Multi-tier Application Management with sams by Example

Nicolas Berthier
mail@nberth.space

November 3, 2016

This rather technical manual presents and exemplifies sams' management capabilities for multi-tier applications. A procedure provided to exercise one of the examples is also described in this manual.

1 Multi-tier Application Management with sams

1.1 Multi-tier Application Management

In terms of sams terminology, the resources involved in a basic *multi-tier application* are the *tiers*. Of course, other kinds of resources might need to be added when considering other layers of the managed system (*e.g.,* hypervisor management).

In this manual, we consider basic multi-tier applications organized as chains of *services* rendered by *tiers* (using the provided facilities for the management of other kinds of more "hierarchical" multi-tier applications would require slightly adapting these utilities).

If a tier T_1 requests services from a tier T_2, then T_1 is called the *source tier* of T_2, and T_2 is called the *tail tier* of T_1.

Example Deployment of a Basic 2-tier Application Figure 1 exemplifies the overall deployment structure of a Apache-Tomcat 2-tier application. In this picture, sams' management infrastructure comprises every agent server. The tick is a synchronous program encoding the decision logic about all management operations to be performed on the resources (as it happens, the tiers). The agent encompassing the tick is the decision code, and all other agents of the center process constitute the associated operating code.

T_a and T_t agents encapsulate the operating code responsible for handling the instructions related to the state management of their respective tiers. Besides, M_a and M_t are agents that multiplex measurements (*e.g.,* CPU loads) sent from remote sensors S_a, $\mathsf{S}_{\mathsf{t}_1}$ and $\mathsf{S}_{\mathsf{t}_2}$, and transmit summarized measurement scalars to the decision code. They are also intended to detect failures by tracking missing notifications from the sensors, and generate events in the form of notifications sent to the tick (or in some cases their

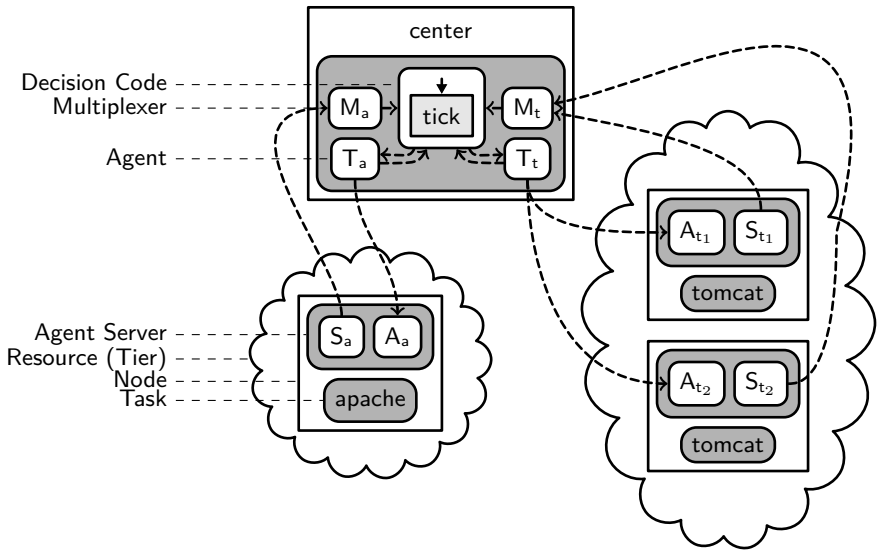

Figure 1: sams' Distributed Software Architecture for the Management of the Apache-Tomcat Application Example

respective $\mathsf{T_i}$). At last, $\mathsf{A_a}$, $\mathsf{A_{t_1}}$ and $\mathsf{A_{t_2}}$ are *task* agents (actuators) that receive command notifications (*e.g.,* configure, start or execute the task) from their managing tier agent, and execute these operations on the remote sites.

1.2 sams' Utilities for Multi-tier Application Management

sams' core source code (assumed to be located under a directory named `sams-core/` in the sequel) comes with a set of utilities dedicated to the management of multi-tier applications (all gathered into sub-packages of `fr.lig.erods.sams.ntier`). Notably, subclasses of `fr.lig.erods.sams.ntier.center.opcode.Tier` define agents encapsulating operating code to manage clusters of nodes running the same service. They offer the possibility to create and drive remote tasks (remote agents, subclasses of `fr.lig.erods.sams.ntier.depl.opcode.Node`), by reacting to dedicated artifacts (to be sent by the decision code).

1.2.1 Multi-tier-application-specific Directory

Apart from the standard files needed by sams' core and already detailed in "sams: Core Manual"[1], the application-specific directory `app/` requires the following additional information for the management of a multi-tier application with the utilities provided along with sams' core (Java sub-packages of `fr.lig.erods.sams.ntier`):

[1] `http://sams.gforge.inria.fr/manual-core.pdf`

app/etc/tiers.properties Default configuration values common to all tier agents. These values comprise various timing information for remote sensors and heartbeat multiplexers, as well as internal timeouts used to detect failures impacting sams itself. The default contents of this file is:

```
# Default values for common tier properties.

#
# Deployed monitors' properties.
#
# The time between two measurements of cpu percentage (in ms).
cpu-monitor.period 5000
# Exponentially weighted moving average factor: 0 < alpha < 1. The higher, the
# smoother (and the lesser reactive to drastic change in loads).
cpu-monitor.alpha 0.6
# The time between two checks of tasks' process status (in ms).
task-monitor.period 5000

#
# Properties used by the center only.
#
# The delay after which the cpu info muxer considers a remote node as possibly
# dead, and triggers an alarm accordingly. This amount must be greater or equal
# than 'cpu-monitor.period' above.
heartbeats-muxer.expiration-delay 5000
# Once a node is detected as repaired, some delay may be needed before it
# becomes fully available again. Hence, some extra delay may need to be awaited
# before decrementing the 'node_repairings' integer measure;
# 'node-repairings.extra-delay' represents this amount of time in milliseconds.
node-repairing.extra-delay 20000
# Internal timout values, in milliseconds.
timeout.deployment 100000
timeout.deployment.threshold 3600000
timeout.undeployment 100000
timeout.init 50000
timeout.stop 20000
timeout.kill 20000
timeout.restart 20000
timeout.configure_node 10000
timeout.notify_source 11000
timeout.failure_suspicion 5000
```

app/etc/tiers/*.properties Tier-specific configuration files: for each tier t, the optional file `app/etc/tiers/`t`.properties` defines properties on a per-tier basis, possibly overriding configuration values defined in the file described above.

Additionally, these files specify automatic data transfers between the tiers, in terms of required or published field groups:

- The `tail` fields group for a tier gathers values expected in configuration artifacts automatically transmitted by its tail tier; these data are also sent to the remote task agents belonging to the tier (see bellow);

- Alternatively, the `as_tail` fields group for a tier gathers values automatically transmitted in artifacts to the source tier whenever one of its tasks is (re-)configured and/or (re-)started. The source tier agent automatically fills in its own `tail` group with these data, and configures its remote tasks appropriately by either directly sending an artifact containing this data, or by building custom Java objects;

`app/remote/*.properties` Some additional keys can be specified in these files already required by sams' core and constituting the basis for the environment of remote task agents.

In these files, several fields groups are needed to setup configuration transmissions to and from the remote tasks constituting "adjacent" tiers:

- A `tail` fields group specifies the data read from configuration artifacts coming from the tier agent managing this remote task; it should at most comprise the fields listed in the `tail` group for the corresponding tier (see above);
- The `as_tail` fields group, in turn, specifies data that is transmitted to the managing tier agent by any remote task whenever its configuration changes and/or it is (re-)started; this should include at least the fields listed in the `as_tail` group for the corresponding tier.

1.2.2 Multi-tier-application-specific Configuration Directory

The following additional configuration files and information are needed in an multi-tier-application-specific configuration directory `cfg/`:

`cfg/center.properties` should define the default *Applications Binary Interface* (ABI) for all remote nodes, by associating the key `remote.arch` to either `i686` or `amd64`; if specified, this value overrides the one specified in sams' core (see `sams-core/src/etc/sams.properties` in the source code).

If sams' center host is also used to execute some of the "remote" tasks, then the key `local.host` can be used to specify the internal host name of the center node (recall that hosts sharing the IP layer must have identical names);

`cfg/clusters.properties` contains specifications of all clusters involved.

Node specifications are of the form [*user*`@`]*host*, the default user being the name of the one running sams' center process on the center host.

The Boolean property `reuse-failed-nodes` (*false* by default) that these files may define specifies that failed nodes of a cluster can be re-allocated immediately after a failure; this is useful for testing purpose, when faults are injected manually;

`cfg/hosts.properties` can optionally specify per-host configuration, by setting values for keys, *e.g.,* *h*`.arch`, for any host *h*; for now, only the ABI can be set this way.

2 Building the Examples

In order to build the examples described in the sequel, be sure to first be able to compile sams' core by following the instructions listed in the manual entitled "sams: Core Manual".

Three example applications are included in the distribution: two of them illustrate the combination of two tiers (either Apache and Tomcat, or MysqlProxy and Mysql), and the latter combines both so as to build a complete 4-tier web application based on the auction site of RUBiS (see `http://rubis.ow2.org/`).

2.1 Additional Dependencies Needed to Compile the Examples

The set of dependencies needed to compile sams' core and the examples included is quite different. The additional dependencies are:

BZR See `http://bzr.inria.fr/`. One noticeable dependency of BZR that we use is:

- **Sigali** Discrete Controller Synthesis tool, available at `https://gforge.inria.fr/projects/bzr` (the version required must be greater or equal than 2.4);

argos2lus This program is not yet distributed as free software (but it should be soon); you can however download the GNU /Linux `amd64` binary that is required for the compilation at `http://people.irisa.fr/Nicolas.Berthier/file:argos2lus` (just make it executable and reachable from your `PATH` as `‘argos2lus‘` — contact sams' author for other versions if needed).

JNA The Java Native Access library is used to integrate external code compiled in C due to intrinsic limitations of the JVM[2]. It is available at `https://github.com/twall/jna`.

gcc Incidentally, a C compiler is also needed to compile the examples ; any decent gcc version should fit the purpose.

2.1.1 Additional Environment Setup

- `heptc`, `sigali` and `argos2lus` should be available in the `PATH`;
- If the file `jna.jar` is not located in `/usr/share/java/`, then the `JNA_JAR` environment variable is used to find it.

2.2 Setting up the Tasks

In order to compile the examples, one also needs to setup the tasks they make use of.

The set of tasks needed to run the examples consists of Apache, Tomcat, Mysql and MysqlProxy. The build process constructs their respective archives by merging a previously compiled and locally installed version of them, and a set of extra files.

[2]Number of arguments and size of methods notably.

In the following, the tasks' installation directories are named $\$t_$TASK_DIR for each task $t \in \{$APACHE, TOMCAT, MPROXY, MYSQL$\}$.

Assuming the ABI of the build machine is compatible with the one of the target hosts (including the set of installed libraries), the following series of commands should install the softwares in their respective directories. Note that, in principle, any release versions earlyer than the ones downloaded by the code excepts below should work. {Note also that one should download files from (other) mirrors and check their integrity afterwards, instead of directly using the addresses in the scripts bellow}.

2.2.1 Apache

Install Apache first (available at `http://httpd.apache.org/download.cgi`); note you may need to update version specification in the following script:

```
# Prepare apache directory:
mkdir -p "$APACHE_TASK_DIR/src";
pushd "$APACHE_TASK_DIR/src";
# Download httpd source code, and decompress it; Also retrieve some
# source code dependencies to avoid installing them on target hosts.
wget "http://archive.apache.org/dist/httpd/httpd-2.4.7.tar.bz2";
wget "http://archive.apache.org/dist/httpd/httpd-2.4.7-deps.tar.bz2";
tar xaf httpd-2.4.7.tar.bz2;
tar xaf httpd-2.4.7-deps.tar.bz2;
cd httpd-2.4.7;
# Configure and install the program:
./configure --prefix="$APACHE_TASK_DIR" \
  && make \
  && make install;
# Cleanup the build directory:
make clean;
# Go back to initial directory:
popd;
```

Adding Tomcat Connectors In order to use our Apache version with Tomcat, one has to build an additional module, to be downloaded from `http://tomcat.apache.org/download-connectors.cgi`. We use the `mod_jk` module:

```
# Enter apache compilation directory:
pushd "$APACHE_TASK_DIR/src";
# Download tomcat-connectors' source code, and uncompress it:
wget "http://archive.apache.org/dist/tomcat/tomcat-connectors/jk/\
tomcat-connectors-1.2.37-src.tar.gz";
tar xaf tomcat-connectors-1.2.37-src.tar.gz;
cd tomcat-connectors-1.2.37-src/native;
# Configure and install the module:
./configure --with-apxs="$APACHE_TASK_DIR/bin/apxs" \
  && make \
  && make install;
# Cleanup the build directory:
make clean;
```

```
# Go back to initial directory:
popd;
```

2.2.2 Tomcat

Tomcat is available at `http://tomcat.apache.org/download-70.cgi`. The version of Java should be ≤ 1.6 to be able to build Tomcat-7 successfully. It is possible to use either the binary release or to compile it from source.

Using the Binary Release Download and decompress the archive available at `http://apache.mirrors.multidist.eu/tomcat/tomcat-7/v7.0.50/bin/apache-tomcat-7.0.50.tar.gz`, and define `$TOMCAT_TASK_DIR` to the absolute path of the first sub-directory it contains.

Using the Source Distribution In case of problems with the previous method, define `$TOMCAT_BASE_DIR` so that `$TOMCAT_TASK_DIR` is of the form `$TOMCAT_BASE_DIR/apache-tomcat-7.0.41-src/output/build/`:

```
# Prepare tomcat directory:
TOMCAT_BASE_DIR="$(dirname "$(dirname "$TOMCAT_TASK_DIR")")";
mkdir -p $TOMCAT_BASE_DIR;
pushd $TOMCAT_BASE_DIR;
# Download archive, and decompress it:
wget "http://apache.mirrors.multidist.eu/tomcat/tomcat-7/v7.0.41/src/\
apache-tomcat-7.0.41-src.tar.gz";
tar xaf apache-tomcat-7.0.41-src.tar.gz;
cd apache-tomcat-7.0.41-src;
# Configure and build:
echo "base.path=$TOMCAT_BASE_DIR" > build.properties;
ant;
# Go back to initial directory:
popd;
```

2.2.3 Mysql

Mysql can be downloaded through an interactive web page at `http://dev.mysql.com/downloads/mysql/#downloads` (to use the binary release, select "Linux - Generic" and download the Compressed TAR Archive that suits the architecture of your remote hosts; to compile it from source, select "Source Code", and download the "Generic Linux (Architecture Independent), Compressed TAR Archive")[3]. Using the source distribution allows to strip some parts of the database server to end up with smaller task archives.

Using the Binary Release Download and decompress the archive, and define `$MYSQL_TASK_DIR` to the absolute path of the first sub-directory it contains.

[3]Notice that one can skip the sign-up step when proposed, by directly clicking on the link at the bottom of the page.

Using the Source Distribution Save the archive into `$MYSQL_TASK_DIR/src/`, then execute the

following commands from the same directory. Notice you need `cmake`[4]:

```
# Decompress the archive:
tar xaf mysql-5.6.13.tar.gz;
mkdir build;
pushd build;
# Configure, build, install and cleanup:
cmake -DCMAKE_INSTALL_PREFIX="$MYSQL_TASK_DIR" \
      -DWITH_EMBEDDED_SERVER=OFF \
      -DWITH_UNIT_TESTS=OFF \
      ../mysql-5.6.13 \
  && make \
  && make install \
  && make clean;
popd;
```

2.2.4 MysqlProxy

Similarly to the Mysql case, MysqlProxy can be downloaded through an interactive web page at `http://dev.mysql.com/downloads/mysql-proxy/`.

If building from source, when in the directory where it has been downloaded, type the following commands:

```
# Decompress:
tar xaf mysql-proxy-0.8.3.tar.gz;
# Configure, build and install, and then clean.
./configure --prefix="$MPROXY_TASK_DIR"    \
    --with-mysql="$MYSQL_TASK_DIR" \
  && make \
  && make install \
  && make clean;
```

2.2.5 Creating Tasks' Archives

The `sams-ntier-tasks/` directory decompressed from the experimentation archive[5] gathers the files needed to create one archive per task and target ABI (For now, *task* $\in$ {`apache`, `tomcat`, `mproxy`, `mysql`}, and $ABI \in$ {`i686`, `amd64`}). It contains:

- for any couple $\langle task,\ ABI \rangle$, an additional directory hierarchy (under `sams-ntier-tasks/src/`*task*`-`*ABI*`/`), to be added to the corresponding task archive, and possibly **overriding** the original files in the distribution directory; as a consequence, default configuration files could be overwritten in the archive simply by placing one with the same name under this directory;

[4] *cf.* `http://www.cmake.org/`, or the packages available in your GNU /Linux distribution.
[5] `http://sams.gforge.inria.fr/archs/sams-ntier-tasks.tgz`

- an Ant build file (`sams-ntier-tasks/build.xml`) and an associated set of scripts (in `sams-ntier-tasks/bin/`) to create an archive for each task. To use the build file, one needs to fill in the distribution directories in `sams-ntier-tasks/tasks.properties` with the appropriate values for each key formed as *task-ABI* (if the directory for one couple is not specified, then the build system will not try to create the corresponding archive).

Optionally, effective creation of archives can be *tested* by running `‘ant‘` under the `sams-ntier-tasks/` directory; by default, this operation should create any archive it can in a fresh `sams-ntier-tasks/_build/` directory.

2.3 Final Steps

The `sams-ntier-examples/common/src/` directory contains most of the source code common to all examples, each of which has its main source code included in a specific sub-directory:

`sams-ntier-examples/apache-tomcat/` 2-tier Apache-Tomcat combination;

`sams-ntier-examples/sqltiers/` 2-tier MysqlProxy-Mysql combination;

`sams-ntier-examples/apache-tomcat-sqltiers/` 4-tier Apache-Tomcat-MysqlProxy-Mysql combination.

2.3.1 Configuring Deployment Targets

A new set of target hosts and clusters may need to be specified to build the application-specific configuration targeting a particular deployment platform (*i.e.,* a set of physical or virtual machines). Such a set *s* is specified by including the files listed in Section 1.2.2 directly into a directory named `sams-ntier-examples/common/etc/target-specs/`*s*`/`.

Be sure to build all needed task archives for every couple $\langle task, ABI \rangle$ implicitly induced by the specification of tasks' clusters and host ABIs.

2.3.2 Setting up Directories

Prior to the compilation of any of the examples, one needs to set the Ant property `tasks.dir` in the `build.properties` file located at the root of the example's directory, to a path to the decompressed `sams-ntier-tasks/` directory (relative paths being interpreted from the directory where `‘ant‘` is run, so in principle the same directory as `build.properties`'). The value of the `target.name` key also needs to be set to the name of the target deployment platform specified as described in the previous Section. (To speedup build times, and due to the possibly significant size of the archives that are going to be built, it is advised to set the `build` key in the same file to a directory that is not located in an NFS mounted file system.)

The `sams-core.dir` entry in the `sams-core.properties` files in each example directory also needs to be set to a path to sams' core distribution.

2.3.3 Compiling

Running ‘`ant`‘ in an example source code directory should produce after some minutes[6], a set of application-specific or corresponding configuration directories under `${build}/dists/`. Besides, if `skip-archives` is set to `false` in the corresponding `build.properties` file, one archive for each one of them is added in a newly created `output/` directory.

‘`ant distclean`‘ cleans up all temporary build files, and the distribution directory (`output/` by default). Other noticeable Ant targets include `compile.java` and `java.clean` to compile and clean Java source code only; `compile.lib` to compile the native library comprising the tick and used through JNA; this last target involves several compilation stages including discrete controller synthesis with SIGALI.

In a particular example directory, ‘`ant example.whole.doc`‘ builds under a new `output/doc/` directory the whole Javadoc documentation for both sams' core and the classes of this example; ‘`ant example.doc`‘ builds under `output/doc/example/` the Javadoc for the classes of this example only.

2.3.4 Basic Console Usage

The commands provided by the console for the n-tier deployment examples are the following (these commands, except the `quit` one, are specified in the class `fr.lig.erods.sams.ntier.center.Center`, defined under the `sams-ntier-examples/common/src/java/` source code directory):

`start` starts deployment of the application;

`stop` stops current deployment;

`quit` radically stops the center process (without any care about remote agents: you need to execute `stop` before!);

`up` & `down` signal fake overload (resp. underload) to the model.

2.3.5 Some Caveats

If you test one of sams' examples involving a Tomcat tier on a computer whose "default" network interface is down — typically, when it's not physically plugged to a network — then don't forget to comment the "`<Cluster ...>`" line in `sams-ntier-tasks/src/tomcat/conf/server.xml`; otherwise, Tomcat refuses to start.

3 Further Digging the Source Code of the Examples

In order to illustrate the definition of the application-specific parts, lets detail the specifications for the examples.

[6]The compilation is about ten times longer for the `apache-tomcat-sqltiers` example than for each of the two others.

3.1 Operating Code Specification

No concrete class is provided by sams' core yet to specify agents and objects belonging to the operating code, since they are only required to handle artifacts sent by the tick. So, the only requirement is to write a class implementing the interface `fr.lig.erods.sams.center.ModelBuilder`.

For each example, the name of the class building the operating code is the value of the key `model-builder.class` in the file `etc/env.properties`; its source code is specific to each example, and is thus present under their individual `src/java/` directory.

These methods essentially build up environments from properties files, instantiate subclasses of `fr.lig.erods.sams.ntier.center.Tier` (to be looked for in sams' core source directory `sams-core/src/java/`, each concrete subclasses being located in `sams-ntier-examples/common/src/java/`), and a subclass of `fr.lig.erods.sams.center.DecisionCode`. An instance of the latter encapsulates the decision logic. Additionally, the `tick.max-delay` global environment entry of the center (that can be overridden on an example basis, in the files `etc/env.properties`) specifies the maximum amount of time between two steps of the tick (defaulting to 10s).

3.2 Decision Logic Specification

Although it could be designed in a more modular way, the model encoding the decision logic is currently highly application-specific[7]. Hence, its entire source code is actually to be looked for in the sub-directories of `sams-ntier-examples/`.

For each example, the tick object is built from the BZR node `src/ept/tick.ept`. This file defines a `tick` node encapsulating rather generic models whose BZR or ARGOS source codes are located in the `sams-ntier-examples/common/src/ept/` and `sams-ntier-examples/common/src/argos/` directories:

`tier_model.argos` is the behavioral model of a non-replicated tier;

`tier_model_repl.argos` is the behavioral model of a replicated tier, that supports simultaneous sizing operations;

`repair.argos` encodes a controllable repair manager;

`sizing.ept.pp` specifies a basic controllable sizing manager. Note that it does not support simultaneous sizing operations (*e.g.,* start an add operation while still adding a node), even though the replicated behavioral model does;

`tier_properties.ept.pp` coordination invariant specifications;

`two_tiers_ctrld.ept.pp` defines the generic combination of one non-replicated tier using services provided by a replicated one;

[7] Modularity of components involving DCS is a research problem yet to be addressed in our case.

`four_tiers_ctrld.ept.pp` defines the generic combination of one non-replicated tier using services provided by a replicated one, itself using services provided by a third non-replicated tier using services provided by a fourth replicated one;

`include/tier_itf.ept` defines record types, gathering tier-related notifications and commands notably.

3.3 Overall Configurations

Several configuration properties are available to tune the overall behavior of sams management infrastructure. Default common values for the tiers involved in the examples are defined in a file located at `sams-ntier-examples/common/src/etc/tiers.properties`; its default contents was listed in Section 1.2.1. These default common values may be overridden on a tier-specific basis, by using tier-specific properties files in `sams-ntier-examples/common/src/etc/tiers/`, or `sams-ntier-examples/common/etc/tiers/` (the latter going into the application-specific configuration directory instead of the application-specific directory).

3.4 Detailing the Extra Example Files for the Tasks

The provided experimentation archive defines an example set of extra files to be added to the tasks archives; they build up an extended version of RUBiS's auction website. They include various files, some being related to the configuration of the tasks.

3.4.1 Configuration Files

As configuration files highly depend on the tasks to be executed, and contain information that depend on the topology of the whole application, then the provided examples use shell scripts to customize their content dynamically.

The set of configuration files of a task t goes paired with a set of procedures defined in the `sams-ntier-examples/common/src/bin/`t`-cmd` script in the source code of the examples. For instance, the `init()` (resp. `start()`, `stop()`...) procedure is executed on remote hosts prior to the start-up of any task, according to the values of the `init` (resp. `start`, `stop`...) entries in `sams-ntier-examples/common/src/etc/remote/`t`.properties`. Note that these scripts must be included in the list of transmitted files for their respective task (see `sams-ntier-examples/common/src/etc/tasks.properties`), and are directly executed from the remote location directory (hence the command lines start with "`./`").

3.4.2 Static Contents

The contents consists in a set of static files defining, *e.g.,* static HTML contents for Apache, a compiled Java application for Tomcat, a custom proxy script for MysqlProxy, and a Mysql database.

4 Testing the `apache-tomcat-sqltiers` Application

The main source code directory of the `apache-tomcat-sqltiers` example contains a `test/` sub-directory featuring a set of files that can be used to check the behavior of the management software for the 4-tier application example.

4.1 Setup

In order to use the test files, one first needs to modify variables in the header of the `test/test.sh` script (and to read the notices at the top of this file). The variables to define are the following:

`sams_dir` should be the path to sams'core distribution directory;

`remote_user` is the name of the user (common to all monitored nodes) that is used to execute the monitoring scripts; this user is also the one executing the program on the node whose failure is simulated;

`remote_hosts` is the list of monitored hosts;

`injection_user` & `injection_host` define the node from which the HTTP request injection is executed;

`injection_files` lists the files to be transmitted to the injection host (in the home directory of the injection user);

`injection_cmd` is the JMeter command executed as remotely for workload injection[8];

`failing_host` is the node whose failure is simulated;

`failure_cmd` is the command to be executed to simulate the failure of the host.

4.2 Execution

1. The first execution step consists in deploying the monitoring scripts; to do so, execute the `test.sh init` command;
2. The second execution step consists in the start-up of the management infrastructure on the center node;
3. Executing `test.sh run` will start the remote monitoring scripts, execute the HTTP injection, and then stop the remote monitoring scripts; resulting data are then retrieved in the current directory, and transformed into Gnuplot-friendly files.
4. Looping to steps 2. or 3. should be possible any number of times;
5. Lastly, executing `test.sh clean` cleans up remote monitoring scripts.

[8] `http://jmeter.apache.org/`