Apache Hama

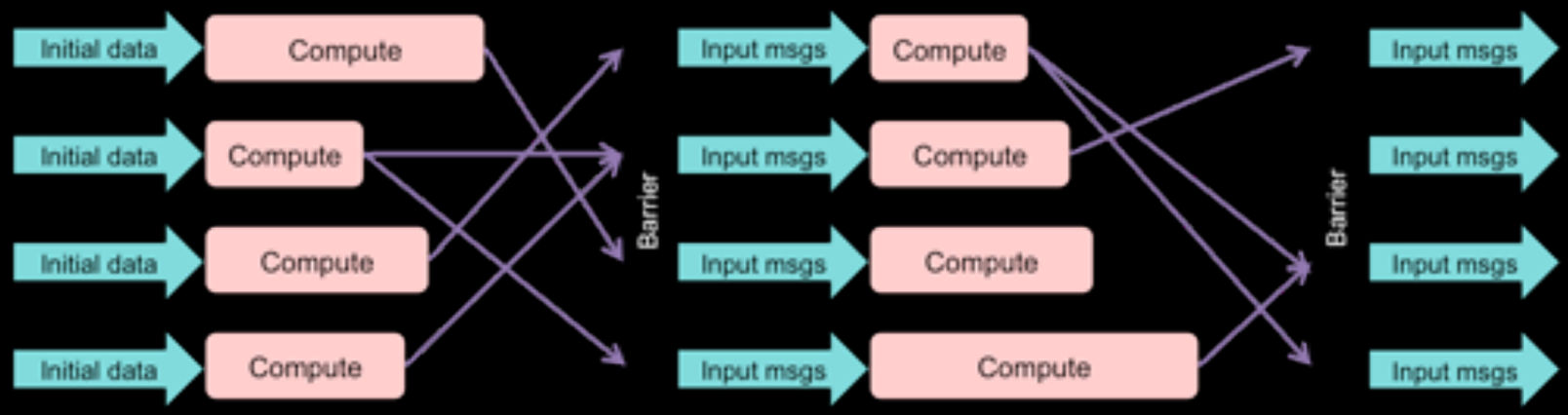
Neil Nistler

What is Apache Hama?

- Apache Hama is a general BSP computing engine on top of Hadoop, which was established in 2012 as a Top-Level Project of the Apache Software Foundation. It provides high performance computing engine for performing massive scientific and iterative algorithms on existing open source or enterprise Hadoop cluster, such as matrix, graph and machine learning.

What is BSP?

- BSP - Bulk Synchronous Parallel
- Introduced message passing and global synchronization to tackle shared memory contention.
- A BSP computer consists of:
  - Components capable of processing and/or local memory transactions
  - A network that routes messages between pairs of such components.
  - A hardware facility that allows for the synchronization of all or a subset of components.



- Each unit task executing in parallel by a peer process is called a Superstep.
- In a Superstep the peers exchange messages with each other.
- Then all the peers enter a synchronization barrier.
- After coming out of the barrier the peers work on the message sent to them in the previous Superstep.

Apache Hama Architecture

- Based on the BSP model
- Comprised of three major components:
  - BSPMaster
  - GroomServer
  - Zookeeper

BSPMaster

- BSPMaster is responsible for the following:
  - Maintaining its own state
  - Maintaining groom server status
  - Maintaining superset and other counters in a cluster
  - Maintaining jobs and tasks
  - Scheduling jobs and assigning tasks to groom servers
  - Distributing execution classes and configuration across groom servers
  - Providing users with the cluster control interface

GroomServer

- Is a process that manages the life cycle of bap tasks assigned by the BSPMaster.
- Each Groom contacts the BSPMaster, and reports task statuses by means of periodical piggybacks with BSPMaster.
- Each Groom is designed to run with HDFS (Hadoop file system) or other distributed storages.

Zookeeper

- Is used to manage the efficient barrier synchronization of the BSPPeers.
- It is also used in the area of a fault tolerance system.

BSP Examples

- Pi calculation
- K-means Clustering
- Sparse Matrix-Vector multiplication
- Parallel Support Vector Machine

Apache Hama

- Features
  - BSP API
  - M/R like I/O API
  - Graph API
  - Job management / monitoring
  - Checkpoint recovery
  - Local & (Pseudo) Distributed run modes
  - Pluggable message transfer architecture
  - YARN supported
  - Running in Apache Whirr

Apache Hama BSP API

- Public abstract class BSP<K1, V1, K2, V2, M extends Writable> …
  - K1, V1 are key, values for inputs.
  - K2, V2 are key, values for outputs.
  - M are they type of messages used for task communication.

Apache Hama BSP API

- public void bsp(BSPPeer<K1, V1, K2, V2, M> peer) throws …
- public void setup(BSPPeer<K1, V1, K2, V2, M> peer) throws …
- public void cleanup(BSPPeer<K1, V1, K2, V2, M> peer) throws …

Getting Started!

Requirements

Current Hama requires JRE 1.6 or higher and ssh to be set up between nodes in the cluster

- hadoop-0.20.2 (non-secure version), since Hama 0.5.0 we are using hadoop 0.1.X
- Sun Java JDK 1.6.X or higher version

Downloads

You can download Hadoop here:

- http://www.apache.org/dyn/closer.cgi/hadoop/core/

You can download Hama here:

- http://www.apache.org/dyn/closer.cgi/hama

Build Latest Version From Source

If you are going to use the latest (unreleased) version, you can check out TRUNK and build it with maven 3 with the following commands:

```
% svn co https://svn.apache.org/repos/asf/hama/trunk hama-trunk

% cd hama-trunk

% mvn clean install -Phadoop1 -Dhadoop.version=1.x.x -U
```

or

```
% mvn clean install -Phadoop2 -Dhadoop.version=2.x.x -U
```

Hadoop Installation

Required Software:

- Java (which you should already have at this point).

- ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.

Hadoop Installation

If your cluster does not have the requisite software you will need to install it.

$ sudo apt-get install ssh

$ sudo apt-get install rsync

Hadoop Installation

Unpack the downloaded Hadoop distribution. In the distribution,
edit the file etc/hadoop/hadoop-env.sh to define some parameters as follows:

Set to the root of your Java installation
export JAVA_HOME=/usr/java/latest

Assuming your installation directory is /usr/local/hadoop
export HADOOP_PREFIX=/usr/local/hadoop

Try the following command:
$ bin/hadoop

This will display the usage documentation for the Hadoop script.

Now you are ready to start your Hadoop cluster in one of the three supported modes:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

Hadoop Installation

Local (Standalone) Mode:

- By default, Hadoop is configured to run in a non-distributed mode, as a single Java process.

Pseudo-Distributed Mode:

- Hadoop can also run on a single-node where each Hadoop daemon runs in a separate Java process.

Fully-Distributed Mode:

- Is for non-trivial clusters ranging from a few nodes to extremely large clusters with thousands of nodes. It is more complicated to configure and outside the scope of this presentation. If you are interested or have more questions on this subject please visit

- http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html

Hama Installation

Untar the files to your destination of choice:

- tar -xzf hama-0.x.0.tar.gz

Don’t forget to chown the directory as the same user you configured Hadoop in the previous step.

Startup Script

The $HAMA_HOME/bin directory contains some script used to start up the Hama daemons.

- start-bspd.sh - starts all Hama daemons, the BSPMaster, GroomServer and ZooKeeper

Make sure you start Hama with the same user which is configured for Hadoop.

Configuration Files

The $HAMA_HOME/conf directory has some config files for Hama.

- hama-env.sh - Contains environment variables. The only variable you should need to change in this file is JAVA_HOME, which specifies the path to the Java 1.6.x installation used by Hama.
- groomservers - Lists the hosts, where the GroomServer daemons will run. By default this contains the single entry localhost.
- hama-default.xml - Contains generic default settings for Hama. DO NOT MODIFY THIS FILE!!!
- hama-site.xml - Contains site specific settings for Hama daemons and BSP jobs. This file is empty by default.

Setting Up Hama

Modes

Just like Hadoop, there are three modes:

- Local Mode - This is the default mode
- Pseudo Distributed Mode - This mode is when you just have a single server and want to launch all the daemon processes.
- Distributed Mode - This mode is just like the previous mode but you have multiple machines, which are mapped in the groomservers file.

Settings

BSPMaster and Zookeeper settings

- Figure out where to run your HDFS Namenode and BSPMaster. Set the variable bsp.master.address to the BSPMaster's intended host:port. Set the variable fs.defaultFS to the HDFS Namenode's intended host:port.

Example hama-site.xml File

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>bsp.master.address</name>
    <value>host1.mydomain.com:40000</value>
    <description>The address of the bsp master server. Either the
    literal string "local" or a host:port for distributed mode
    </description>
  </property>

  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://host1.mydomain.com:9000/</value>
    <description>
      The name of the default file system. Either the literal string
      "local" or a host:port for HDFS.
    </description>
  </property>

  <property>
    <name>hama.zookeeper.quorum</name>
    <value>host1.mydomain.com,host2.mydomain.com</value>
    <description>Comma separated list of servers in the ZooKeeper Quorum.
    For example, "host1.mydomain.com,host2.mydomain.com,host3.mydomain.com".
    By default this is set to localhost for local and pseudo-distributed modes
    of operation. For a fully-distributed setup, this should be set to a full
    list of ZooKeeper quorum servers. If HAMA_MANAGES_ZK is set in hama-env.sh
    this is the list of servers which we will start/stop zookeeper on.
    </description>
  </property>
</configuration>
```

Starting and Stopping A Hama Cluster

You can skip this step if you are in Local Mode

% $HAMA_HOME/bin/start-bspd.sh

- This will startup a BSPMaster, GroomServers and Zookeeper on your machine.

% $HAMA_HOME/bin/stop-bspd.sh

- This will stop all the daemons running on your cluster.

Run Some Examples

Run The Command:

% $HAMA_HOME/bin/hama jar hama-examples-0.x.0.jar

It will then offer you some examples to choose from.

Hama Web Interface

- The web UI provides information about BSP job statistics of the Hama cluster, running/ completed/failed jobs. By default, it is available at http://localhost:40013

A Few Examples!

BFS on Graph Example

```
public void expand(Map<Vertex, Integer> input, Map<Vertex, Integer> nextQueue) {
        for (Map.Entry<Vertex, Integer> e : input.entrySet()) {
                Vertex vertex = e.getKey();
                if (vertex.isVisited() == true) {
                        continue;
                } else {
                        vertex.visit();
                }
                //Put vertices adjacent to current vertex into nextQueue with increment distance
                for (Integer i : vertex.getAdjacent()) {
                        if (needToVisit(i, vertex, input.get(e.getKey()))) {
                                nextQueue.put(getVertexByKey(i), e.getValue() + 1);
                        }
                }
        }
}
```

```
public void process() {
    currentQueue = new HashMap<Vertex, Integer>();
    nextQueue = new HashMap<Integer, Integer>();
    //Initially put a start vertex in the current queue
    currentQueue.put(startVertex, 0);
    while (true) {
        expand(currentQueue, nextQueue);
        //The peer.sync method can determine if certain vertex resides on
        //local disk according to vertex id.
        peer.sync(nextQueue); //Sync nextQueue with other peers
        //At this step, the peer communicates vertex ID's corresponding to
        //vertices that reside on remote peers
        if (peer.state == TERMINATE)
            break;
        //Convert nextQueue that contains vertex ID's into currentQueue queue that contains vertices
        currentQueue = convertQueue(nextQueue);
        nextQueue = new HashMap<Vertex, Integer>();
    }
}
```

PI Estimator Example

- Each task executes locally its portion of the loop a number of times.

```
iterations = 10000
  circle_count = 0

do j = 1,iterations
generate 2 random numbers between 0 and 1
xcoordinate = random1
ycoordinate = random2
if (xcoordinate, ycoordinate) inside circle
then circle_count = circle_count + 1
end do

PI = 4.0*circle_count/iterations
```

Pi Estimator Example

- One task acts as master and collects the results through BSP communication interface.

```
PI = pi_sum / n_processes
```

Pi Estimator Example

- Each process computes the value of Pi locally.
- Sends it to the master task using send() function.
- Master task receives the message using sync() function.
- The final calculation for the average value of sum of Pi values from each peer as follows.

```
@Override
    public void bsp(
        BSPPeer<NullWritable, NullWritable, Text, DoubleWritable> peer)
        throws IOException, SyncException, InterruptedException {

      int in = 0, out = 0;
      for (int i = 0; i < iterations; i++) {
        double x = 2.0 * Math.random() - 1.0, y = 2.0 * Math.random() - 1.0;
        if ((Math.sqrt(x * x + y * y) < 1.0)) {
          in++;
        } else {
          out++;
        }
      }

      double data = 4.0 * (double) in / (double) iterations;
      DoubleMessage estimate = new DoubleMessage(peer.getPeerName(), data);

      peer.send(masterTask, estimate);
      peer.sync();
    }

    @Override
    public void setup(
        BSPPeer<NullWritable, NullWritable, Text, DoubleWritable> peer)
        throws IOException {
      // Choose one as a master
      this.masterTask = peer.getPeerName(peer.getNumPeers() / 2);
    }

    public void cleanup(
        BSPPeer<NullWritable, NullWritable, Text, DoubleWritable> peer)
        throws IOException {
      if (peer.getPeerName().equals(masterTask)) {
        double pi = 0.0;
        int numPeers = peer.getNumCurrentMessages();
        DoubleMessage received;
        while ((received = (DoubleMessage) peer.getCurrentMessage()) != null) {
          pi += received.getData();
        }

        pi = pi / numPeers;
        peer
            .write(new Text("Estimated value of PI is"), new DoubleWritable(pi));
      }
    }
}
```

References

- http://hama.apache.org
- http://hadoop.apache.org
- http://www.slideshare.net/teofili/machine-learning-with-apache-hama
- http://en.wikipedia.org/wiki/Bulk_synchronous_parallel
- http://www.cs.rutgers.edu/~pxk/417/exam/study-guide-final.html
- http://wiki.apache.org/hama/
- http://www.slideshare.net/udanax/apache-hama-an-introduction-tobulk-synchronization-parallel-on-hadoop-2699426