Hadoop Setup

In order to use HadoopUnit (described in Sect. 3.3.3), a Hadoop cluster needs to be setup. This cluster can be setup manually with physical machines in a local environment, or in the cloud. Creating a Hadoop cluster in a local setting can be time consuming since the necessary software needs to be installed on all the machines manually. The benefit of configuring a Hadoop cluster in the cloud is that the process is automated. One machine is customized with all the necessary software and this customized machine image can be used to launch a cluster of N nodes.

Amazon AWS, mentioned in Sect. 1.2.2 is an example of a cloud environment where HadoopUnit can be setup to execute test cases in the cloud. One tangible benefit to using Amazon as the cloud provider is that it has support for Hadoop. Hadoop also provides pre-built scripts to launch a cluster on Amazon EC2 specifically.

The next two sections describe how a Hadoop cluster is setup in a local environment and in the cloud on Amazon EC2 to conduct the case studies described in Chap. 4.

1 Cluster

A local test execution environment was setup in a Hadoop cluster of 13 machines to measure its performance. The cluster consisted of 13 virtual machines hosted on one physical server (the server used for the single machine described in Chap. 4). The master node does not usually function as the slave node (other than with installations that has only 1 or 2 machines). Therefore, this cluster offered 12 nodes of actual computation; the 13th node did not participate in the concurrent execution with HadoopUnit. The Hadoop cluster was created manually; no automated scripts were used for the setup.

Hadoop: A working installation of Java version 1.6 was used. Java needed to be installed on all 13 machines. One machine was randomly selected to be the master and the rest to be slaves. The master requires SSH access to manage its slave nodes. Therefore, SSH needs to be configured with shared keys (public/ private key) on all the nodes so that they can communicate with the master via a password-less SSH login. This was achieved by adding the public key of the master to all the slave nodes (on.ssh/authorized keys file).

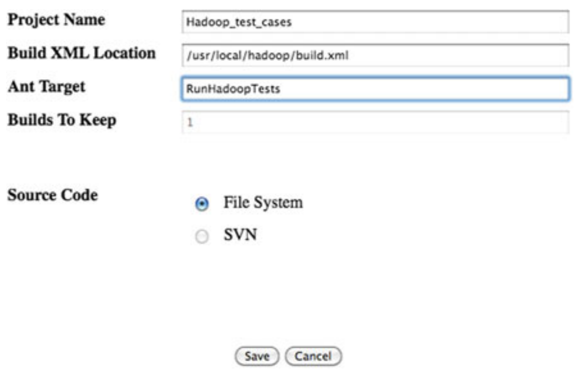
Hadoop was installed manually and the environment variables such as JAVA and HADOOP_HOME were set on all the machines. The master and slaves were listed in the configuration file. Other configurations, such as the directory where Hadoop will store its data files, the network ports it listens to, and so on were also made.

Environment variables such as JAVA, HADOOP_HOME, HADOOP_DIR, and HADOOP_SLAVES were set in the configuration files for all the machines. These configurations are necessary for Hadoop cluster setup and the details can be found in Hadoop documentation [53, 132].

HDFS: Once configured, HDFS needs to be formatted. It is necessary to do this for the first time a Hadoop cluster is setup by running the command */bin/hadoop namenode -format*. After formatting the namenode, HDFS daemons were started by running the command */bin/start-all.sh* on the namenode (master). This starts up HDFS with the namenode on the master and datanodes on the slave machines listed in the conf/slaves file. This will also start jobtracker running on the master and tasktrackers running on the slaves. At this point, a Hadoop cluster is setup where jobs can be distributed among multiple slave machines.

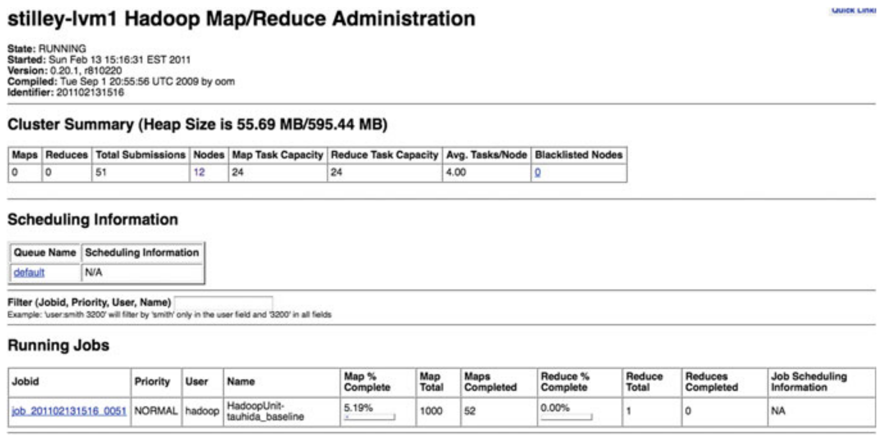
HadoopUnit: Once the Hadoop cluster is setup, HadoopUnit is installed on the master node. Jobs can be submitted and monitored using the command line or the Web UI [16]. Fig. 1 shows the Web UI to submit test cases to be executed concurrently. The information submitted through this Web UI include the project name, the location of a build file, and the target in the build file that needs to be executed. For example, in Fig. 1 the name of the project given by the user is hadoop_test_cases indicating that the test cases for Hadoop will be executed using HadoopUnit. The location of the build file on the master node, and the target of the build file called RunHadoopTests. This target contains the location where the test cases and other libraries that the test cases need.

Figure 2 shows the master node of the cluster and progress of the job. It shows that there are 12 nodes with a maximum capacity of 24 maps. Within Hadoop, a map task is a process run on one of its nodes; therefore the total number of concurrent processes would be equal to the number of map tasks per node multiplied by the number of nodes. Although each node is assigned the same number of map tasks to execute concurrently, it does not mean that each node will run the same amount of map tasks all the time. It may run up to the number of map task assigned. The current running job is also shown in this figure.



Project Name	Hadoop_test_cases
Build XML Location	/usr/local/hadoop/build.xml
Ant Target	RunHadoopTests
Builds To Keep	1
Source Code	[✓ File System] SVN

Fig. 1 HadoopUnit Web UI to submit a job



stilley-lvm1 Hadoop Map/Reduce Administration

State: RUNNING
Started: Sun Feb 13 15:16:31 EST 2011
Version: 0.20.1, r810220
Compiled: Tue Sep 1 20:55:56 UTC 2009 by oom
Identifier: 201102131516

Cluster Summary (Heap Size is 55.69 MB/595.44 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	51	12	24	24	4.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201102131516_0051	NORMAL	hadoop	HadoopUnit-tauhida_baseline	5.19%	1000	52	0.00%	1	0	NA

Fig. 2 Web UI to view the progress of jobs

2 Cloud

A Hadoop cluster was setup on EC2 to use HadoopUnit in the cloud. In order to create a Hadoop cluster on EC2, a custom image needs to be created, configured, and the custom image can be used to launch the Hadoop cluster. The configuration scripts are provided as part of the Hadoop distribution. They were modified here to meet our individual requirements.

1. Create a custom image
2. Launch a cluster of N nodes configured with custom image

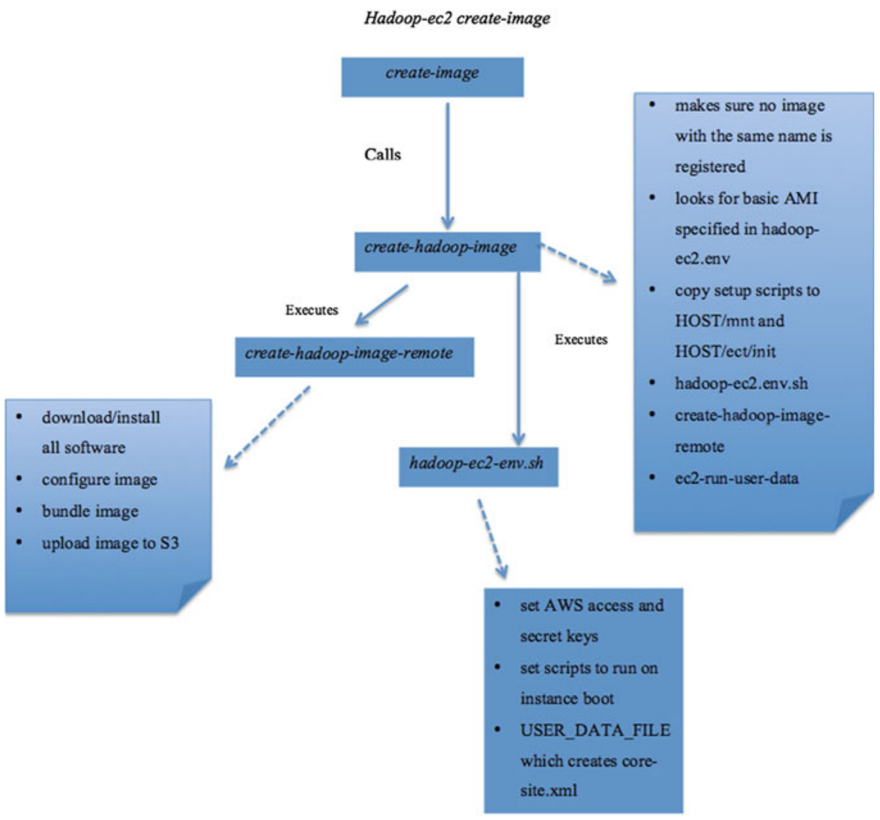

Hadoop-ec2 create-image
create-image
Calls
create-hadoop-image
Executes
create-hadoop-image-remote
Executes
hadoop-ec2-env.sh
makes sure no image with the same name is registered
looks for basic AMI specified in hadoop-ec2.env
copy setup scripts to HOST/mnt and HOST/ect/init
hadoop-ec2.env.sh
create-hadoop-image-remote
ec2-run-user-data
download/install all software
configure image
bundle image
upload image to S3
set AWS access and secret keys
set scripts to run on instance boot
USER_DATA_FILE which creates core-site.xml

Fig. 3 Steps to creating a custom image on EC2

2.1 Create A Custom Image

In order to setup HadoopUnit in the cloud, Amazon's storage service S3 and compute service EC2 were used. Amazon also provides a newer Web service called Elastic MapReduce, but it was found that it supports Hadoop version 18 and HadoopUnit was developed using the Hadoop version 20 API. Therefore, a custom image needed to be built with Hadoop 20. There are also benefits to using a custom image for the flexibility it provides. The custom image could be built using any software and dependencies necessary for testing purposes, saved and used to launch as many machines as needed at a later time.

Hadoop provides a set of scripts that can be used to create EC2 custom images. These scripts are located in hadoop/src/contrib/ec2/bin folder. The main scripts that are needed include:

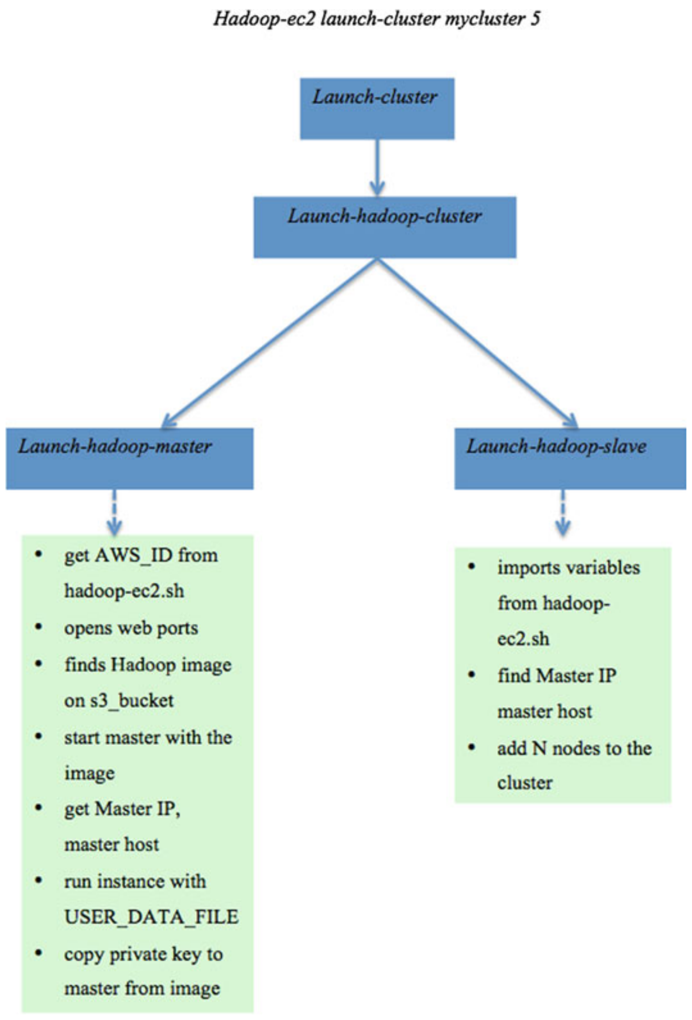
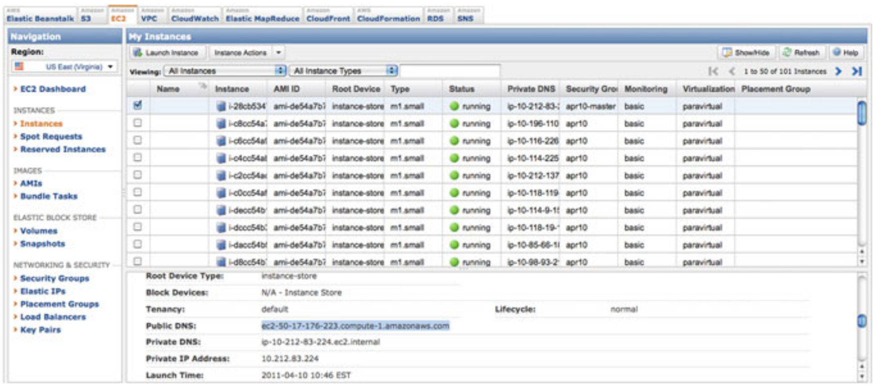

Hadoop-ec2 launch-cluster mycluster 5
Launch-cluster
Launch-hadoop-cluster
Launch-hadoop-master
Launch-hadoop-slave
get AWS_ID from hadoop-ec2.sh
opens web ports
finds Hadoop image on s3_bucket
start master with the image
get Master IP, master host
run instance with USER_DATA_FILE
copy private key to master from image
imports variables from hadoop-ec2.sh
find Master IP master host
add N nodes to the cluster

Fig. 4 Steps to launching a Hadoop cluster in EC2

- hadoop-ec2-env.sh: This script is used to set information such as Amazon access key, secret key, S3 bucket names, base machine image to use, and other environment variables.
- create-hadoop-image-remote: This script allows one to specify the software and dependencies that are needed to create an image. For example, for this research, software such as Java, Ant, Hadoop 0.20.0, HadoopUnit, GCC, MPFR, GMP, Dejagnu, Tcl, Expect (and many others) were pre-installed. For simplicity, all the necessary software were uploaded to S3 and transferred to the image from S3. This script was also used to configure the image. For example,



	Name	Instance	AMI ID	Root Device	Type	Status	Private DNS	Security Gro	Monitoring	Virtualization	Placement Group
✓		i-28cb534	ami-de54a7b7	instance-store	m1.small	running	ip-10-212-83-	apr10-master	basic	paravirtual	
☐		i-c8cc54a	ami-de54a7b7	instance-store	m1.small	running	ip-10-196-110	apr10	basic	paravirtual	
☐		i-c6cc54af	ami-de54a7b7	instance-store	m1.small	running	ip-10-116-226	apr10	basic	paravirtual	
☐		i-c4cc54al	ami-de54a7b7	instance-store	m1.small	running	ip-10-114-225	apr10	basic	paravirtual	
☐		i-c2cc54a	ami-de54a7b7	instance-store	m1.small	running	ip-10-212-137	apr10	basic	paravirtual	
☐		i-c0cc54af	ami-de54a7b7	instance-store	m1.small	running	ip-10-118-119	apr10	basic	paravirtual	
☐		i-decc54b	ami-de54a7b7	instance-store	m1.small	running	ip-10-114-9-1	apr10	basic	paravirtual	
☐		i-dccc54b	ami-de54a7b7	instance-store	m1.small	running	ip-10-118-19-	apr10	basic	paravirtual	
☐		i-dacc54b	ami-de54a7b7	instance-store	m1.small	running	ip-10-85-66-1	apr10	basic	paravirtual	
☐		i-d8cc54b	ami-de54a7b7	instance-store	m1.small	running	ip-10-98-93-2	apr10	basic	paravirtual	

Root Device Type:	instance-store		
Block Devices:	N/A - Instance Store		
Tenancy:	default	Lifecycle:	normal
Public DNS:	ec2-50-17-176-223.compute-1.amazonaws.com		
Private DNS:	ip-10-212-83-224.ec2.internal		
Private IP Address:	10.212.83.224		
Launch Time:	2011-04-10 10:46 EST		

Fig. 5 AWS console showing machines launched

configurations of Hadoop, HadoopUnit, setting environment variables were all done using this script. Once the image is created, it is bundled and uploaded to S3. When a cluster is launched, the launch-cluster scripts looks at S3 for the image and use it to launch the cluster.

- hadoop-ec2-init-remote.sh: This script is used to setup the HDFS and MapReduce environment for the cluster.

Once the scripts were setup, to create the image the hadoop-ec2 create-image command was executed from hadoop/src/contrib/ec2/bin folder. Fig. 3 shows how the different commands work together to create a custom image.

2.2 *Launch an* n-*Node Cluster with Custom Image*

Once the custom image is created, the next step is to use this image to launch a Hadoop cluster of N nodes. This can be done using the command *hadoop-ec2 launch-cluster mycluster 5,* where launch-cluster is a script that calls *launch-hadoop-cluster,* which in turn calls launch-hadoop-master and launch-hadoop slave scripts. The *launch-hadoop-master* script does the following:

- Gets AWS ID from Hadoop-env.sh, which is located in the hadoop/conf folder and makes sure that the correct AWS ID is used
- Opens Web ports to monitor jobs using Web UI
- Finds Hadoop image on S3
- Starts an instance using the image and starts master (runs namenode deamon)
- Gets master IP and master host names
- Copies private key to the master

The *launch-hadoop-slave* script launches the slaves. This script imports variables to set the environment variables from the hadoop-ec2-env.sh file. It then adds N nodes to the cluster. Slaves are started after the master, and are told its address by sending a modified copy of hadoop-ec2-init-remote file that sets the MASTER_HOST variable. A node knows if it is the master or not by inspecting the security group name. If it is the master then it retrieves its address using instance data. Fig. 4 illustrates the steps to launching a cluster in EC2.

Once the Hadoop cluster is launched on EC2, HadoopUnit can be installed and used to execute test cases concurrently. Fig. 5 shows the AWS console where user can see the launched machines. It can be used to view progress of the job. Once the machines are launched, the IP of the master node can be acquired to log on to the master to run jobs using HadoopUnit.

Transforming GCC Test Cases

This section describes how GCC test cases were transformed to JUnitWrapper. A wrapper around the GCC test cases was used so they structurally look like JUnit test cases. Wrapping was used for the GCC test cases to keep the core functionality of GCC intact but have a way to work with HadoopUnit's Web interface.

A list of all the test cases that needs a wrapper was obtained. This was extracted using a shell script (Sect. 4.2.3). This list of test cases was saved in a text file, which becomes the input to the JUnitWrapper program. Each of the input command lines consists of a "expect" file (similar to a Test Class in JUnit) and a test file (similar to an unit test).

The JUnitWrapper consists of two main classes named ToCreate() and ToParent(). The user executes ToCreate with two parameters: an input file that contains a list if the test cases (e.g. commands.txt), and secondly a jar extension output filename (e.g. myGCCTests.jar). Then, the ToCreate class works as follows:

- It creates a Java JUnit source file by:

  1. Initiating a set of strings to hold all the JUnit class file names

  - Reading the input file with the commands (command.txt) line by lineReading the input file with the commands (command.txt) line by line
  - When an expect file that doesn't exist in the set of class file names is found, it creates a JUnit java source class with the heading that determines the libraries and the class name. In addition, it adds the name of the class to the set that holds the JUnit class file names.
  - It adds the GCC unit test, now wrapped as a JUnit test, to the java source file.

- When an expect file that exists in the set of class file names is found, it adds a footer that closes the JUnit java source class. In addition, it removes the class file name from the set.

2. It copies the ToParent java source file to a temp output folder that holds all the wrapped java source files
3. It copies an ant build.xml to the temp output folder in order to allow compilation of the java JUnit source files
4. It then creates an shell script in the temp output folder that runs “ant” in order to compile the java files into class files
5. It executes the shell scripts. Thus, class files are created from the wrapped JUnit java files
6. It creates the jar file containing the JUnit Wrapped Source files, the JUnit Wrapped Class files, and the ToParent java and class files. The name of this jar file is the name specified at the execution of the program.

- In order to use the JUnit wrapper, users need to:

1. Create an input file with a list of command lines to be executed in the format <<CLAS_NAME>> @ COMMAND
2. Place the file in the same folder as the ToCreate.class file
3. Make sure that in the folder where ToCreate.class is located, there is a folder called “src” with the ToParent.java, and buildWrapper.xml
4. Make sure the in the folder where ToCreate.class is located, there are no more files other than ToCreate.class, the src folder, and the input file
5. From the command line, execute “java ToCreate inputFile.txt my_output.jar”

The output of the program is a jar file with all the wrapper classes to be executed on the cloud. For the jar file, there will be one wrapper class for each “expect” file.

Once this is done, GCC test cases wrapped around JUnit (or with the help of JUnit framework) can be executed as follows:

1. A build file is created that is used for the input/outut of the test cases. It looks as follows (Fig. 1):

```
<?xmlversion="1.0"?>
<project name="gcc_testcases" default="RunGcc"
basedir=".">
<property name="testclasses.dir" value="${basedir}/gcc-
4.4.1/JunitWrapper/classes"/>
<property name="lib.dir" value="/usr/local/hadoop/lib"/>
<property name="test.excludes" value="ToParent.class"/>
<path id="hu.test.path">
<pathelement location="${testclasses.dir}/tree.jar"/>
<pathelement location="${lib.dir}/junit-4.5.jar"/>
<pathelement path="${lib.dir}/ant.jar"/>
<pathelement path="${lib.dir}/ant-launcher.jar"/>
<pathelement path="${lib.dir}/ant-junit.jar"/>
</path>
<target name="RunGcc" description="Run main tests">
<taskdef name="hadoopUnit"
classname="org.apache.hadoop.ant.unit.HadoopUnitTask"/>
<hadoopUnit classpathref="hu.test.path" taskspermap="1"
parallel="false">
<batchtest>
<fileset dir="${testclasses.dir}"
excludes="${test.excludes}"/>
</batchtest>
</hadoopUnit>
</target>
</project>
```

Fig. 1 Build file for GCC JUnit test cases

2. The WebUI can be used to submit the job and run the test cases.