

DART HOWTO

Brent Chun

This document describes how to use DART, a framework for distributed automated regression testing of large-scale network applications. The current DART implemenation targets Emulab.

Table of Contents

1. Preliminaries	2
2. Installing the Software.....	2
3. Writing a Distributed Test.....	2
4. Example: Query Correctness Test on PIER	2
5. Running a Distributed Test	5

1. Preliminaries

The current DART implementation targets Emulab. Thus, order to use DART, you will first need to obtain an account on Emulab. In the rest of this document, we assume that your Emulab username is the same as the username on the machine where you are running DART from.

DART is also currently implemented in Python and has been tested primarily on Redhat 7.3 using Python 2.2. If your platform differs from this, you may experience some Python versioning issues, although, should they occur, they are likely to require minor fixes.

2. Installing the Software

DART is distributed as a tarball. Deal with it! In particular, make sure to untar it at the top-level in your home directory.

3. Writing a Distributed Test

A DART test consists of several key pieces: (1) a set of files to install on each node, (2) an optional preexecution script, (3) scripts that run at specific times on subsets of nodes (e.g., a big "run" script that runs on all nodes), (4) a postexecution script and (5) a reset script (which cleans out a node and allows Emulab experiments to be reused). In addition, each test allow requires a unique name, the name of your Emulab project, a unique Emulab experiment name, and several other details. We'll work through an example below to make this clearer. Also, while this may sound like a lot of work, a significant fraction of the above is usually common to multiple tests. Thus, writing a new test often requires just writing the "diff".

4. Example: Query Correctness Test on PIER

```
<?xml version="1.0" ?>
<dart>

  <test>
    <name>sq.pier0032</name>
    <outdir>/tmp/sq.pier0032</outdir>
  </test>

  <topology>

    <project>planetlab</project>
    <experiment>pier0032</experiment>
    <nsfile>pier0032.ns</nsfile>
    <eipsfile>pier0032.eips</eipsfile>
    <iipsfile>pier0032.iips</iipsfile>

  </topology>

  <commonfiles>
    <dir>
      <src>/homes/bnc/pier/server</src>
```

```

    <dst>${DART_COMMON_DIR}/server</dst>
</dir>
<dir>
    <src>/homes/bnc/pier/client</src>
    <dst>${DART_COMMON_DIR}/client</dst>
</dir>
<dir>
    <src>/homes/bnc/pier/sensors</src>
    <dst>${DART_COMMON_DIR}/sensors</dst>
</dir>
<dir>
    <src>/homes/bnc/pier/scripts</src>
    <dst>${DART_COMMON_DIR}/scripts</dst>
</dir>
<file>
    <src type="remote">/proj/planetlab/tarfiles/edata.tar.gz</src>
    <dst>${DART_COMMON_DIR}/edata.tar.gz</dst>
</file>
<file>
    <src type="remote">/proj/planetlab/rpms/j2sdk-1_4_2_03-linux-i586.rpm</src>
    <dst>${DART_COMMON_DIR}/j2sdk-1_4_2_03-linux-i586.rpm</dst>
</file>
</commonfiles>

<preexecution>
    <script>${DART_COMMON_DIR}/scripts/preexecution</script>
</preexecution>

<execution duration="700">
    <nodegroup>
        <nodes>*</nodes>
        <cmd>${DART_COMMON_DIR}/scripts/startPier</cmd>
    </nodegroup>
    <nodegroup>
        <nodes>*</nodes>
        <cmd>${DART_COMMON_DIR}/scripts/startsensors</cmd>
    </nodegroup>

    <!-- Wait 120 seconds for PIER, 10 seconds between each test -->
    <nodegroup>
        <nodes>0</nodes>
        <cmd time="120">${DART_COMMON_DIR}/scripts/sq.selectall/runclient -b 1 1</cmd>
    </nodegroup>
    <nodegroup>
        <nodes>0</nodes>
        <cmd time="250">${DART_COMMON_DIR}/scripts/sq.selectall/runclient -b 4 1</cmd>
    </nodegroup>
    <nodegroup>
        <nodes>0</nodes>
        <cmd time="380">${DART_COMMON_DIR}/scripts/sq.selectall/runclient -b 16 1</cmd>
    </nodegroup>
    <nodegroup>
        <nodes>0</nodes>

```

```

        <cmd time="510">$DART_COMMON_DIR/scripts/sq.selectall/runclient -b 64 1</cmd>
    </nodegroup>

</execution>

<postexecution>
    <script>$DART_COMMON_DIR/scripts/sq.selectall/postexecution</script>
</postexecution>

<reset>
    <script>$DART_COMMON_DIR/scripts/reset</script>
</reset>

</dart>

```

The test section includes two things. First, a unique test name and second, a local directory to store test output from all nodes in the test. Under the output directory, node output will be stored under subdirectories named by virtual nodes numbers ranging from 0 to n - 1. Usually I just store all output in /tmp under /tmp/testname as in the example.

The topology section includes Emulab information and names of useful related files. The project is the name of the Emulab project you are part of (recall the Preliminaries section). The experiment and nsfile correspond to the specific Emulab topology the distributed test is to be run on. nsfile is an Emulab topology file generated by dart using the mktopologies command. (NOTE: add more on this) while experiment is the name of the experiment corresponding to that file. When running multiple tests that use the same topology, it's convenient to use the same experiment and nsfile names. This will ensure that the topology gets cached rather than reinstantiated each time. Lastly, the eipsfile and iipsfile are local files (that DART creates) that store the list of external IP addresses and internal IP addresses for the nodes in your Emulab experiment. These can be useful if you need/want to ssh into specific nodes to examine what is going on (e.g., when debugging a distributed test for the first time).

The commonfiles section specifies a set of local files/directories (dir for directories, file for files) to be transferred to each node. For each file/directory, src specifies a local path of the relevant data and dst specifies a remote path on Emulab nodes. Specifying type="remote" specifies that src should be taken relative to the destination node. This is useful if you are repeatedly transferring common files over the wide-area that are large and are better served off of Emulab's local NFS (which all nodes mount). For example, in the above, the JDK's src is a path to an NFS mounted directory on an Emulab node and dst is a local path on that Emulab node.

The preexecution section specifies a script to run on all nodes after all files/directories in commonfiles have been transferred over. In the preexecution script above, we install the JDK RPM and copy static data (to be queried) off of Emulab's NFS server. This is entirely application-specific and is optional (omit this section if you don't want it).

The execution section is where the main action takes place. It takes a required attribute called duration which specifies how long the test should run. Within the execution section, one or more nodegroup sections can be specified. Each nodegroup section specifies a command to run at a specified time on a specific subset of nodes. The time is specified in seconds since the test start. The nodes are specified by a comma separated node numbers (0 to n-1). Two conveniences are provided: a wildcard * which means all nodes and ranges of nodes (e.g., 0-3). In the PIER example, there are three things going on. First, we start the servers up by calling startPier. This starts up PIER. Second, we start the sensors that have data to be queried. Third, we issue a sequence of queries by a single client in succession. Note that after starting PIER, we wait 120 seconds before issuing the first such query. This is to give the DHT time to stabilize and to allow the multicast tree for query dissemination to form. In the test code itself (i.e., the client code), output is written to a well-known directory (/tmp/out). This output will then be collected by the master node (node 0) which we can then run a postexecution script against to check the results.

The postexecution section specifies a script to run on the master node (node 0) after everything in the execution section has completed. Note that, in contrast to the preexecution section, this runs only on the master. The idea here is that after the test completes, all the output data (from /tmp/out on each node) from each node will be collected on the master in a well-known directory (/tmp/allout on the master). We then run a postexecution script to compute the results of the test.

Finally, the reset script specifies how to reset a node so that an Emulab experiment can be reused across tests that use the same topology. This might kill all application processes and remove all application files for example.

5. Running a Distributed Test

Given a test file (or set of test files) such as the one in the previous section (e.g., suppose it was named sq.pier.0032.xml and we have another test file named mq.pier.0032.xml), we run it as follows:

```
bin/dart sq.pier0032.xml mq.pier0032.xml
```

Assuming we followed the output directory conventions described earlier, this command will instantiate the necessary Emulab experiments (reused cached ones if possible), set up distributed tests, run them, and collect the output in directories on your local machine.