PigTestProposal

Pig Testing Proposal

Definitions

For the purposes of this document, the following terms are defined as:

- **unit test**: A test that verifies the functionality of an individual class, method, or interface. An example is a test that checks that `Tuple.append()` adds one field to the end of a Tuple.
- **end-to-end test**: A test that verifies behavior of Pig as a package. An example is a test that checks that `A = load foo; B = order A by $0` produces properly sorted data.
- **integration test**: A test that verifies Pig works with other members of the Hadoop family. An example is a test to check that `HBaseStorage` properly loads data from and stores data to HBase.
- **JUnit test**: A test run via JUnit, regardless of whether it is a unit test or an end-to-end test.
- **e2e harness**: The test harness recently added to Pig (see PIG-1899).

Introduction

Throughout its life, Pig has been tested in two different ways: JUnit tests and the e2e harness, which was originally used only for Yahoo's nightly tests. A significant portion of the JUnit tests are end-to-end tests or integration tests. As a result running them takes around 8 hours on a developer's machine. The e2e harness has recently been added to Apache Pig, but is not yet running as part of the nightly tests.

There are a number of issues with the current situation:

1. When developers do `ant test`, JUnit tests take 8 hours to run. This seriously discourages people from running them before submitting /committing patches. It wastes Apache build resources for automated builds that need to do a quick test (such as pre-commit). Also, since some of the JUnit tests take a long time it is almost guaranteed that a user will experience at least one test failure due to timeout during the test run.
2. JUnit is not a good test tool for end-to-end and integration tests, for several reasons:
    a. It requires users to write Java code. Ideally users would just need to write Pig Latin scripts to test and some kinds of verification script; not Java code.
    b. It is hard from Java to instantiate Hadoop clusters (beyond MiniCluster or local mode) or other related services (HCatalog, HBase, etc.) that tests may need to use.
    c. It is hard to generate and work with significant amounts of data. Tests end up being run over just a few records. This is not a reasonable test of Pig.
3. The e2e harness requires an existing cluster and a Postgres installation at this point.
4. There is no documentation for the e2e harness.
5. Yahoo is running hundreds of tests nightly using the e2e test harness, which benefits Pig in that many issues are caught. But other developers are not able to add to these tests or see the results to fix issues that arise.

Proposed Solution

The proposed solution has two key parts:

1. Use the e2e harness in nightly builds and developer testing for end-to-end and integration tests.
2. Change JUnit tests to be only unit tests; migrate all JUnit tests that are end-to-end and integration tests to the e2e harness.

This will produce a number of benefits:

1. Adding an end-to-end test will require no code. The user will only need to write a Pig Latin script to test the feature and indicate whether it should be tested against a previous version of Pig or a second Pig Latin script. When writing true unit tests for a given feature developers will still need to write Java, but this makes sense since they are testing a Java class.
2. Unit tests will run in a minute or so instead of 8 hours.
3. All Pig users and developers will have access to the rich set of tests used to verify Pig functionality.
4. End-to-end tests can run against larger data sets without requiring the user to hand generate kilobytes or megabytes of data and then eyeball the results to make sure they are good.

A number of changes need to be made to accomplish this goal:

1. Currently the e2e harness depends on there already existing a cluster to create data on and run tests against. It needs to work in the case where a cluster is not available by instantiating a pseudo-distributed cluster on the users box. Ideally it should also work with cloud solutions, such as Amazon's EC2 so that users who do not have the hardware can still run tests in a true distributed cluster.
2. Currently the e2e harness depends on there being a Postgres database to generate expected results from. Originally using a Java only no server database such as Hsqldb or Derby was investigated (see PIG-2154). However, none of these databases support enough SQL to fully test Pig. In particular, they do not support full outer joins. Early in Pig's development it was important to have a SQL database as a source of truth because Pig was implementing relational operators. Now that it has a full relational operator set it seems sufficient to instead allow tests to verify either against an older version of Pig or against an alternate Pig Latin script. For example, when new macro features are added they can be verified against a Pig Latin script that has the macro code inlined.
3. Some tests require special data. The current e2e harness requires that users specify data sets up front. It should be easy for a user to add a few lines of data to existing data sets so that tests that require special data can do so without modifying the deployment code that creates data sets and so that entire data sets are not required for individual tests.

4. Currently the e2e tests take about 8 hours to run on an existing cluster, and would take longer on a pseudo-distributed cluster running on the developer's local box. Since the tests are mostly small it is possible to parallelize them to take advantage of extra nodes in the cluster when available. This would allow nightly tests to be run in significantly less time. This is a long term goal and will not be realized immediately.

The end state then would be that doing `ant test` would run the unit tests plus a couple of simple e2e tests, similar to the `TestCommit` that is a JUnit test today. This should finish in a few minutes. `ant test-e2e` would run the end-to-end tests. Depending on properties passed to ant these could be run on a pseudo-distributed cluster on the local box, in local mode, on an existing cluster, or in a cluster that is created in a cloud such as EC2. The nightly builds could then go against a small cluster created as part of the nightly test script and run all of the unit and e2e tests.

Introduction to Pig e2e Harness

The test harness that Yahoo has been using to test Pig since the beginning has been added to Pig's source. It is a fairly simple test harness written in Perl. The concept is that it allows users to write a driver for testing particular packages. This driver must implement three methods: `runTest`, `generateBenchmark`, and `compare`. The driver runs these in order, and declares a test to have succeeded or failed based on the result of `compare`. If at any point one of these three functions fails the test is declared to have aborted. Global setup and cleanup methods are also available. The harness reports on the test results as they run, and at the end produces a summary of how many tests succeeded, failed, and aborted.

Tests are declared in a Perl hash. This hash must have a key `driver` that tells the harness what Perl module to use to run the tests and a key `groups`. The `groups` key must point to an array. Each element of the groups array is a hash that must have two keys: `name` and `tests`. `name` is the name of the group, and `tests` points to an array. The test array contains one or more tests. Each test is again a hash. It must have a tag `num`, which provides a number of this test. Beyond this the contents are up to the driver.

The test driver for Pig expects to find a `pig` key in the test hash which specifies the Pig Latin script to run and a `sql` key that specifies SQL to test it against. The Pig Latin is run against a cluster that is provided to the harness. The SQL is run against a Postgres instance. `compare` then uses `md5` and `sort` (for tests where order is important) to check that the results match. A simple example test looks like:

```
$cfg = {
    'driver' => 'Pig',

    'groups' => [
        {
        'name' => 'Checkin',
        'tests' => [
            {
            'num' => 1,
            'pig' => q\a = load ':INPATH:/singlefile/studenttab10k' as (name, age, gpa);
store a into ':OUTPATH:';\,
            'sql' => "select name, age, gpa from studenttab10k;",
            'floatpostprocess' => 1,
            'delimiter' => '    ',
            },
        }
    ],
};
```

`:INPATH:` and `:OUTPATH:` are variables replaced by the Pig driver as the test is run. Given this test, the driver will run the Pig Latin specified by the `pig` key in `runTest`. `generateBenchmark` will run the query specified in `sql`, and `compare` will use `md5` to test that the results of the two queries match.