TransLattice Application Platform (TAP) Information and Troubleshooting

David Lambert

July 25, 2012

Kim's (Formerly) Secret Manual

Kim Wallace has created a manual containing lots of valuable troubleshooting information: https://dolphin.eng.translattice.com/newwiki/bin/view/Main/SecretManual. Kim's manual complements this document.

The Software On Each Node

The operating system of each node is a modified version of Ubuntu Server 12.04 LTS (Precise Pangolin). As is usual with Ubuntu Server, the Desktop is not installed. Also, "man" is not installed; some programs have been disabled or removed to optimize node performance.

A customized version of PostgreSQL 9.2.0 is located in /usr/bin, and Jetty 6.1.24 is located in /tlbin. Except for PostgreSQL, all TransLattice executables are located in /tlbin.

The database portion of the product is called the TransLattice Elastic Database (or "TED"). This is a modified version of PostgreSQL that is designed to run in a distributed fashion. A single running instance of this product is called a "node". A collection of nodes working together in a group is called a cluster. Most of this document is devoted to TED. There is another product called TAP (TransLattice Application Platform). When an application is deployed through the Administrative Interface, two things happen on each node: that application is deployed in Jetty, and a corresponding PostgreSQL database is created[1]. The application's ID is a number, and the default name of the database is "appdb<Application ID>". The application portion is only deployed on TAP. An application is added to TED to create the database instance to run an external application in a separate environment.

Running "ps –ef | more" on a node will show a variety of processes executing, including PostgreSQL and Jetty. Most of these running processes are parts of TED.

Methods Of Accessing A Node

There are three methods of accessing a node described here. An operating system login and a PostgreSQL login are both possible, and are described in this document.

The preferred method is through the Administrative Interface. The "admin" login of the Administrative Interface is neither a database nor an operating system login, is an entirely separate login used only by the Administrative Interface. Customers should only use the Administrative Interface to access nodes.

[1] An installed PostgreSQL instance can have multiple databases.

Node Access Method 1: Through The Administrative Interface

The Administrative Interface is the preferred method for accessing TAP. Please consult the System Administration Guide for instructions on how to use the Administrative Interface.

Node Access Method 2: Through PostgreSQL

Psql, the PostgreSQL database client, can be run on the command line on a node. PostgreSQL logins from other systems, including a GUI client such as pgAdmin3, are disabled by default. To enable remote PostgreSQL logins:

1. Go in the Administrative Interface.
2. Select the "Config" tab at top.
3. On the left side, under "Configuration", select "Options".
4. On the right side will be a checkbox labeled "Allow Remote Connections".
5. Check that checkbox.
6. Press the button to the right labeled "Save". These 6 steps will allow one to ping the node.
7. If you have a PostgreSQL login, use it and stop here. Otherwise, continue following these directions to create a PostgreSQL login.
8. Go to the "Apps" screen of the Administrative Interface.
9. On the left side, under "Application Settings", select "Database Console".
10. On the right side, under "Console", select any node, and press the "Connect button".
11. In the text box at the bottom, type in the sql command
    "create user <username> <role> password '<password>';"
    For example, "create user pguser superuser password 'pguser';"

Note that the user created in step 11 above was created on only one node, but TED propagates that user to all node databases. This propagation occurs for most database objects (enumerated below).

Node Access Method 3: Through the Node's Operating System

Logging Onto A Node

Sometimes, the Administrative Interface and a PostgreSQL connection do not get us the information we need, and we have to actually log onto the node. We typically use "ssh". To use ssh on a Mac:

1. Create a Finder window.
2. Go into the "Applications" folder.
3. Then go into the "Utilities" folder
4. Click on "Terminal" to get a command line prompt.
5. At the prompt, execute "ssh root@<Node Address>", where <Node Address> is the node's IP Address or DNS name.

Copying Files To Or From A Node

To copy a file from a node to a Mac, follow steps 1-4 above, and at the prompt, execute "scp root@<Node Address>:<Node source full path> <Mac destination full path>".

To copy a file from a Mac to a node, follow steps 1-4 above, and at the prompt, execute
"scp <Mac destination full path> root@<Node Address>:<Node source full path>".

Out of the box, SCP GUI programs like CyberDuck and Fugu do not work because SFTP is not enabled on the nodes. To enable SFTP, open /etc/ssh/sshd_config in a text editor, remove the hash mark at the beginning of the line "#Subsystem sftp /usr/lib/openssh/sftp-server". Then kill -9 the sshd process, and execute "/usr/sbin/sshd" to restart sshd.

TransLattice Version Information

The files in /manifests give all the build and version information for TAP. The overall revision number is at the end of the first line of /manifests/foundation.sum.

TransLattice Log Files

As described in the System Administration Guide, Alerts can be viewed in various places in the Administrative Interface. Alerts are a subset of the information written to the log files on the nodes in a cluster. While troubleshooting, you may find that Alerts do not contain all the information needed, and access to the original log files is needed. In this section, we describe these log files.

With the released versions of Ubuntu, PostgreSQL, and Jetty, all three programs have their own log file. In the TransLattice versions of Ubuntu, PostgreSQL, and Jetty, there are no separate log files. Instead, there is one combined log file, named /var/log/tlsys.log. Note that each node has its own tlsys.log: log entries specific to the local node are not propagated to tlsys.log on other nodes. Thus, troubleshooting TED may require getting tlsys.log from a number of nodes.

The preferred method for viewing tlsys.log is through the Administrative Interface's Monitor tab, where you can download the entire log file, or you can search through it for lines containing certain strings.

Tlsys.log can also be viewed on each node in ssh or copied to other machines using SCP, as described previously. If you access tlsys.log one of these ways, you should know that it is compressed using gzip, so its contents are not human readable as is. Because of how tlsys.log is written, gzip –d can't decompress tlsys.log. Instead, the contents of tlsys.log can be viewed running zmore, zcat (e.g. "zcat tlsys.log > tlsys.txt"), or zgrep, all of which are part of gzip[2]. TransLattice has created /tlbin/ztail to parallel the functionality of tail. Gzip, zmore, zcat, and zgrep are found on all Mac's, and can be found in any operating system where gzip is installed.

The uncompressed contents of tlsys.log can be analyzed in a number of ways. Awk, grep, or zgrep can be used on the command line. The uncompressed tlsys.log can be imported into Microsoft's Excel or the equivalent of Excel in OpenOffice.org or LibreOffice.org.

[2] When running zcat or zmore on tlsys.log, the message "gzip: tlsys.log: unexpected end of file" always occurs. This message is normal and is of no concern.

The format of tlsys.log is different from the format described in the System Administration Guide, in the section titled "System Monitoring Screen". Each Log Data field is delimited on both sides by the "|" character. These fields, from left to right, are:

1. The Module Name (up to 17 characters has been observed).
2. The Process ID (a number between 0 and 32,768, inclusive).
3. The Thread ID (a 12-digit hexadecimal number).
4. The UTC Date and Time, in the format <YYYYMMDD>T<HHMMSS>.
5. The Log Level, with values 3, 8, 13, 15, and 18. These numbers translate as follows:
   3 – Critical
   8 – Major
   13 – Alert
   15 – Dbdebug
   18 – Informational
   25 – SysDebug
6. The number 0.
7. The actual text of the logged message (up to 235 characters has been observed).

This command will find SQL statements in tlsys.log:

```
zgrep "UPDATE|INSERT|DELETE|SELECT|CREATE|DROP" tlsys.log.
```

Below is a shell script, written by development for fetching crucial information and writing it to file. Note that it finds only currently running SQL statements, not completed SQL statements, whereas zgrep can fetch both types of SQL statements.

```
#!/bin/bash
if [[ $# -lt 2 ]]
then
        echo "tball2.sh <output_file> <Node> "
        exit
fi
echo '----- jetty processes'                            >/cores/tball_info.out
ps aux | grep Postgres | grep -v jetty                   >>/cores/tball_info.out
echo '----- free'                                        >>/cores/tball_info.out
free                                                     >>/cores/tball_info.out
echo '----- all processes'                               >>/cores/tball_info.out
ps aux                                                   >>/cores/tball_info.out
echo '----- background workers processes'                >>/cores/tball_info.out
ps aux | grep 'bg worker' | wc –l                        >>/cores/tball_info.out
echo '----- remote transactions processes'               >>/cores/tball_info.out
ps aux | grep Postgres | grep "remote transaction"       >>/cores/tball_info.out
echo '----- distributed query  processes'                >>/cores/tball_info.out
ps aux | grep Postgres | grep "distributed query plan" >>/cores/tball_info.out
echo '----- Select  processes'                           >>/cores/tball_info.out
ps aux | grep Postgres | grep SELECT                     >>/cores/tball_info.out
echo '----- Insert  processes'                           >>/cores/tball_info.out
ps aux | grep Postgres | grep INSERT                     >>/cores/tball_info.out
echo '----- Update  processes'                           >>/cores/tball_info.out
ps aux | grep Postgres | grep UPDATE                     >>/cores/tball_info.out
```

```
echo '----- Delete  processes'                    >>/cores/tball_info.out
ps aux | grep Postgres | grep DELETE               >>/cores/tball_info.out
echo '----- Out of Memory Messages'                >>/cores/tball_info.out
zgrep oom /var/log/tlsys.log                       >>/cores/tball_info.out
echo '----- Unicast Messages'                      >>/cores/tball_info.out
zgrep tlc_unicast /var/log/tlsys.log               >>/cores/tball_info.out
echo '----- No good Memory Messages'               >>/cores/tball_info.out
zgrep 'No good' /var/log/tlsys.log                 >>/cores/tball_info.out
echo '----- TRAP Messages'                         >>/cores/tball_info.out
zgrep TRAP /var/log/tlsys.log                      >>/cores/tball_info.out
echo '----- Shoudnt Happen Messages'               >>/cores/tball_info.out
zgrep SHOULDN /var/log/tlsys.log                   >>/cores/tball_info.out
echo '----- Killing Messages'                      >>/cores/tball_info.out
zgrep killing /var/log/tlsys.log                   >>/cores/tball_info.out
echo '----- SegFault of Memory Messages'           >>/cores/tball_info.out
zgrep segfault /var/log/tlsys.log                  >>/cores/tball_info.out
echo '----- End of  Messages'                      >>/cores/tball_info.out
tar -czhf  $1-$2.tar.gz  /cores/*  /usr/bin/Postgres  /var/log/tlsys.log /manifests/*.sum `ldd -v
/usr/bin/PostgreSQL | awk -F '>' '{print $2}' | awk '{print $1}' | grep -v ^$ | grep / | sort -u`
>/dev/null 2>&1
```

The Structure of Standard PostgreSQL and TED

The Structure of Standard PostgreSQL

Standard PostgreSQL is organized into layers. From the highest level on down, the layers are:

The Layered Structure of Standard PostgreSQL

1 Cluster						
Servers						
Databases: postgres, maybe others						Users
public Schema			pg_catalog Catalog		information_schema Catalog	
Tables	Indexes	Views	44 Metadata Tables	36 Metadata Views	55 Metadata Objects	
Records		Records	Records	Records	Records	
Fields		Fields	Fields	Fields	Fields	

Table 1

A good starting point is a single installation of PostgreSQL on a single server (the second row from the top of Table 1). A single server can contain multiple databases. Users are contained in servers, not in databases, and are present in all of the server's databases, though the privileges of a user can vary from database to database. In standard PostgreSQL, each server always has a database named "PostgreSQL".

Just below the database layer is the schema layer. Each database can contain multiple "schemas", and schemas are containers of the usual database objects: tables, views, functions, etc. In standard

PostgreSQL, every database always has a schema named “public” and two “catalogs” (like schemas) named “information_schema” and “pg_catalog”. Pg_catalog contains 44 metadata tables, while information_schema contains 55 metadata tables. These metadata tables are well documented in PostgreSQL books, so we won’t go into them here.

Just below the table layer is the record layer, and the layer below the record layer is the field layer. And there can be one more layer above the server: a cluster. Information moves between servers, but the each server retains a separate identity.

The Structure of TED

The layered structure of TED is similar to the layered structure of Standard PostgreSQL. From the highest level on down, the layers are:

The Layered Structure of TED

1 Cluster						
Databases: postgres, fxc, and appdb*						Users
public Schema			pg_catalog Catalog		information_schema Catalog	
Tables	Indexes	Views	44 Metadata Tables, tl_shard	36 Metadata Views	55 Metadata Objects	
Shards (0-1 per server)	Shards (0-1 per server)		Shards (0-1 per server)		Shards (0-1 per server)	
Records		Records	Records	Records	Records	
Fields		Fields	Fields	Fields	Fields	

Table 2

There is no longer a Server layer between the Database and the Cluster layers. Each Schema, Catalog, Database, and Table exists on all servers (aka nodes). The cluster contains the “postgres” database, the “fxc” database (used by TED), and at least one database named “appdb<App-ID>”, where <App-ID> is the ID of the application that the database is for.

In standard PostgreSQL, schemas can be created at will. But in TED, schemas cannot be manually created. Instead, the usual “public” schema and the usual catalogs “information_schema” and “pg_catalog” are created. All TED applications must use the public schema in the appdb* databases.

Between the Table and Record layers is a new layer: the Shard layer. Shards contain the actual records; they are the actual physical tables. Each shard exists on one node. The “Tables” in Table 2 are top-level virtual tables that appear on each node, but they are not real tables, they are abstractions designed to hide the complexities of their associated collection of shards. Customer applications interact only with the top-level virtual tables, and behind the scenes, TED handles the translation between the top-level virtual tables and the physical shard layer.

All nodes will have the top-level virtual table, but some nodes may not get shards for a particular top-level virtual table if there are more nodes than required by the policy redundancy rules. TED does not store everything everywhere like in traditional replication systems.

The shard tables are named so that each shard has a name that is unique across the entire schema the table occurs in: <App-Table>_<Shard-GUID>_<Disk-GUID>, where <App-Table> is the name of the shard's parent application table, <Shard-GUID> is the GUID value of the shardid column for the shard's record in table tl_shard, and <Disk-GUID> is the GUID of the node's disk as displayed in the Administrative Interface.

An Example of a Table in TED

An example may be useful. Let's consider a cluster with four nodes whose names are "1", "2", "3", and "4", and each node has one disk. Let the disk GUIDs be "1", "2", "3", and "4", respectively (in reality, the disk GUIDs are long strings of hexadecimal numbers).

In the database console window, execute CREATE TABLE foo (id INTEGER PRIMARY KEY, comment VARCHAR(255)). The virtual top-level table "foo" will be created on all four nodes. But no shards are created; they're only created when records are inserted into "foo".

Next, create a policy template with a redundancy rule that requires that all records in "foo" have three copies, each copy being on one node. Thus, each shard will appear on only three of the four nodes.

Now execute INSERT INTO foo (id, comment) VALUES (0,"pickles"). A record will be inserted into tl_shard for the three shards that are about to be created. Let the GUID in tl_shard.shardid be "A" (in reality, this GUID is a long string of hexadecimal numbers). The shard tables on nodes "1", "2", and "3" will be named "foo_A_1", "foo_A_2", and "foo_A_3". Each of these shard tables will contain the record with id=0 and comment="pickles". Node "4" has no records and no shard tables, but like all the other nodes, it will have the top-level virtual table "foo" on it.

When the application needs the contents of table "foo", it will issue a SQL SELECT table for table "foo".

Indexes in TED

In TED, indexes are also sharded. This makes sense, since indexes can contain a considerable amount of information. In analogy to how the table being indexed is sharded, the top-level virtual index is present on all nodes. Each node's index shard is a physical index that indexes the data in that node's table shard.

In TED, the create index command will not allow a name for the index; index names are created by TED. The top-level virtual index's name is <Table-Name>_<Suffix>, where <Suffix> is some automatically assigned suffix and <Table-Name> is the top-level virtual table being indexed. The index shard name is will be <Table-Name>_<Shard-GUID>_<Disk-GUID>_<Suffix>, where <Shard-GUID> is the value of the shardid column for the table shard's record in table tl_shard, and <Disk-GUID> is the GUID of the node's disk as displayed in the Administrative Interface.

Because there are hyphens ("-") in the GUIDs used to generate the names of table and index shards, the names of table and index shards used in SQL statements must be enclosed in double quotes, otherwise PostgreSQL will complain about the presence of those hyphens.

Tablespaces in TED

In a standard PostgreSQL server, tablespaces may be created as desired, and tables and indexes may be placed in any tablespace. In contrast, TED automatically creates tablespaces and assigns tables and indexes to them. Top-level virtual tables and indexes are always in the tablespace tl_distributed. Each node has its own tablespace, tl_<Disk-GUID>, and the table and index shards on a node are placed in that tablespace.

System Objects in TED

System objects in TED and PostgreSQL are identical, with two exceptions:

- TED has a new system table named tl_shard.
- TED's pg_class has two new fields: relpkeypartition and relpkeymask, which are derived from the two new CREATE TABLE options SHARDPARTITION and SHARDMASK.

The system tables pg_database and tl_shard are identical on all nodes. For all other system objects, each node has its local version of the system object, listing information local to that node only. Table and index shards are listed, along with their parent tables and indexes, in the system objects that contain table and index information, respectively.

Accessing Information on Cluster Topology

As previously stated, the system tables pg_database and tl_shard are the same on all nodes. All other system objects have only local versions on each node and contain information only about the local node (verify this). Thus, the node databases know only local information: node databases contain no information about the cluster topology and no information that allows one node to find another node.

This information is contained in JSON objects in 'confsrv', and is read by TED at node startup. Changes to confsrv typically restart TED, so that TED gets the latest information. Confsrv is also the source of cluster information for the Administrative Interface and for TAK. The interface to confsrv on a particular node is /tlbin/cli. The complete list of commands can be found by running "cli list".

The only command we'll discuss is 'confget', which fetches confsrv information. The complete command line is "/tlbin/cli confget -k <key_value>". <key_value> is case-sensitive.

The fetched data is arranged in a tree. The root of the tree is "", and specifying <key_value>="" will fetch the entire data tree, which is megabytes of information. The allowed first-level key values are listed in Table 3 below.

<key value>	Description
"SOAPServices"	
"_ephemeral"	
"_globalStats"	
"_stats"	
"_uiActive"	
"alreadyJoined"	
"apps"	
"dataRules"	Cluster policies and rules
"dbTuningParameters"	
"disks"	Node disks
"flayer"	
"ntpdate"	
"postgres"	
"procmon"	
"tleng"	
"tms"	
"topology"	Cluster topology
"users"	Administrative Interface Users

Table 3

For example, suppose we execute /tlbin/cli confget -k “postgres”. This command might generate this output:

```
{
 "_value": {
  "allowRemoteConnections": true,
  "auth": {
   "method": "md5"
  },
  "dataDir": "/postgres",
  "port": 0x1538,
  "replicationOn": 0
 },
 "_key": "postgres",
 "_errno": 0
}
```

This output shows that the valid second-level key values are “allowRemoteConnections”, “auth”, “dataDir”, “port”, and “replication”, and it shows that “auth” has a third-level value “method”. The separator between different levels is a period (.), so we can specify <key_value>=”postgres.auth” or even <key_value>=”postgres.auth.method”. If we specify <key_value>=”postgres.dataDir”, we get

```
{
 "_value": "/postgres",
 "_key": "postgres.dataDir",
 "_errno": 0
}
```

This output shows that the returned error value is 0 (no errors), the specified key was "postgres.dataDir", and that the fetched value is “/postgres”.

Creating EC2 Nodes

To start working with EC2, go to the URL https://quartermaster/provtool. You’ll be prompted for a username and password. Once you finish logging in, you’ll see a bulleted list of hyperlinks:

- images
- ami
- customers
- nodes
- regions
- volumes
- orphans

“Images” displays a list of all TAP builds. Usually, you’ll want to you the one that is “blessed” or the one at the bottom of the list.

“Customers”, not surprisingly, shows a list of all customers with EC2 nodes, and a list of nodes for each of these customers. For simply experimenting with TAP, you’ll be customer #2, which is TransLattice Development.

“Nodes” displays a list of all existing builds, plus at the bottom is a hyperlink for creating a new node.

“Regions” displays a list of all regions, which are the highest level grouping of Amazon’s EC2 nodes.

To create a node, click the “nodes” hyperlink. Then click the “[Create Node]” hyperlink, which takes you to a screen where the new node’s properties are selected. In the “Customer ID” text box, enter your customer ID, which is usually “2”.

For the “Build Number” dropdown, select the desired build. You’ll need to select an Availability Zone, which are subsets of the Regions listed under “Regions”. There are three groups of Availability Zones: Asia Pacific Zones are prefixed by “ap-“, European Zones are prefixed by “eu-“, and United States Zones are prefixed by “us-“.

There are four options for Service Level. For simple experimentation, pick the option with “standard developer machine” in it.

In the “Purpose” text box, it helps to type in a description that is memorable.

To start the process of instantiating a node, press the button “Submit Query” at the bottom. This will take you to a page that shows your new node’s node number, its IP address (eipv4), and two or three hyperlinks at the bottom. Click the “[Control Node]” hyperlink. On the next page, press the “Launch” button, which starts the instantiation of that node and takes you to a screen that displays the status of the node instantiation. When the node is running, you’ll be taken back to the previous page that listed the node’s number and IP address, but will also say that the status is “running”.

Copy the URL on that page, and navigate to that URL in the browser. You should get a screen saying that the connection is untrusted. Go ahead and connect to that URL. You should then go to the cluster setup wizard.

To delete a node, go to https://quartermaster/provtool/nodes/<node-id> and click "[Control Node]". You'll get taken to a screen with two buttons on it: "Launch" and "Terminate". Press the "Terminate" button. The page will then say "node terminated", then change to the usual screen with the node's URI, node_id, eipv4, and state. Eventually, the state will change to "terminated", and the hyperlink "[Delete Node]" will appear at the bottom. Click that hyperlink. You'll be taken to a screen with two check boxes and a "Delete" button. Check both check boxes and press the "Delete" button.

NEEDS CLEANING UP AND VERIFYING.

SQL Tracing TED

This only affects the current connection: SET log_statement = 'all';

To get the current setting: SELECT current_setting('log_statement');

Miscellaneous

The Java binaries are in /usr/lib/jvm/java-6-openjdk, its jar files are in /usr/share/java.

To get jetty version, execute "java -jar start.jar –version"

Jetty has output files in /var/log/jetty, config files in /etc/jetty, and jars in /usr/share/jetty.

The Admin Interface web pages are in /var/www/admininterface, its home page is AdminInterface.html.

The Postgres binaries are in /usr/bin. The Postgres "config" files are in /etc/postgres, but really they're generated from the lua files in /flua.

Custom data types are forbidden.

To get access to the TAP console status screen, ssh –XC muzzle, and run virt-manager. Where is this screen on the node file systems? Access it through a serial port, according to Kim.

To time a bash command "x", run "time x" instead of "x".

For SQL logging on standard PostgreSQL, use these commands:

SELECT current_setting('log_statement');

SELECT set_config('log_statement_stats', 'off', false); false for current session, true for current transaction.

log_statement_stats
log_statement(enum) = none, ddl, mod, or all.

These do not work on TED, will have to figure out how to do the equivalent in TED.

When dropping tables or indexes, use “DROP [TABLE|INDEX] <Object-Name > CASCADE;”