Distributed Systems
Assignment 7 Review & Using a Hadoop MapReduce Cluster

Paul Krzyzanowski
TA: Yuanzhen Gu
Rutgers University
Fall 2014
Part 1
Assignment 7 Review: Questions about Pregel, Spanner, & Dynamo
If a machine dies, that means some workers responsible for certain partitions of the graph died.

The Pregel framework anticipates that machines might die and asks each worker to create a checkpoint every several supersteps. The checkpoint requires each worker to save

vertex values, edge values, and incoming messages

When a machine dies:

1. Master reassigns graph partitions to the currently available set of workers
2. All workers reload their partition state from the most recent available checkpoint at the beginning of a superstep $S$, which may be several supersteps earlier than the latest superstep $S'$ that was completed before the failure
3. Computation restarts from that point, recreating the missing supersteps.

See [section 4.2]
Question 2

Even without using a combiner, Pregel tries to reduce the amount of messages that are sent among machines at each superstep. Explain how it does this.

- Messages to remote machines are buffered for delivery at the sender – one buffer per destination machine.
- When the buffer size reaches a threshold, the largest buffers are flushed, delivering each to its destination as a single message.
- In the local case, the message is placed directly in the destination vertex’s incoming message queue.

See [section 4.3]
What is external consistency?

It’s when the commit order is the same order as users actually see the transactions executed with respect to wall-clock time.

Spanner provides external consistency through its timestamped transactions and commit wait.

ACID transactions do not strictly require this. They require

- **Consistency**: the database remains in a valid state after each transaction
- **Isolation = serializability**: even if transactions are executed concurrently, the result is the same as if they executed in *some* serial order.

*External consistency is a stronger requirement than serializability*
Question 4

What is a *commit wait*?

- A commit wait is when a transaction delays its commit to ensure that the timestamp of the transaction is *definitely* in the past.

- Spanner provides a global clock (TrueTime) but the value has a window of uncertainty. Any timestamp is bound by $TT.now().earliest$ and $TT.now().latest$.

- Commit wait means
  - get transaction timestamp from TrueTime: $T = TT.now().earliest$
  - wait until that time is definitely in the past
    check $TT.now().latest$ until $TT.now().latest > T$
Dynamo exposes two operations: `get()` and `put()`

- `get(key)`
  - locates the object replicas associated with the key in the storage system.
  - returns a single object or a list of objects with conflicting versions along with a context.

- `put(key, context, object)`
  - Determines where the replicas of the object should be placed based on the associated key and writes the replicas to disk.

- **Context** is a vector timestamp that the user does not manipulate but is used by Dynamo to determine if two versions have been modified concurrently.
Question 6

What is a virtual node and what is the advantage to using them in consistent hashing over physical nodes?

• Virtual node
  – A virtual node acts like a node in the Chord-like hash table: it is responsible for a contiguous range of values
  – A physical node (machine) runs multiple virtual nodes

• Advantages of using virtual nodes
  1. Load balancing. If one node becomes unavailable, the load is evenly dispersed across the remaining nodes.
  2. A newly available node accepts a roughly equivalent amount of load from each of the other available nodes.
  3. The number of virtual nodes that a node is responsible can decided based on its capacity, accounting for heterogeneity in the physical infrastructure.
Node failures combined with concurrent updates may lead to conflicting versions of an update. How does Dynamo determine whether two updates are causally related or concurrent?

- Dynamo uses vector clocks to capture causality between different versions of the same object
  - Vector clock = list of \(<node, counter>\) pairs
  - One vector clock is associated with each version of each object
  - If all counters of the vector clock on one object \(\leq\) all corresponding counters of the vector clock on the second object, then the first object is an ancestor of the second.
  - Otherwise, the objects are concurrent and require the user to resolve conflicts
Question 8

An application can tune its use of Dynamo to achieve a desired level of performance, availability, and durability. What parameters control this?

- **N**: number of replicas (commonly 3)
  - Read & write operations involve the first $N$ healthy nodes
- **R**: minimum number of nodes that must participate in a successful *read* operation
- **W**: minimum number of nodes that must participate in a successful *write* operation
  - If $W = 1$ then Dynamo will never reject a write if there is at least one node in the system that can process it.
Question 9

When is a put operation considered successful?

In question #8:

– $W =$ minimum number of nodes that must participate in a successful write operation

For a put to be successful:

– At least $W-1$ nodes respond
Part 2

Configuring Your First Hadoop Cluster
Recap – What is Hadoop?

• An open source framework for “reliable, scalable, distributed computing”

• It gives you the ability to process and work with large datasets that are distributed across clusters of commodity hardware

• It allows you to parallelize computation and ‘move processing to the data’ using the MapReduce framework
Recap – Hadoop Architecture

HDFS Architecture

Metadata (Name, replicas, ...): 
/home/foo/data, 3, ...

Client

Read

Datanodes

Write

Rack 1

Replication

Rack 2

Blocks

Client

Datanodes

Namnode

Metadata ops

Block ops
You’ve learned MapReduce in class and in our recitation today.

Now we’ll try to have fun with it
Prerequisites

• Ubuntu Linux 12.04 LTS
• Install Java v1.7+
• Add a dedicated Hadoop system user
• Configure SSH access
• Disable IPv6
Install Java & Hadoop

• We need to install java on the cluster machines in order to run Hadoop

  `sudo add-apt-repository ppa:webupd8team/java`

  `sudo apt-get update`

  `sudo apt-get install oracle-java7-installer`

• Configure JAVA_HOME in both `~/etc/.bashrc & hadoop-env.sh`

  `export JAVA_HOME=/usr/lib/jvm/java-7-oracle`
Hadoop Configuration
Environment variables Setup

• Modify environment variables

Go back to the root and edit the `.bashrc` file

```bash
# Set Hadoop-related environment variables
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
#
# Native Path
export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export HADOOP_OPTS="-Djava.library.path=${HADOOP_PREFIX}/lib"
#
# Java path
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
#export JAVA_HOME=/usr/local/Java/jdk1.7.0_45
#
# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_HOME/sbin
```
Configure HDFS

• HDFS is the distributed file system (similar to Google’s GFS) that sits behind Hadoop instances, syncing data so that it’s close to the processing and providing redundancy
  – We should set this up first

• We need to specify some mandatory parameters to get HDFS up and running in various XML configuration files

/usr/local/hadoop/etc/hadoop/yarn-site.xml

```xml
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>
```
Start HDFS 1a

• Begin by starting the HDFS file system from the master server

• There is a script which will run the *name node* on the *master* and the *data nodes* on the *slaves*:

  $ cd /usr/local/hadoop/bin/../start-dfs.sh

• Monitor the log files on the *master* and *slaves*:

  $ tail -f /usr/local/hadoop/logs/
Or you can start all together

```bash
hduser@yuanzhen-ThinkPad-T420:/usr/local/hadoop/bin$ ls
container-executor hdfs mapred.cmd yarn
hadoop hdfs.cmd rcc yarn.cmd
hadoop.cmd mapred test-container-executor

This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh

14/11/18 21:56:22 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
Localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-hduser-namenode-yuanzhen-ThinkPad-T420.out
Localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-hduser-datanode-yuanzhen-ThinkPad-T420.out
Localhost: Java HotSpot(TM) 64-Bit Server VM warning: You have loaded library /usr/local/hadoop/lib/native/libhadoop.so.1.0.0 which might have disabled stack guard. The VM will try to fix the stack guard now.
Localhost: It's highly recommended that you fix the library with 'execstack -c < libfile>', or link it with '-z noexecstack'.
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-hduser-secondarynamenode-yuanzhen-ThinkPad-T420.out
14/11/18 21:56:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-hduser-resource manager-yuanzhen-ThinkPad-T420.out
Localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-hduser-n odemanager-yuanzhen-ThinkPad-T420.out
hduser@yuanzhen-ThinkPad-T420:/usr/local/hadoop/bin$```

© 2014 Paul Krzyzanowski
Explore Hadoop
Web Interfaces

- HDFS Namenode and check health using http://localhost:50070
- HDFS Secondary Namenode status using http://localhost:50090
NameNode 'localhost:9000' (active)

| Started:       | Tue Nov 18 21:56:31 EST 2014 |
| Version:       | 2.2.0. 1529768                |
| Compiled:      | 2013-10-07T06:28Z by hortonmu from branch-2.2.0 |
| Cluster ID:    | CID-9dd9c660-c4eb-41ff-a3b7-25143ae5997 |
| Block Pool ID: | BP-194910470-127.0.1.1-1416323569749 |

Browse the filesystem
NameNode Logs

Startup Progress

<table>
<thead>
<tr>
<th>Phase</th>
<th>Completion</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loading fsimage /usr/local/hadoop/yarn_data/hdfs/namenode/current/fsimage_00000000000000000000000017 (859 B)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>inodes (7/7)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>delegation keys (0/0)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>delegation tokens (0/0)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>Loading edits</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>/usr/local/hadoop/yarn_data/hdfs/namenode/current/edits_00000000000000000000000018-00000000000000000000000018 (1 MB) (1/1)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>Saving checkpoint</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>/usr/local/hadoop/yarn_data/hdfs/namenode inodes (7/7)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>/usr/local/hadoop/yarn_data/hdfs/namenode delegation keys (0/0)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>/usr/local/hadoop/yarn_data/hdfs/namenode delegation tokens (0/0)</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>Safe mode</td>
<td>100.00%</td>
<td>0sec</td>
</tr>
<tr>
<td>awaiting reported blocks (0/0)</td>
<td></td>
<td>0sec</td>
</tr>
</tbody>
</table>

Live nodes: 1 ( Decommissioned: 0 )
Dead nodes: 0 ( Decommissioned: 0 )
Decommissioning Nodes: 0
Number of Under-Replicated Blocks: 0
Running our first MapReduce job

see daemon
The End