done, the employee in Solution 1 has tallied only two registers. As the number of cash registers—that is, the size of the data—increases, the disparity between the time to execute the solutions quickly increases.

MapReduce

Solution 2 illustrates the MapReduce paradigm. In the first step, the Mappers, working independently, examine separate chunks of data, and produced a simple “key: value” mapping, with the key being the bill denomination and the value being the count of that bill. Since the Mappers work independently and on separate portions of the total dataset, they work in parallel.

In the second step, each Reducer (independently of all other Reducers) performs an operation on all Mapper output for a single key. Each employee in this case is tasked with simply summing the count of a specific denomination across all registers. Again, the independence of the tasks lets the employees work in parallel.

We make some simplifying assumptions in this example, notably that the time required to count each register is always the same. In practice, one of the four employees could finish counting two registers before another had finished counting one, and could thus move on and count a third register. In this case, the overall task would finish even faster.

The power of MapReduce is clear. To make Solution 1 faster requires making the single employee faster—that is, a faster computer. Alternatively, when large amounts of inexpensive compute power is available, the key is to ensure that the algorithm(s) allow incremental additions of compute power to yield improved total throughput.

© 2014 MapR, Inc. All Rights Reserved.

Hive and Pig

MapReduce algorithms have been invented to handle a broad range of tasks, from simple instance counting, instance grouping, data filtering, and graph analysis, to matrix multiplication. Many use cases are extremely common and don't need to be reinvented.

Hive and Pig independently provide higher-level languages that generate MapReduce tasks to process data. Hive's HiveQL language is similar to a query language like SQL. Pig's PigLatin language is a less rigidly structured data pipelining language. Both languages solve similar data processing tasks.

Below are brief samples based on our cash-register example. Imagine that you have a single tab-delimited text file representing all bills in all cash registers. Each record is simply the register number and the bill denomination. The file is in MapR-FS at /user/queryuser/registers.txt.

Hive Sample. In 2009 Facebook published Hive - A Warehousing Solution Over a Map-Reduce Framework. Afterward, Hive became an Apache open source project.

Hive lets you assign schema information, such as table and column names and datatypes, to describe data in files stored in MapR-FS. The schema information is inserted into the Hive "metastore," a database stored inside MapR-FS.

You can implement the cash register example in HiveQL this way:

```
CREATE TABLE register (
   register _ id INT,
   bill INT
)
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY
   '\t'
STORED AS TEXTFILE;
LOAD DATA INPATH '/user/mapruser/registers.txt'
OVERWRITE INTO TABLE register;
SELECT bill, COUNT(bill) AS bill _ count
FROM register
GROUP BY bill:
```

Issuing this code block to the Hive shell generates a MapReduce job within MapR to count the bill denominations across all registers.

© 2014 MapR, Inc. All Rights Reserved.

Pig Sample. In 2006 Yahoo began developing Pig. In 2007, it became an Apache open source project.

Pig provides a procedural, relation-centric interface to data stored in MapR-FS. There is no associated metastore. Instead,the structure of data is declared in the code that sets up the relationship.

You can implement the cash register example in PigLatin this way:

```
data = LOAD '/user/mapruser/registers.txt'
   AS (register _ id: INT, bill:INT);
grp = GROUP data BY (bill);
sums = FOREACH grp GENERATE
   FLATTEN(group), COUNT(data.bill) AS bill _ count:LONG;
DUMP sums;
```

Issuing this code block to the Pig shell generates a MapReduce job within MapR to count the bill denominations across all registers.

© 2014 MapR, Inc. All Rights Reserved.

Appendix B: MapR Software Components

MapR is a complete enterprise-grade distribution for Apache Hadoop engineered to improve Hadoop reliability, performance, and ease of use. MapR's distribution is fully compatible with Apache Hadoop, HDFS, and MapReduce APIs.

An installation of MapR contains two key software components:

- MapR File System (MapR-FS)
- MapR MapReduce

MapR-FS is a fully read-write distributed file system that eliminates the single Namenode issues associated with cluster failure in standard Hadoop distributions. The MapR-FS provides high availability at the cluster and job levels to eliminate any single points of failure across the system. MapR-FS ensures that data is immediately available across multiple nodes, so that computation is flexibly deployed close to the data.

The MapR MapReduce infrastructure is modeled on the Google MapReduce framework. It provides a computing model that lets you rapidly deploy simple operations as tasks on compute nodes that are close to the input data. MapR uses highly optimized, efficient remote procedure call connections to improve shuffle performance during the shuffle phase of a MapReduce job. JobTracker can be configured across multiple nodes for failover in a MapR cluster. If one JobTracker fails, another JobTracker continues the scheduling and management tasks without any delays in the MapReduce job.

A Hive installation provides the following:

- A metastore for tagging data with "table" and "column" schema information
- An interactive and scriptable shell for SQL-like data loading and querying (HiveQL)
- Programmatic support ("Embedded Hive") for languages such as Java and Python

HiveQL query commands are turned into MapReduce jobs that process the associated data files stored in HDFS.

A Pig installation provides the following:

- An interactive and scriptable shell for procedure language data pipelining (PigLatin)
- Programmatic support ("Embedded Pig") for languages such as Java and Python

PigLatin code is turned into MapReduce jobs that process data files stored in the Hadoop file system.

Note: [1]Big Data commonly refers to data sets so large that they are impractical to process using traditional database-management and data-processing tools.

MapR delivers on the promise of Hadoop with a proven, enterprise-grade platform that supports a broad set of mission-critical and real-time production uses. MapR brings unprecedented dependability, ease-of-use and world-record speed to Hadoop, NoSQL, database and streaming applications in one unified big data platform. MapR is used by more than 500 customers across financial services, retail, media, healthcare, manufacturing, telecommunications and government organizations as well as by leading Fortune 100 and Web 2.0 companies. Amazon, Cisco, Google and HP are part of the broad MapR partner ecosystem. Investors include Lightspeed Venture Partners, Mayfield Fund, NEA, and Redpoint Ventures. MapR is based in San Jose, CA. Connect with MapR on Facebook, LinkedIn, and Twitter.

© 2014 MapR Technologies. All rights reserved. Apache Hadoop, HBase and Hadoop are trademarks of the Apache Software Foundation and not affiliated with MapR Technologies.