HowToUseSystemTestFramework

System tests development How To

This document described how to develop cluster based system tests with new Hadoop cluster test framework (code name Herriot). For more information about Herriot visit HADOOP-6332

Here you can find up-to date javadocs of the framework's APIs.

Definitions

The following definitions will be used through the guide:

- **Test client** a computer/source location where the execution of tests is initiated
- **Daemon proxy** an RPC class which provide access to a remote Hadoop's daemon's (NN, DN, JT, TT) APIs
- **Cluster proxy** Herriot class which combines and provides a convenient API to control Hadoop cluster from Herriot **test client**
- **Herriot library** the combination of above APIs residing on a **test client**

Test development environment

To develop tests for Herriot you don't need any extra tools. Herriot is embedded into Hadoop source code base. The APIs exposed for test development are static and present in the form of interfaces. Test framework specific classes such as *cluster proxy* and so on are available in the form of Java classes. First, clone a git repository and check out latest Hadoop branch:

```
git clone git://github.com/apache/hadoop-hdfs.git hdfs
git checkout -t -b trunk origin/trunk
```

For Common and Mapreduce place adjust above command accordingly and change the name of the branch in case in you need a different one.

All you need is to make sure that the following source directories are included to the project's definition of your favorite IDE:

```
src/test/system/aop
src/test/system/java
src/test/system/test
```

The first two are needed only for test framework development. So if your purpose is Herriot test development you can limit your configuration to the latter location only.

Tests structure

Herriot tests make use of the JUnit4 framework (they may also use !TestNG if this framework is exposed to Hadoop). JUnit fixtures are used in Herriot tests such as `@Before`, `@After` and so on. For our immediate purpose Herriot tests are JUnit tests. Therefore if you know how to develop JUnit tests you are good to go.

In the current environment tests should be placed under

```
src/
  test/
    system/
      test/
        [org.apache.hadoop.hdfs|org.apache.hadoop.mapred]
```

Framework related classes belong to `org.apache.hadoop.test.system` for the shared code and/or to `org.apache.hadoop.hdfs.test.system`, `org.apache.hadoop.mapreduce.test.system` for HDFS and MR specific parts, respectively.

Examples

Let's take a look at the real test example available from `src/test/system/test/org/apache/hadoop/mapred/TestCluster.java`. As always your best source of information and knowledge about any software system is its source code 🙂

- Let's start with `@BeforeClass` fixture creating an instance of *cluster proxy* (in this case for a MapReduce cluster) which provides access to MapReduce daemons (the Job Tracker [JT] and Task Trackers [TTs]). The second call creates all needed *daemon proxies* and makes them available through *Herriot library* APIs. As part of this setup Herriot will guarantee that the test environment is clean and all internal states are reset. Also, a number of exceptions that arise in the daemon logs will be saved. This is particularly useful as it allows us to disregard exceptions raised in the log files before a Herriot test has been started. `@BeforeClass` will guarantee that only one instance of *cluster proxy* is created (for this is an expensive operation) for use in all test cases defined in the test class.

```
@BeforeClass
public static void before() throws Exception {
  cluster = MRCluster.createCluster(new Configuration());
  cluster.setUp();
}
```

- It is easy to submit and verify a MapReduce job:

```
@Test
public void testJobSubmission() throws Exception {
  Configuration conf = new Configuration(cluster.getConf());
  SleepJob job = new SleepJob();
  job.setConf(conf);
  conf = job.setupJobConf(1, 1, 100, 100, 100, 100);
  RunningJob rJob = cluster.getJTClient().submitAndVerifyJob(conf);
  cluster.getJTClient().verifyJobHistory(rJob.getID());
}
```

The new JT's API call {{submitAndVerifyJob(Configuration conf) }} will check if the job has been completed successfully by looking into the job details (e.g. number of maps and reducers), monitoring their progress and success of job execution, as well as proper cleanup. If some of the conditions aren't met proper exceptions will be raised.

- The following example demonstrates how to modify a cluster's configuration and restart the daemons with it. At the end the original the cluster is restarted with its original configuration.

```
@Test
public void testPushConfig() throws Exception {
  final String DUMMY_CONFIG_STRING = "mapred.newdummy.conf";
  String confFile = "mapred-site.xml";
  Hashtable<String,Long> prop = new Hashtable<String,Long>();
  prop.put(DUMMY_CONFIG_STRING, 1L);
  Configuration daemonConf =  cluster.getJTClient().getProxy().getDaemonConf();
  Assert.assertTrue("Dummy varialble is expected to be null before restart.",
      daemonConf.get(DUMMY_CONFIG_STRING) == null);
  cluster.restartClusterWithNewConfig(prop, confFile);
  Configuration newconf = cluster.getJTClient().getProxy().getDaemonConf();
  Assert.assertTrue("Extra varialble is expected to be set",
      newconf.get(DUMMY_CONFIG_STRING).equals("1"));
  cluster.restart();
  daemonConf = cluster.getJTClient().getProxy().getDaemonConf();
  Assert.assertTrue("Dummy variable is expected to be null after restart.",
      daemonConf.get(DUMMY_CONFIG_STRING) == null);
}
```

The above example also works for Hadoop clusters where DFS and MR are started under different user accounts. For this to happen multi-user support needs to be enabled in the Herriot's configuration file (see below).

- Apparently, it is nice to clean up after you when everything is done:

```
@AfterClass
public static void after() throws Exception {
  cluster.tearDown();
}
```

Tests execution environment

For execution of the tests the test client needs to have:

- access to a cluster which runs instrumented build of Hadoop
- available copies of configuration files from a deployed cluster (located under `$HADOOP_CONF_DIR` in this example).

No Hadoop binaries are required on the machine where you run tests (there is a special case of single-node cluster where you need to build instrumented Hadoop before running the tests; see below). Herriot tests are executed directly from a source code tree by issuing the following ant command:

```
  ant test-system -Dhadoop.conf.dir.deployed=${HADOOP_CONF_DIR}
```

To run just a single test use usual property `-Dtestcase=testname`

Once the test run is complete the results and logs can be found under `build-fi/system/test` directory.

Normally, **test client** is expected to be a cluster's gateway. However, it should possible to run tests from one's deployment machine, laptop, or from a cluster node.

http://people.apache.org/%7Ecos/images/Herriot%2Ddeployment.jpg

The following software tools should be available on a

test client:

- Ant (version 1.7+)
- Java6

In the future a means to run Herriot tests from a jar file may also be provided.

Some of the tests might throw lzcodec exceptions. To address this `build.xml` will accept new system property `lib.file.path` that needs to be set before testa are run.

```
  ant test-system -Dhadoop.conf.dir.deployed=${HADOOP_CONF_DIR} -Dlib.file.path=${HADOOP_HOME}/lib/native/Linux-
i386-32
```

The feature is pending MAPREDUCE-1790

`hadoop-common/src/test/system/scripts/*` has to have executable permissions, otherwise some of the mapreduce test cases will fail.

Settings specific for security environment

To make Herriot protocols trusted in a secure Hadoop environment `hadoop-policy-system-test.xml` must be included in `hadoop-policy.xml`

/* TODO: not yet implemented Execution specific file is `proxyusers` that needs to be created under `${HADOOP_CONF_DIR}`. This file has to contain the username of the users who can impersonate others. This features only makes sense if security is enabled. To successfully run TestDecommisioning hadoop-policy.xml's property `security.admin.operations.protocol.acl` should include the test user's group in the value; mapred-site.xml's property `mapreduce.cluster.administrators` should include test user's group.

*/

Configuration files

When the Herriot **test client** is starting it is looking for a single configuration file `system-test.xml`. Internally, this file is supplied by an automated deployment process and should not be a concern for test developers. However, more information will be provided in the deployment section of this document.

Herriot cluster deployment procedure/requirements

Herriot configuration files are located in `src/test/system/config`

The framework uses its own custom interfaces for RPC control interaction with remote daemons. These are based on the standard Hadoop RPC mechanism. However, these are designed to not interfere with normal Hadoop traffic. On DFS side no additional configuration is needed - the framework will derive all needed information from the existing configuration files. However, to allow connectivity with TaskTrackers the following property needs to be added to `mapred-site.xml`:

```
  <property>
    <name>mapred.task.tracker.report.address</name>
    <value>0.0.0.0:50030</value>
    <final>true</final>
  </property>
```

This configures an extra RPC port thus enabling direct communication with TTs that isn't normally available.

The content of `system-test.xml` needs to be customized during installation according to the macros defined in the file.

Some modification is required to enable multi-user support on the *test client* side. Herriot distribution provides a special binary which allows setuid execution. The source code for this tool is located under `src/test/system/c++/runAs`. As an additional security guarantee the binary must have the $HADOOP_HOME environment variable defined properly at compile time. This reduces the risk of malicious use of the framework. To compile it use the following ant target

```
ant run-as -Drun-as.hadoop.home.dir=<HADOOP_HOME location>
```

As the final step, the binary has to be installed into the `test.system.hdrc.multi-user.binary.path` specified in `system-test.xml`. Configure setuid permissions with

```
chown root ${test.system.hdrc.multi-user.binary.path}/runAs
chmod 6511 ${test.system.hdrc.multi-user.binary.path}/runAs
```

Single-node cluster installation

It is convenient to be able to run a cluster on one's desktop and to be able to execute the same cluster tests in your own environment. It is possible. The following steps need to be taken

```
 % cd $WORKSPACE
 % ant binary-system
 % export HADOOP_HOME=$WORKSPACE/build-fi/systdoop-$version-SNAPHOT
 % export HADOOP_CONF_DIR=<location of your configs>
 % cd $HADOOP_HOME
 % chmod +x bin/*
 % ./bin/start-all.sh
 % cd $WORKSPACE
 % ant test-system -Dhadoop.conf.dir.deployed=$HADOOP_CONF_DIR
```

Edit `src/test/system/conf/system-test.xml` file and put it under `$HADOOP_CONF_DIR`