TATP

Telecommunication Application Transaction Processing (TATP) Benchmark

Benchmark Suite Guide

Version 1.1

IBM Software Group
Information Management

http://tatpbenchmark.sourceforge.net

TATP Suite Guide

Document version 1.1 (TATP version 1.1.0)
Last modified on 6 July, 2011

Table of Contents

1 Introduction

The purpose of the Telecommunication Application Transaction Processing (TATP) benchmark is to measure the performance of a database system from the perspective of a typical telecom application. The benchmark emulates a Home Location Register (HLR) application of a mobile telephone network. The related HLR database is used to store information about the users and the associated services. In TATP, the load is generated by concurrent applications (clients) running pre-defined transactions on the target database. With TATP, you can evaluate the performance of any SQL database product (supporting the ODBC interface) in the HLR or similar Home Subscriber Services (HSS) application area. The TATP benchmark is specified in detail in a separate document [1].

This document describes a system built to run the TATP benchmark, called *TATP Benchmark Suite*. In the following section, the principles of the TATP Benchmark Suite operation are laid out. In section *Running the Benchmark*, the instructions how to execute the TATP benchmark are given. The section *Input Files* contains a detailed introduction to the input files needed to run a benchmark. The section *Output Files* describes the files resulting from a benchmark run. In the section *Test Input and Result Database*, the associated test input and result database in introduced. In Appendix 1, the internal architecture of the TATP Benchmark Suite is unveiled. The expected audience for this document are software engineers intending to run the TATP test against one or more database products.

The TATP Benchmark Suite is built to be database product agnostic. However, because it was developed by the IBM solidDB team, and has been heavily used in solidDB product testing, there are a few solidDB-specific optional features. The solidDB-specific features are clearly marked in this guide.

2 Operation of the TATP Benchmark Suite

2.1 Principles of Operation

A typical test run consists of three phases: (1) database population, (2) test ramp-up, and (3) performance measurement (sampling). The database is populated with initial data before the actual benchmark is started. The data is generated in a way that satisfies the given statistical distributions in key values, and cardinalities, as specified in [1].

The timing of a test run involves the ramp-up and sampling (measurement) phases. For example, following the ramp-up time of 10 min, the sampling phase is run for 20 minutes. For big disk-based databases, both periods should be long enough to expose stable behavior. In in-memory databases, the test periods can be shorter (say, 5 + 10 minutes) due to the fact that there is no page buffer pool to warm up, and the system attains a quiesced state in a shorter time.

TATP uses seven pre-defined transactions that insert, update, delete, and query the data in the database. During the run, the number of times each transaction is executed follows the transaction probability assigned for the transactions in the specified transaction mix. The transaction are specified in SQL in a separate file. They can be modified to reflect SQL syntax differences among products.

Test runs can be combined into a collective benchmark session that can be used to execute a series of test runs covering a certain test dimension, such as user load (number of client threads) or Read/Write ratio.

To setup the test runs, two types of definition files are used. A Database Definition File (DDF) includes, among others, the target database identification, connection data, and its configuration settings. A Test Definition File (TDF) includes the session identification, the database population (in terms of number of subscribers), and the actual test run definitions. See Section *Input Files* for more details about the files.

For each benchmark run, TATP reports Mean Qualified Throughput (MQTh) of the target database. The throughput is calculated as the sum of successfully executed transactions from all clients divided by the duration of the test run. The result is the average transaction rate per second. The throughput data is collected over the whole benchmark as a function of time with the given resolution (minimum resolution is one second). Thus, for further study, the throughput can also be displayed as a function of time. The response times for each transaction type are also collected and summarized over the measurement time with one microsecond resolution.

All the data collected during a test session is stored in the Test Input and Result Database (TIRDB) that can be implemented with any SQL database system.

TATP is implemented in C programming language. The executables are delivered for a few platforms. The source code is also included in the delivery package. The ODBC interface is used for access to the target database.

You should configure the database products under test in a comparable way. Especially, ensure that the following applies:

- Adjust the in-memory cache (shared buffer pool) sizes to be the same.
- Set the transaction isolation level to be the same.
- Set checkpointing and logging to be approximately the same.

For detailed specification of the database system configuration settings, see the document "Telecommunication Application Transaction Processing (TATP) Benchmark Description" [1].

2.2 TATP Benchmark Suite Architecture

2.2.1 Basic test configuration

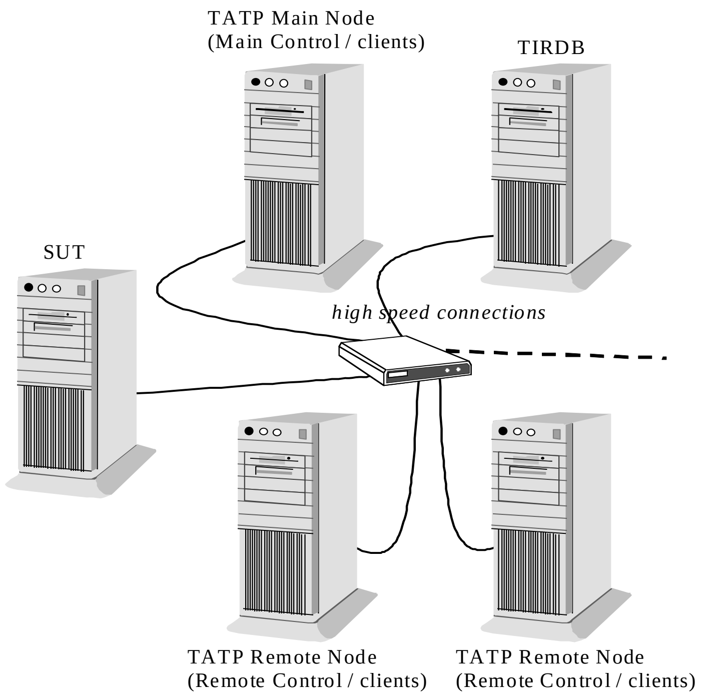

Fig. 1. A Possible Hardware Configuration of the TATP Benchmark Suite.

Fig 1 shows a typical configuration of a TATP test setup. A SUT (System Under Test) system is hosting the target database. One or more client computers can be used to generate the load. There is at least one TATP client mode called the Main Node. If more nodes are needed to generate the load, the additional nodes are called TATP Remote Nodes. The test is started at the Main Node, which automatically sends the test configuration data to the Remote Nodes, coordinates the execution of the overall test session, and collects the test results. By default, all instances of TATP load generators are multi-threaded processes whereby the total number of client threads represents the "user load".

It is wise to dedicate a separate computer to run the Test Input and Result Database (TIRDB). The existence of TIRDB is not mandatory. If TIRDB is not present, the test results (most of them) are written to files and the console.

While selecting the hardware configuration to be used in the TATP Benchmark Suite runs, ensure that the computers running TATP clients will not become the test bottleneck. At the same time, the SUT computer should closely resemble a database server computer used in real production. Typically, this involves a lot of memory, fast disks, and separate disks for the database files and log files.

2.2.2 Test configuration for consolidated systems

Separate TATP client nodes are not needed if a *consolidated system* is tested. In such a system, both the applications and the database server run in the same computer node. A consolidated system is representative of the trend to "scale-up" by way of high-end multicore platforms. A setup for a consolidated system testing is shown in Fig. 2. It can be collapsed to a single computer node if TIRDB is not used.

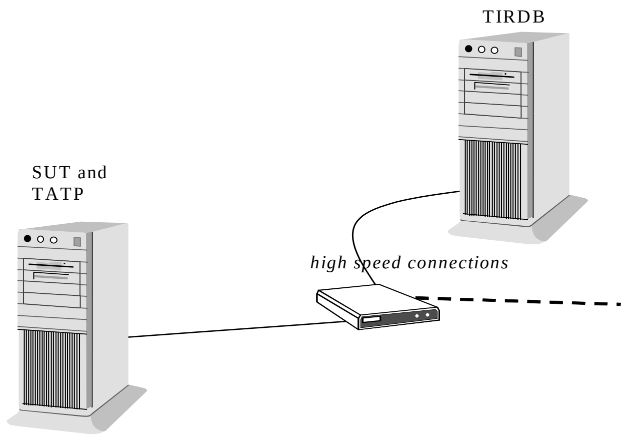

Fig. 2. TATP setup for testing consolidated systems.

2.2.3 Other configurations and options

Configuration capabilities of TATP make it possible to test databases in various configurations and test sequences.

The flexibility starts with the structure of a benchmark execution, A single invocation of TATP can involve multiple *test sessions*, each one defined in a separate Test Definition file (TDF). A session is automatically given a unique ID in TIRDB and it pertains to a defined population size. A session can be composed of one or more *test runs* defined in the same TDF. The test runs can vary in terms of user loads, test timing, and transaction mixes. Each test run is also given a unique ID.

You can specify an SQL script to be executed before a session, or SQL commands to be interleaving with test runs. You can also insert sleep intervals between the test runs.

The basic configuration can be expanded to

- include more than one client process per node (with any number of threads)
- have more than one target database on one or more computer nodes
- partition the database across several target databases
- populate the target database incrementally (say, from a size of 1 M subscribers to 2 M subscribers)
- execute multithreaded population (for larger database sizes)
- in solidDB setups, use directly linked solidDB drivers to bypass driver managers

3 Running the Benchmark

This section describes how to execute a TATP benchmark. The distribution package includes executables for a specific platform. The source code is also included, and can be compiled using the corresponding makefile(s) (Linux and UNIX) and VC++ project files (Windows).

3.1 Steps of Execution

The execution steps are divided into two groups: the preparation steps and the test session steps.

3.1.1 TATP Suite Preparation Steps

1. Dedicate the computers to run the test: the SUT computer and one or more TATP Benchmark Suite client computer(s). The computers should be interconnected with a fast Ethernet over a noncongested (or isolated) Ethernet segment.

2. Set up TIRDB under any SQL system offering ODBC interfaces.

3. Load the TIRDB schema and initialize the data pertaining to the test environment (see the subsection *Initial Data*).

4. Set up an ODBC data source name in the Main Node computer, for connection with TIRDB (alternatively, make the solidDB ODBC drivers available for direct linking).

5. Install TATP Benchmark Suite on the Main Node computer (unzip the installation package in any location). To run TATP in the Main Node, three binary executables are available in the bin/ directory of the installation directory: 'tatp', 'statistics' and 'client'.

6. Install TATP Benchmark Suite on the Remote Node computer(s) (at least the executables 'tatp' and 'client' are needed).

7. Set the settings in the *tatp.ini* file in the Main Node computer.

3.1.2 Test Session Steps

1. Install and configure the target database system at the SUT computer.

2. Install the corresponding ODBC driver on Main Node and Remote Node(s).

3. Set up an ODBC data source name in both Main Node and Remote Node(s) for connection with the target database system (if a driver manager is used).

4. Set up the database directives in the *.ddf* file in Main Node.

5. Set up the Remote Node definitions in remote nodes definition file in the Main Node.

6. If needed: set up SQL commands in the target database initialization file in Main Node (if the file is defined).

7. If needed: modify DDL SQL commands in the target database schema file in Main Node (if the file is defined).

8. Set up the test directives in the *.tdf* file in Main Node.

9. If the TATP Remote Control in Remote Node(s) are not already running, start it with `tatp -r` command.

10. Start the TATP Benchmark Suite with the `tatp` command in the Main Node.

When the intended sessions have been run, retrieve the result data from TIRDB. See also the *Execution workflow* subsection below for more information about running tests.

Note: To optimize the test runs (that is, save clock time) the following decomposition of the step 10 above could be considered:

10.1 Run `tatp` to populate the database without logging or with relaxed logging.

10.2 Make a backup of the database (for example, by copying the files to another location).

10.3 Iterate the two previous steps for each population needed.

10.4 Restore a necessary database and run `tatp` to run the test.

3.2 File Locations

The media delivered with the TATP package consists of seven major file groups, available in separate directories:

- Executables and configuration files (*bin/*)
  -> all the binary executable files needed to run TATP; example configuration files
  - Microsoft Windows: *tatp.exe*, *statistics.exe*, *client.exe*
    Linux: *tatp*, *statistics*, *client*
  - Additionally, examples of the following input files: TATP initialization file *tatp.ini,* a remote nodes definition file *remoteNodes.ini*, transaction files for solidDB and MySQL, a target database initialization file *targetDBInit.sql* and a target database schema file *targetDBSchema.sql*
    Note: bin/ is the working directory in the examples shown in this guide.
- Database definition files (*ddf/*)
  -> an example database definition file
- Test definition files (*tdf/*)
  -> an example test definition file
- Test Input and Result Database (*tirdb/*)
  -> TIRDB schema definition file and initial data population example
- Documentation (*doc/*)
- Source code (*source/*)
  -> C source code of TATP programs, and the related Makefiles.

3.3 TIRDB Initial Data

To be able to run the TATP benchmark, you need to have certain information about the SUT stored in the Test Input and Result Database (TIRDB). This requirement reduces the possibility of user-generated errors while writing test description files. The following data describing the SUT can be stored in TIRDB (see section TIRDB for the full TIRDB schema).

- Hardware
  -> The following information about SUT: processor type, number of processors, BIOS version, bus type, memory type, amount of memory, swap file size, disk type and number of disks. **The hardware ID for each distinct SUT equipment is mandatory**.
- Operating system
  -> Detailed information about the operating system of SUT. **The OS name and version are mandatory.**

- Target databases
  -> Identity information and configuration data about the target databases. **Each database product name and version has to exist in TIRDB** before this specific product version is tested.

The delivery package includes an example SQL script file containing the above information. You can modify the data in the file to correspond to your environment. The file is in *tirdb/initDataExample.sq*l

Note: If TIRDB is not used (the TIRDB connect string is not specified in 'tatp.ini'), the checking of the SUT description data is not done. If TIRDB is not used, some of the test result data (like MQTh and summary response times) are written to the result files and the console.

Note: TATP can be forced to read the SUT description data from the DDF file. This is done with a command line option "`-a`".

3.4 Command Line Syntax

The TATP benchmark is run from a command prompt. The name of the program is `tatp`. The program can be run in two modes, the Main Control mode and the Remote Control mode.

`tatp` can be run inside a single directory that contains all the files (default) or separating the TATP binaries and the other parts. The working directory for the configuration files and TATP run log files can be set using a command line parameter.

There are two mandatory command line parameters when running `tatp` in the Main Control mode:

- Database Definition File (.ddf) (see *Input files* / *Database definition files* for details about the content of a DDF file)

- Test Definition File (.tdf) (see *Input files* / *Test definition files* for details about the content of a TDF file).

Synopsis:

tatp [–c *directory*] [-i *inifilename*] *[other options*] *ddf tdf [tdf…*]

Description:

-c *directory*	Sets the working directory to the directory path given as an argument. The working directory is expected to contain all the input files that are needed to run TATP benchmark (see *Input files* for details). The TATP log files are also collected in the working directory. The default is the directory containing the executable TATP binaries.
-i *filename*	Sets the INI file name to the name given as an argument (default: 'tatp.ini')

Other options:

-a	Extracts the values of the hardware ID, operating system name & version, databases name & version, and inserts them into TIRDB.
-h	Prints usage help
-r	Runs as Remote Control (a TATP instance in a Remote Node)

-s	Shows detailed statistics after the TATP run, including the amounts of executed/rollbacked/ignored transactions for each client. The parameter is propagated to Remote Controls from the Main Control.
-t	Turns on TPS table processing. TPS is a table that shows the *tps* data (transactions per second) for each client connection in real time during the test run. It is mainly used in demonstrations that utilize performance visualizations. The frequency of the TPS table updates is 1 second by default. It can be changed with the directive "throughput_resolution", in the TDF file.
-v	The command line option –v may be used to control the verbosity of the program.

Possible verbosity levels are

-v0	No messages at all.
-v1	Only fatal error messages
-v2	Fatal and normal error messages.
-v3	Fatal and normal error messages and warnings.
-v4	All messages (including informative messages). This is the default value.
-v5	All messages + debug messages

Examples:

A simple command line:

```
tatp ../ddf/soliddb61.ddf ../tdf/soliddb100k.tdf
../tdf/soliddb1M.tdf.
```

The following command starts the benchmark in the mode where only fatal messages are printed on the console.

```
tatp –v1 ../ddf/soliddb61.ddf ../tdf/soliddb1M.tdf
```

3.4.2 Remote Control mode

To start TATP in the Remote Control mode, the `tatp` command is issued with the `–r` command line option.

The command line options -v, -i and -c can be used in the Remote Control mode.

For example, the following command starts the Remote Control in a mode where all debug messages are printed both in the console and the log files. This is likely to produce a lot of messages and should only be used for software debugging purposes.

```
tatp –v5 -r
```

Note: All the Remote Nodes included in a TATP run must have a Remote Control running before the test is started with the `tatp` command in the Main Node.

3.5 Execution Workflow

The flow chart in Fig. 3 depicts the execution stages in running the TATP benchmark.

Note: If the TATP Benchmark Suite is run without the Test Input and Result Database (TIRDB), the sampling timeline data collected during the run will be lost. However, the result value of MQTh, in tps, will be shown in the console output and in the TATP log.

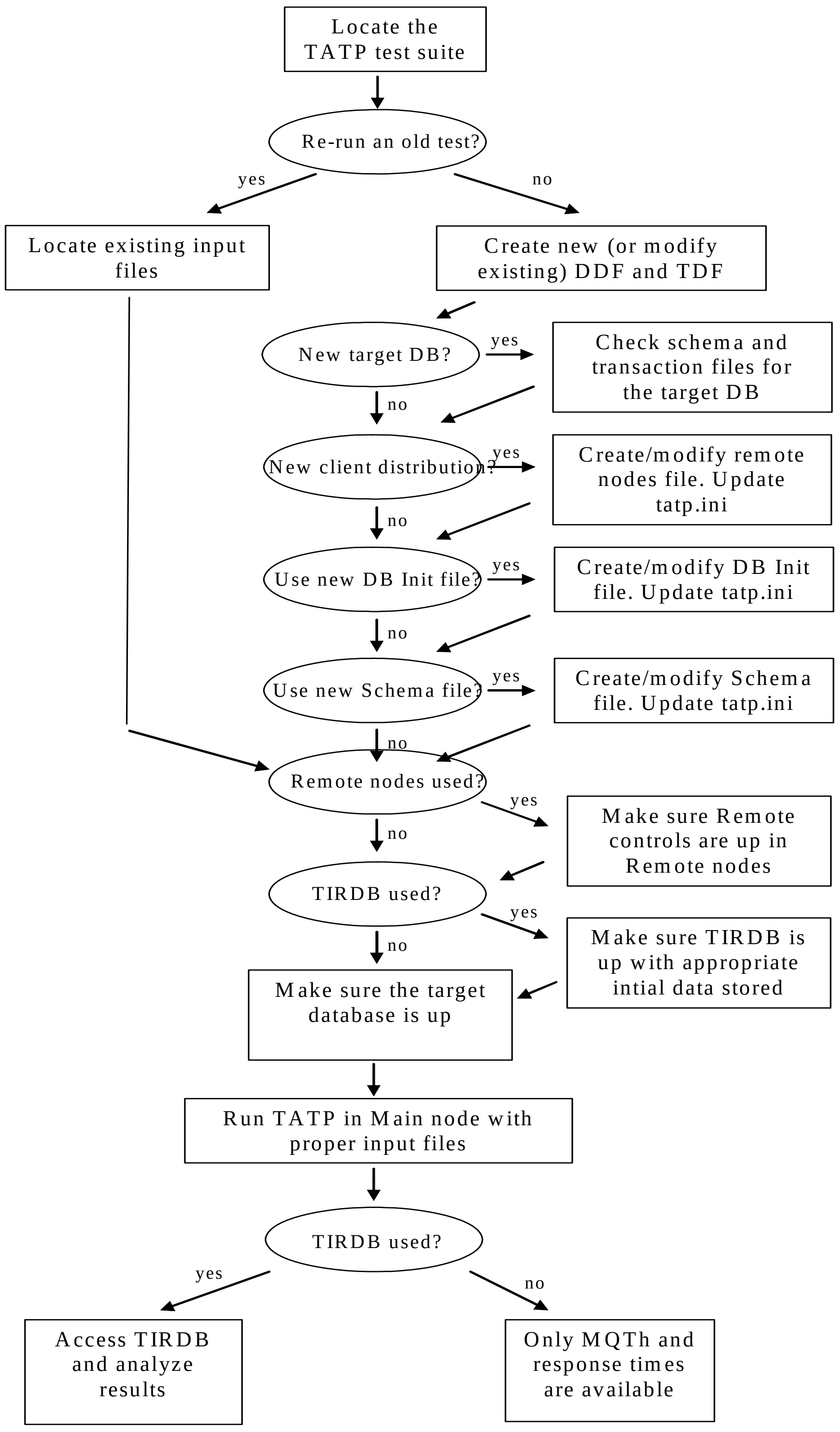
Locate the TATP test suite
Re-run an old test?
yes
no
Locate existing input files
Create new (or modify existing) DDF and TDF
New target DB?
yes
Check schema and transaction files for the target DB
no
New client distribution?
yes
Create/modify remote nodes file. Update tatp.ini
no
Use new DB Init file?
yes
Create/modify DB Init file. Update tatp.ini
no
Use new Schema file?
yes
Create/modify Schema file. Update tatp.ini
no
Remote nodes used?
yes
Make sure Remote controls are up in Remote nodes
no
TIRDB used?
yes
Make sure TIRDB is up with appropriate intial data stored
no
Make sure the target database is up
Run TATP in Main node with proper input files
TIRDB used?
yes
no
Access TIRDB and analyze results
Only MQTh and response times are available

Fig. 3. Execution Workflow of TATP.

4 Input files

The functionality of TATP is controlled by several input files. The following sections desribe the input files, including examples.

Note:

- By convention, all textual (non-numeric) values in the input files are to be entered in a quoted form (like: "text"). Numeric values are entered without quotes.
- In TATP configuration files, the symbol "//" in the beginning of a line means that the line is interpreted as a comment. In the SQL script files, both "//" and "--" (standard SQL comment marker) are accepted.

4.1 INI file

The *tatp.ini* (default name that can be changed with the `–i` option) file contains the initializing data for the TATP Benchmark Suite. The file has to be located in the working directory of the Main Node. If the working directory is not explicitly set using the `–c` option, the tatp.ini has to be located in the same directory with the TATP Benchmark Suite executables.

tatp.ini is a pure ASCII text file. The first line is the identification line contains the string "//tatp_ini". Subsequent lines contain the directives, typically keywords and data. The directives are mandatory unless otherwise mentioned. Comments can be added when preceded with characters '//'. The keywords are described below:

TIRDBConnect (optional)

ODBC connect string (or Data Source Name) to TIRDB. If this one is left out, TIRDB is not used and the benchmark results are not collected (only printed in the console output and in the TATP log).

If a driver manager is used, the format of the connect string is, for example:
```
TIRDBConnect = "DSN=TIRDB-S1;UID=dba;PWD=dba"
```
where "TIRDB-S1" is a data source name (DSN) defined in the driver manager. The keywords UID and PWD represent the username and password.

Another allowed format is
```
TIRDBConnect = "TIRDB-S"
```
In this case, the TIRDB credentials used (username, password) are "dba","dba".

solidDB note: If TATP is directly linked to the solidDB driver, TIRDB has to operate under solidDB too. In that case, an example of the TIRDB connect string follows the solidDB connect string format and is, for example:
```
TIRDBConnect ="tcp s1.acme.com 1314"
```
With directly linked solidDB driver, the TIRDB credentials (user name and password) are fixed to be "dba" and "dba", respectively.

In Windows environments, if you do not want to use a database, you can use a data source defined with the Microsoft Text Driver. In that case, there must be a properly configured text file for each table defined in the TIRDB SQL schema.

Log

The main text log file written by the TATP program. Log records are always appended to the existing log file. The log files are written to the program working directory. All the log files of the TATP test clients are collected in the *logs/* directory under the working directory in the Main Node.

RemoteNodes (optional)

The remote nodes definition file. If TATP is run with distributed clients, the information about all the Remote Nodes must be available in the remote nodes definition file. The information for each Remote Node includes an arbitrary name (referenced to from a TDF file), an IP address of the Remote Node, and the ODBC connect string for the target database defined in the Remote Node. If this directive is not used, the only valid client computer name used in the TDF file is "localhost".

SynchThreshold (optional)

The synchronization threshold for the TATP clients given in milliseconds. The elapsed test time running in each client (whether in the Main Node or any of the Remote Nodes) has to be within SynchThreshold milliseconds. If this is not the case, the test is not started and you are informed with messages in the log. Default is 10 milliseconds.

WaitDatabaseStart (optional, solidDB-specific, default -1)

When using an accelerator version of TATP (a build that is linked with the solidDB library called LLA--Linked Library Access a.k.a Accelerator), the WaitDatabaseStart parameter can be used to set the method that is used when making the client module to wait for the target database to be available. The value of the parameter will be propagated from Main Control to the Remote Controls.

The default value is -1, which means that the Main Control and Remote Control will continuously poll the target database until it responds.
If WaitDatabaseStart is set to 0, Main Control and Remote Control will wait until the user presses Enter.
If WaitDatabaseStart is set to any positive value X, the Main Control will wait X seconds before it polls the database and continues with the test.

An example of the *tatp.ini* file:.

```
//tatp_ini
TIRDBConnect = "DSN=tirdb2;UID=dba;PWD=dba"
Log = "tatp.log"
RemoteNodes = "remoteNodes.ini"
SynchThreshold = 15
```

4.2 Database Definition Files

A Database Definition File (DDF) defines the target database (identification and configuration details), the hardware of SUT, and the operating system of SUT. The DDF file is an ASCII text file. If TIRDB is defined, the data is verified against the corresponding data stored in TIRDB before the benchmark execution is started. If the data in the file does not match the data found in TIRDB, an error is generated in the log file and the program is stopped.

The first line of the DDF file has to be the identification line containing the string "//tatp_ddf". Subsequent lines contain directives, typically keywords with data. The directives are mandatory unless otherwise mentioned. Comments can be added when preceded with characters "//". Suggested file name extension for a DDF file is 'ddf'.

The recognized keywords are:

TargetDBInit (optional)

Target database initialization file. If the directive is defined, the SQL commands placed in the initialization file are executed against the target database prior any other action is taken against the target database (that is, before the benchmark is started) in a single invocation of TATP. The first line of this file has to start with "//tatp_sql". For more details on the database initialization file, see the section "Target Database Initialization Files" below.

ConnectionInit (optional)
Session initialization file. This SQL script is executed each time a new connection is opened to the target database, prior to any other action. It can be used to set session specific attributes (like isolation level, etc..) at run time. The first line of this file has to start with "//tatp_sql".

TargetDBSchema
The target database schema file. The SQL commands placed in the schema file are executed against the target database when the ‘populate’ directive is encountered in a TDF file. Typically, the SQL commands clean up the pre-existing schema and create a new TATP schema to be populated by the ‘populate’ directive. Various products vary in data type support and thus the target schema may need some modifications. The target schema in 'targetDBSchema.sql' will work with most products. The package contains also other target database schemas for most popular database products. The first line of this file has to start with "//tatp_sql". For more details on the target database schema file, see the section "Target Database Schema Files" below.

solidDB-specific note: With solidDB, the schema file 'targetDBSchema.sql' is used.

transaction_file
The target DBMS transaction specification file. Here, the transactions of the transaction mix are defined. The transaction files are database product specific because of the product-specific error codes that are allowed by TATP. For the description of the transaction definition syntax, see the section "Transaction Files" below. The transaction files for the most popular database products are supplied with the TATP Benchmark Suite package.

db_name
Name of the target database.
Note: The value must match a value in TIRDB, unless the command line option "-a" is used.

db_version
Version of the target database.
Note: The value must match a value in TIRDB, unless the command line option "-a" is used.

solidDB-specific note: If solidDB is the target database system, the value can be also left empty - it will be retrieved from the driver.

db_connect
The ODBC connect string (or Data Source Name) to the target database.

If a driver manager is used, the format of the connect string is, for example:
```
db_connect = "DSN=TARGET_DB;UID=dba;PWD=dba"
```
where "TIRDB-S1" is a data source name defined in the driver manager. The keywords UID and PWD represent the username and password, respectively.

Another allowed format is
```
TIRDBConnect = "TARGET_DB"
```
In this case, the TIRDB credentials used (username, password) are "dba","dba".

solidDB-specific note: If TATP is directly linked to the solidDB driver, the connect string

follows the solidDB connect string format and is, for example:
`db_connect ="tcp s2.acme.com 1964"`
With directly linked solidDB driver, the database credentials (username and password) are fixed to be "dba" and "dba", respectively.

os_name
Name of the operating system of the Server where the target database is running in.
Note: The value must match a value in TIRDB unless the command line option "-a" is used.

os_version
Version of the operating system of the Server where the target database is running in.
Note: The value must match a value in TIRDB unless the command line option "-a" is used.

hardware_id
Identification of the hardware configuration of the Server where the target database is running in.
Note: The value must match a value in TIRDB unless the command line option "-a" is used.

configuration_code
A name for the target database configuration used. The system calculates a unique identifier (checksum) for the content of the related configuration file. If the user given name - checksum pair does not exist in TIRDB, the new configuration (= name, checksum, the file content and potential comments) is stored to TIRDB.

configuration_file
A path to the database configuration file of the target database.

configuration_comments (optional)
Additional comments related to the configuration settings used in the target database.

An example of a DDF file follows.

```
//tatp_ddf
db_name = "solidDB"
db_version = "6.5"
db_connect = "DSN=solid65;UID=dba;PWD=dba"
os_name = "MS Windows"
os_version = "XP"
hardware_id = "cs1_wcd"
configuration_file = "../server65/solid.ini"
configuration_code = "default_config"
configuration_comments = "Default configuration"
ConnectionInit = "setSessionState.sql"
TargetDBSchema = "targetDBSchema.sql"
transaction_file = "tr_mix_solid.sql"
```

The parameter 'db_version* can be omitted if the target database system is solidDB. The version value will be extracted automatically.

4.3 Test Definition Files

A Test Definition File (TDF) represents a test session and defines one or several TATP test runs. A TDF file is an ASCII text file. The first line of a TDF file has to be the identification line containing the string "//tatp_tdf". Subsequent lines contain directives, typically keywords with data. The directives are mandatory unless otherwise mentioned. Comments can be added when preceded with characters "//". The suggested file name extension for a TDF file is 'tdf'.

A TDF file is divided to sections, each of them starting with a section marker. A section contains only directives dedicated to that section (the section directives in sequel). The sections are: ‘Session parameters’, ‘Population parameters’, ‘Test parameters’, ‘Transaction mixes’, ‘Database client distributions’ and ‘Test sequence’. The following sections are mandatory: ‘Session parameters’, ‘Transaction mixes’, ‘Database client distributions’ and ‘Test sequence’.

The structure of the file is:

```
//tatp_tdf
[Session parameters]
…
…
[Population parameters]
…
…
[Test parameters]
…
…
[Transaction mixes]
…
…
[Database client distributions]
…
…
[Test sequence]
…
…
```

Some of the section directives are mandatory and the rest is optional. If an optional section parameter is left out, it assumes its default value. Most of the section directives can be given directly (as command parameters) in ‘populate’ and ‘run’ commands in the ‘Test sequence’ section. In this case the value given directly overrules other definitions (in other sections) and default values for the same entity.

All the section directives are listed per section next. A section directive is mandatory unless marked "optional". The default values are given when appropriate.

4.3.1 [Session parameters]

This is a mandatory section in a TDF file.

- **session_name** (optional)
  A user-defined name for the benchmark session. A benchmark session can contain several benchmark runs. The session_name should illustrate the nature of the test run sequence, for example: "ULS 80/20, 1-128 clients".
  Note: ULS means User Load Scalability. It is a test sequence where the number of client threads is varied.

- **author** (optional)
  The name of the author executing the benchmark.

- **throughput_resolution** (optional, default 1 second)
  Throughput measurement (MQTh) resolution in seconds. The transactions are counted during the specified time step and the number of executed transactions is stored with the specified resolution.

- **comments** (optional)
  Free text containing the comments for the benchmark session.

4.3.2 [Population parameters]

This is an optional section in a TDF file.

- **subscribers** (optional, default 100 000)
  The number of subscribers populated in the target database. It corresponds to the cardinality of the SUBCRIBER table.
  Note: The TATP schema consists of four tables. The number of rows populated in the other three tables depends on the 'subscribers' row count.

- **serial_keys** ("yes"/"no", optional, default "no")
  A flag indicating whether the subscriber table is populated in random order or in serial order based on the key value of the table.
  "no" indicates random order and "yes" serial order. In disk-based databases, it is faster to populate the database with serial keys.

- **commit_block_rows** (optional, default 2000)
  Defines the number of inserts for the table 'subscribers' in one transaction in the population of the benchmark tables. If zero ('0') is given, auto-commit mode will be switched on, resulting to that every insert statement will be committed separately.

- **post_population_delay** (optional, default 10)
  Defines a pause (in minutes) after the population phase and before the start of a benchmark. This allows the DBMS to complete any asynchronous tasks before the benchmark is started.

- **check_targetdb** (optional, default "no")
  Defines whether the target database should be checked before the benchmark run. If set to "yes", the TATP schema will be validated and the amount of subscriber rows will be checked before conditional 'populate' command and 'run' commands.

4.3.3 [Test parameters]

This is an optional section in a TDF file.

- **warm_up_duration** (optional, default 10)
  Defines the warm up duration in minutes before the actual benchmark duration begins.

- **run_duration** (optional, default 20)
  Defines the benchmark duration in minutes.

- **uniform** ("yes"/"no", optional, default "no")
  Defines whether uniform subscriber ID key generation is used. Uniform ("yes) means that during the load execution, each value of subscriber ID search key will be generated with equal probability. In the non-uniform distribution ("no"), discrete hotspots are introduced in the search key generation. Non-uniform distribution is preferable because it reflects the reality in a better way.

4.3.4 [Transaction mixes]

This is a mandatory section in a TDF file.

One or several transaction mixes are defined in this section. Each transaction mix is given a unique name (within the section) and the transactions with their probabilities are listed within brackets. The syntax of a transaction mix is as follows:

transaction_mix ::=
  mix_name = {
  TRANSACT1 prob1
  [*TRANSACT2 prob2*
  ...]
  }

TRANSACTn ::= *name of transaction definition in SQL-file*

probn ::= integer, probability for the transaction,
  (all the probabilities must sum up to 100)

Example:

The definition of the standard 80/20 read/write mix is as follows:

```
tatp_mix_8020 = {
   GET_SUBSCRIBER_DATA 35
   GET_NEW_DESTINATION 10
   GET_ACCESS_DATA 35
   UPDATE_SUBSCRIBER_DATA 2
   UPDATE_LOCATION 14
   INSERT_CALL_FORWARDING 2
   DELETE_CALL_FORWARDING 2
}
```

4.3.5 [Database client distributions]

This is a mandatory section in a TDF file.

One or several database client distributions ("client loads") are defined in this section. Each distribution is given a unique name (within the section). Computers running TATP clients with the client counts are listed within braces. The word 'localhost' is a reserved word to denote the computer running the Main Control of TATP (Main Node). Other names must be defined in the Remote Nodes definition file. The syntax of a database client distribution is as follows:

database_client_distribution ::=
  distribution_name = {
  CLIENT_COMPUTER1 [num_of_processes1/]*num_of_clients1 [start_of_s_id_range1 end_of_s_id_range1]*
  [CLIENT_COMPUTER2 [num_of_processes2/]*num_of_clients2 [start_of_s_id_range2 end_of_s_id_range2]*
  ...]
  }

CLIENT_COMPUTERn ::= *name of a computer defined in the Remote Nodes file OR "localhost" to indicate the computer running the Main Control of TATP*

num_of_processesn ::= number of client processes run on a node. Default is 1

num_of_clientsn ::= integer, number of clients (threads) to be run in a single client process.

start_of_s_id_rangen::= the first value of the s_id value range (in the SUBSCRIBER table) for this client
  computer
  (optional)

end_of_s_id_rangen::= the last value of the s_id value range (in the SUBSCRIBER table) for this client computer
(optional, required if start_of_s_id_range is used)

Example 1: Single local (that is, located in the Main Node) multithreaded process:

```
[Database client distributions]
local_32cli = {
   localhost 32
}
```

Example 2: Two remote nodes, in addition to the main node, single multithreaded process per node:

```
[Database client distributions]
main_plus_two_remote_nodes_32cli = {
   localhost 8
   node2 12
   node3 12
}
```

Note: The names of the client computers are checked against the names defined in the remote nodes file 'remoteNodes.ini'. If a client computer name is not defined in the remote nodes file, TATP returns an error.

In addition to the number of threads, the number of processes can be specified too.

Example 3: To switch from the thread-based parallelism to process-based parallelism, use;

```
[Database client distributions]
main_plus_two_remote_nodes_32cli_processes = {
   localhost 8/1
   node2 12/1
   node3 12/1
```

The above configuration will use single-threaded client processes.

The optional S_ID range specification allows to distribute the load over a partitioned database, with each client generating load for a particular partition. This feature allows to run tests over a multi-front-end solidDB Cache configurations.

Example 4: 4 partitions over a 1 M subscriber database:

```
[Database client distributions]
over_four_partitions_64cli = {
   localhost 16 1 250000
   node2 16 250001 500000
   node3 16 500001 750000
   node4 16 750001 1000000
```

4.3.6 [Test sequence]

This is a mandatory section in a TDF file. It contains the execution directives, that is, it tells what steps will be executed in the session. There may be consecutive test run steps, or population steps intertwined with test run steps. Arbitrary SQL scripts and statements can be executed between the steps.

The test sequence section has the following directives: *populate*, *run, execute*, and *sleep*. The number or the order of the directives is not limited (for example, a TDF may have only a populate directive in its ‘Test sequence’ or it can have several benchmark run directives with one populate directive).

- **populate**
  Populates the target database with specified number of subscribers. The parameters are optional. The values override the values specified in the section "Population parameters". The directive can be furnished with a parameter “conditional” that instructs for the population to process only if the target database does not have the number of subscribers specified.
  Note: The value of the ‘check_targetdb’ in the section “Population parameters” must be set to “yes” in in order to activate conditional operation.

  Multithreaded population is possible with the help of the optional parameter "database_client_distribution". The value (distribution name) given is one of the distribution names defined in the same TDF.

  The syntax of “populate” is as follows:

  ```
  <populate> ::= populate [subscribers = <rows>] [serial_keys = <yes/no>]
          [commit_block_rows = <rows>] [post_population_delay = <duration>]
          [database_client_distribution=< cli_distr >]
          { [conditional] |
          [incrementally min_subscriber_id= <id_value>] }

  <rows> ::= integer

  <yes/no> ::= “yes” || “no”

  <duration> ::= integer

  <id_value> ::= integer

  <cli_distr> ::= database client distribution name in quotes. The distribution has to be defined in the
          [Database client distribution] section
  ```

  Example 1: Basic conditional population:

  ```
  populate conditional
  ```

  Example 2: Multithreaded conditional population:

  ```
  populate conditional database_client_distribution="local_32cli"
  ```

  Example 3: Incremental population (from 500001 to 1000000):

  ```
  populate subscribers=1000000 incrementally
  min_subscriber_id=500001
  ```

- **run**
  Defines a benchmark run with several transactions to be executed in parallel with specified probabilities.

  The syntax of “run” is as follows:

```
<run> ::= run transaction_mix = <tr_mix>
        database_client_distribution = <cli_distr> [name = <name>]
        [warm_up_duration = <duration>] [run_duration = <duration>]
        [repeats=<count>]
```

<name> ::= "string"

<tr_mix> ::= transaction mix name in quotes. The mix has to be defined in the [Transaction mixes] section

<cli_distr> ::= database client distribution name in quotes. The distribution has to be defined in the [Database client distribution] section

<duration> ::= integer (minutes)

<count> ::= integer (number of repeats)

Example: Basic test run:

```
 run name="1M 16 cli R80/W20"
    database_client_distribution="local_32cli"
    transaction_mix="tatp_mix_8020"
```

- **execute file**
  Specifies an SQL script to be executed at this point.

  The syntax of "execute file" is as follows:

```
<execute file> ::=
        execute file = "<path to the SQL script file>"
```

The output of the script execution is discarded.

Example:

```
execute file="reset_script.sql"
```

- **execute SQL**
  Specifies an SQL statement to be executed at this point.

  The syntax of "execute SQL" is as follows:

```
<execute> ::=
        execute file = "< SQL statement>"
```

The statement has to be terminated with a semicolon (;). The output of the statement execution is discarded.

Example:

```
execute SQL="DELETE FROM TPS;"
```

- **sleep**
  Sleeps for the given amount of seconds.

  The syntax of "sleep" is as follows:

```
<sleep> ::=
        sleep duration = <duration>

<duration> ::= integer (seconds)
```

Example:

```
sleep duration=10
```

The directives defined in the sections “Population parameters” and “Test parameters” apply to all the “populate” and “run” directives, respectively, in this section, unless overruled by the directive itself.

4.3.7 TDF example

An example of a Test Definition File follows. It defines a target database population and four benchmark runs.

```
//tatp_tdf

[Session parameters]
session_name = "TATP example session"
author = "John Smith"
throughput_resolution = 1  // seconds
comments = ""

[Population parameters]
subscribers = 1000000
serial_keys = “yes”
commit_block_rows = 2000
post_population_delay = 1 // delay after population in minutes
check_targetdb = "yes"

[Test parameters]
warm_up_duration = 10  // minutes
run_duration = 20  // minutes

[Transaction mixes]
tatp_mix_8020 = {
   GET_SUBSCRIBER_DATA 35
   GET_NEW_DESTINATION 10
   GET_ACCESS_DATA 35
   UPDATE_SUBSCRIBER_DATA 2
   UPDATE_LOCATION 14
   INSERT_CALL_FORWARDING 2
   DELETE_CALL_FORWARDING 2
}


[Database client distributions]
// single process per node
local_dcd_1 = {
   localhost 1 // number of database clients
}
local_dcd_4 = {
   localhost 4 // number of database clients
}

local_dcd_16 = {
   localhost 16 // number of database clients
}
```

```
local_dcd_64 = {
   localhost 64 // number of database clients
}
3_nodes_dcd_64 = {
   localhost 20 // number of database clients
   Merkurius 22 // number of database clients
   Venus 22 // number of database clients
}
}

[Test sequence]
populate conditional

run name="1M 1 cli R80/W20"
   database_client_distribution="local_dcd_1"
   transaction_mix="tatp_mix_8020"

sleep duration=10

run name="1M 4 cli R80/W20"
   database_client_distribution="local_dcd_4"
   transaction_mix="tatp_mix_8020"
sleep duration=10
run name="1M 16 cli R80/W20"
   database_client_distribution="local_dcd_16"
   transaction_mix="tatp_mix_8020"
sleep duration=10
run name = "1M 64 cli 3 nodes R80/W20"
   database_client_distribution = "3_nodes_dcd_64"
   transaction_mix = "tatp_mix_8020"
```

4.4 Remote Nodes Definition File

Remote Nodes definition file (typically 'remoteNodes.ini') contains information about the Remote Nodes (computers) that can act as hosts for Remote Controls (and clients run in Remote Nodes). The information for each Remote Node includes:

- a unique identifier
- the IP address or the DNS hostname
- an ODBC connect string to the benchmarked database

The unique name is used as the reference name to the Remote Node in question. The name is used in a TDF file when defining the database client distribution for a benchmark. An ODBC connect string is provided for the Remote Clients in the Remote Node to connect to the benchmarked database.

Note: The Remote Node must be able to recognize the given ODBC DSN, not the Main Node.

A remote nodes definition file is an ASCII text file. The first line of a file has to be an identification line containing the string "//tatp_remotenodes". Subsequent lines contain remote node definitions. Comments can be added when preceded with characters "//".

An example of a RemoteNodes file follows.

```
//tatp_remoteNodes
Merkurius = "192.168.0.100" "DSN=ServSol;UID=dba;PWD=dba"
Venus = "192.168.0.101 "DSN=ServSol;UID=dba;PWD=dba"
```

In the example file above, there are two possible Remote Nodes defined (Merkurius and Venus) that can be referenced to from a TDF file. The target database connect information will depend on the 'ServSol' data source definition in each Remote Node.

4.5 Target Database Initialization Files

The target database initialization file defines SQL commands that are executed against the target database prior any other action is taken against the target database (= before the first test session is started). The commands are executed one by one in the order they are listed in the file. If the target database returns an error while executing an SQL command, the error is logged and the execution is continued with the next SQL command (the program execution is not terminated).

A target database initialization file is an ASCII text file. The first line of a file has to be an identification line containing the string "//tatp_sql". Subsequent lines contain ';' terminated SQL commands. Comments may be added when preceded with characters "//".

A simple example of a Target Database Initialization file follows.

```
//tatp_sql
call init_tatp_run();
```

4.6 Target Database Schema Files

The target database schema file defines SQL commands that are executed against the target database when a 'populate' directive is encountered in a TDF file. Typically, the file contains a sequence of SQL commands that clean up the pre-existing schema and then create a new TATP schema to be populated by the 'populate' directive. The commands are executed one by one in the order they are listed in the file. If the target database returns an error while executing an SQL command, the error is logged and the execution is continued with the next SQL command (the program execution is not terminated).

A target database schema file is an ASCII text file. The first line of a file has to be an identification line containing the string "//tatp_sql". Subsequent lines contain ';' terminated SQL commands. Comments can be added when preceded with characters "//".

An example of a Target Database Schema file follows (the file defines a valid TATP schema).

```
//tatp_sql
// A valid tatp schema definition follows

DROP TABLE call_forwarding;
DROP TABLE special_facility;
DROP TABLE access_info;
DROP TABLE subscriber;

CREATE TABLE subscriber (s_id INTEGER NOT NULL PRIMARY KEY,
   sub_nbr VARCHAR(15) NOT NULL UNIQUE, bit_1 TINYINT,
   bit_2 TINYINT, bit_3 TINYINT, bit_4 TINYINT, bit_5 TINYINT,
   bit_6 TINYINT, bit_7 TINYINT, bit_8 TINYINT, bit_9 TINYINT,
   bit_10 TINYINT, hex_1 TINYINT, hex_2 TINYINT, hex_3 TINYINT,
   hex_4 TINYINT, hex_5 TINYINT, hex_6 TINYINT, hex_7 TINYINT,
   hex_8 TINYINT, hex_9 TINYINT, hex_10 TINYINT, byte2_1
   SMALLINT, byte2_2 SMALLINT, byte2_3 SMALLINT, byte2_4
   SMALLINT, byte2_5 SMALLINT, byte2_6 SMALLINT, byte2_7
   SMALLINT, byte2_8 SMALLINT, byte2_9 SMALLINT, byte2_10
   SMALLINT, msc_location INTEGER, vlr_location INTEGER);

CREATE TABLE access_info
   (s_id INTEGER NOT NULL, ai_type TINYINT NOT NULL,
      data1 SMALLINT, data2 SMALLINT, data3 CHAR(3), data4 CHAR(5),
      PRIMARY KEY (s_id, ai_type), FOREIGN KEY (s_id) REFERENCES
      subscriber (s_id));

   CREATE TABLE special_facility
      (s_id INTEGER NOT NULL, sf_type TINYINT NOT NULL, is_active
      TINYINT NOT NULL, error_cntrl SMALLINT, data_a SMALLINT,
      data_b CHAR(5), PRIMARY KEY (s_id, sf_type), FOREIGN KEY
      (s_id) REFERENCES subscriber (s_id));

   CREATE TABLE call_forwarding
      (s_id INTEGER NOT NULL, sf_type TINYINT NOT NULL, start_time
      TINYINT NOT NULL, end_time TINYINT, numberx VARCHAR(15),
      PRIMARY KEY (s_id, sf_type, start_time), FOREIGN KEY (s_id,
      sf_type) REFERENCES special_facility(s_id, sf_type));
```

The schema defined in the schema definition file is checked for validity before it is fed to the target database. For TATP to work, mandatory tables and columns with right column types have to be defined. Additional columns and tables can exist. If the schema is not supplied (the directive targetDBSchema is not present in *tatp.ini*), the existing schema is validated in the target database.

4.7 Transaction Files

A transaction file comprises the Telecommunication Application Transaction Processing (TATP) benchmark transactions to be executed. It is advised to have one transaction file per a target database product tested. By doing this, it is easy to take into account possible differences in the SQL dialects of the target databases. In addition, so called "acceptable" error codes returned by the target database are defined in this file for each transaction. These error codes are likely to differ among various database products.

The transactions in a transaction file are written in SQL .The defined transactions are referenced by name in the TDF file. The keyword *transaction_file* in the DDF file identifies the path to the file where the transactions are defined.

A transaction file is an ASCII text file. The first line of a file has to be an identification line containing the string "//tatp_transaction". Subsequent lines contain ';' terminated SQL commands encapsulated inside named transactions. A single transaction can have several SQL commands. Comments can be added when preceded with characters "//".

TATP generates a transaction load on the target database based on the transaction definitions found in the transaction file. A special syntax is adopted to enable so-called placeholders within an SQL statement. These placeholders can act as parameters carrying values from one SQL clause to another (within a transaction) or a placeholder can refer to a hard-coded method, for example, to get a random value for a parameter of an SQL command. The placeholders are encapsulated within brackets (<,>) (less-than and greater-than characters used in SQL must be quoted). A placeholder has the following format inside an SQL command (the reader is expected to be familiar with the basics of the SQL command syntax thus the surrounding SQL command syntax is omitted here):

transaction_placeholder ::=
    start_tag value_type operation [*variable [column]*] *stop_tag*

start_tag ::= <

stop_tag ::= >

```
value_type ::= msc_location | vlr_location | s_id | bit | is_active | hex | byte |
        data1 | data2 | data_a | error_cntrl | start_time | end_time | end_time_add |
        ai_type | sf_type | sub_nbr | numberx | dtaa3 | data4 | data_b
```

operation ::= rnd | rndstr | value | bind

variable ::= *a variable name, the scope is the transaction in question*

column ::= *a column name*

A set of value types are reserved to cover mandatory TATP table column types. The following table lists the types and their value ranges (listed also in the syntax definition above).

Placeholder value type	**Actual value type and range**
msc_location vlr_location	IINTEGER [1,2^32-1]
s_id	INTEGER [1, p] where P=number of subscribers
bit	0 or 1
is_active	0 (15% probability) or 1 (85% probability). Biased random number.
hex	INTEGER [0,15]
byte data1 data2 data_a error_cntrl	INTEGER [0,255]
start_time	0, 8 or 16
end_time	INTEGER [0,24]
end_time_add	INTEGER [1,8]
ai_type sf_type	INTEGER [1,4]
sub_nbr numberx	a character string representation of an integer from [1,P] where P=number of subscribers. The length is 15 characters, the rest of the characters set to "0" (for example, "000000000073425")
data3	random characters from A-Z, length 3 characters
data4 data_b	random characters from A-Z, length 5 characters

The operations that are required to provide values for TATP table columns based on well-defined cardinalities and distributions are implemented in the TATP Benchmark Suite. The following table lists the available operation (types) and their descriptions.

Placeholders operation type	**Description**

`Rnd [`*`variable`*`]`
 	
`rndstr [`*`variable`*`]`
	To create a random value of the *value type* given. *`rnd`* is used for scalar types and *`rndstr`* for characters and character strings. If there is a *variable* defined, the value is saved under this name and it can be used within the same transaction (using the *value* method).

`Value` *`variable`*	To get a saved value from a *variable* and to use it as a bounded parameter in a SQL clause. *Value type* indicates the type of the value. When *value* is used *variable* has to be defined in the placeholder clause.
`Bind` *`variable column_name`*	Can be used in a SELECT clause. It binds a return value of the column and saves it under a specified variable. When *bind* is used both the *variable* and the *column* have to be defined in the placeholder clause (see the syntax before and the following examples).

An example of a simple transaction follows (this is an arbitrary example and not any of the seven TATP transactions).

```
GET_SUBSCRIBER_MSC_LOCATION {
   SELECT msc_location
   FROM subscriber
   WHERE s_id = <s_id rnd>;
}
```

In this example a named transaction GET_SUBSCRIBER_MSC_LOCATION has been defined – it can be referenced from a TDF file. One simple SELECT command is defined which projects the msc_location column of a random subscriber from the subscriber table. The search condition is defined with s_id = <s_id rnd> where the *rnd* operation is invoked to return an *s_id* type value.

An example of one transaction using transaction variables follows (this is a valid TATP transaction).

```
UPDATE_SUBSCRIBER_DATA {
   UPDATE subscriber
   SET bit_1 = <bit rnd>
   WHERE s_id = <s_id rnd father>;

   UPDATE special_facility
   SET data_a = <data_a rnd>
   WHERE s_id = <s_id value father>
         AND sf_type = <sf_type rnd>;
}
(ERRORS ALLOWED 13, 1017)
// 13 -> 'No data found', 1017 -> 'No rows affected'
```

In this example a named transaction UPDATE_SUBSCRIBER_DATA (referenced to from a TDF file) has been defined. It consists of two update commands. In the first update command a random bit value (0 or 1) is generated for *bit_1* (bit_1 = <bit rnd>). The value type definition (*bit*) tells the hard-coded random method that a random bit value is needed. The same principle applies to the random *s_id*, which is generated for the search condition and is saved in the variable *father* (s_id = <s_id rnd father>). The value of *father* is then used in the second update clause as part of the search condition (s_id = <s_id value father>).

The 'ERRORS ALLOWED' definition in the end of the transaction definition above states the database returned errors that are considered as allowed errors; they are not considered as error messages by the TATP benchmark software. In practice, TATP skips the allowed errors and continues the benchmark execution.

An example of a transaction using the SELECT result binding follows (this is a valid TATP transaction).

```
DELETE_CALL_FORWARDING {
   SELECT <s_id bind subid s_id>
   FROM subscriber
   WHERE sub_nbr = <sub_nbr rndstr>;

   DELETE
   FROM call_forwarding
   WHERE s_id = <s_id value subid>
         AND sf_type = <sf_type rnd>
              AND start_time = <start_time rnd>;
} (ERRORS ALLOWED 40001)
```

The SELECT command fetches one row from the subscriber table (sub_nbr is guaranteed to be unique). The *s_id* column is projected from the result row and the result value is bound to the variable subid. The value of the variable is used to control the rows deleted in the following DELETE command.
Note: Binding the result in a SELECT command requires that the SELECT command returns only one row. If the SELECT command returns multiple rows the behavior of the binding operation is undefined.

4.8 Allowed errors

The following are acceptable errors from the TATP perspective:

1. A UNIQUE constraint violation error encountered in the INSERT CALL FORWARDING transaction (an effort to insert a duplicate primary key).

2. A foreign key constraint violation error in the update and insert transactions, when an effort is made to insert a foreign key value that does no match a corresponding value in the referenced table.

5 Output files

5.1 Log Files

The log files of TATP can contain informative (I), warning (W), error (E), and fatal (F) messages. The files are ASCII text files and each line of a log file begins with the character I, W, E or F followed by a time stamp, message source and the message itself.

The severity level of the message types are given below.

- **I** stands for a message for information only,
- **W** means warnings that are useful to know but do not affect the reliability of test results,
- **E** is for error messages that cause the benchmark run to halt or the results to be found incorrect while the program execution may be continued,

- **F** is for fatal messages where the program execution cannot be continued.

The log files are located in a directory structure under the *TATPbinaries/logs/* directory so that all the log files of a session are located under a session directory. The name of the session directory is of the form *date_sessionID* (for example *TATPbinaries/logs/20080914_197*). The log files are further grouped in the session directory so that the log files from a computer participating in the TATP session (Main Node and Remote Nodes) are located in a directory named after the computer name (for Main Node, the name 'localhost' is used;for the Remote Nodes the host name given in the Remote Nodes definition file is used).

The following example shows the the log files directory structure after two TATP sessions:

```
"TATP installation directory"/
bin/
   logs/
      20080914_197/
         localhost/
            223_client1.log
            223_client2.log
            …
            223_client15.log
            tatp.log
            statistics.log
         Merkurius/
            223_client16.log
            …
            223_client30.log
            223_tatp.log
      20080915_198/
         localhost/
            tatp.log
            statistics.log
         Merkurius/
            224_client1.log
            …
            224_client20.log
            224_tatp.log
         Venus/
            224_client21.log
            …
            224_client40.log
            224_tatp.log
```

Note: In the example directory structure above, session 20080915_198 was run without clients running in the Main Node (localhost) so the only log files in the *localhost* directory are the main log and *statistics.log*.

Main Log

The Main log file contains log messages from the control module of TATP. The name of the Main log file is defined in *tatp.ini*.

Example of a typical Main log file after a successful TATP run:

```
I 2009-02-21 15:21:15 CONTROL Processing Data Definition File 'example.ddf'
I 2009-02-21 15:21:15 CONTROL Processing DB initialization file 'targetDBInit.sql'
I 2009-02-21 15:21:15 CONTROL Processing Test Definition File 'example.tdf'
I 2009-02-21 15:21:15 CONTROL Starting session number 63 'TATP example session'
I 2009-02-21 15:21:15 CONTROL Processing DB schema file 'targetDBSchema.sql'
I 2009-02-21 15:21:16 CONTROL Populating TATP with 100000 subscribers
I 2009-02-21 15:21:17 CONTROL Starting test run number 61 'TATP example run'
I 2009-02-21 15:22:20 CONTROL MQTh for test run is 2453
I 2009-02-21 15:22:20 CONTROL Finalizing test run number 61
I 2009-02-21 15:22:20 CONTROL Finalizing session number 63
```

```
I 2009-02-21 15:22:20 CONTROL Processing of TDF example.tdf completed
I 2009-02-21 15:22:20 CONTROL No errors
I 2009-02-21 15:22:20 CONTROL *** End ***
```

The Main log of the Main Node is the first thing to check after a TATP run. The end of the file indicates the number of execution errors, if any, during the run ('No errors' in our example above). The MQTh value for the test run is also given in the log.

The Main log of the Main Node is located in the appropriate *localhost* directory in the Log files directory structure. The Main logs of the Remote Nodes are located in the Remote Node specific directories.

statistics.log

The *statistics.log* log file collects the log messages from the *Statistics* module of the TATP software. The *Statistics* module communicates with the database clients and collects and computes the MQTh and response time results and finally stores them to TIRDB.

Example of a typical *statistics.log* after a successful TATP run:

```
I 2008-11-10 15:32:02 STATISTICS Started
I 2008-11-10 15:32:03 STATISTICS Rampup time of 30 minutes
I 2008-11-10 15:47:04 STATISTICS All clients finished
I 2008-11-10 15:47:04 STATISTICS Write results to TIRDB
```

statistics.log is located in the appropriate *localhost* directory in the Log files directory structure.

clientN.log

The *clientN.log* log file (for example, *client2.log*) collects the log messages from a specified database client (*client2.log* from the client number two).

Example of a typical *clientN.log* after a successful TATP run:

```
I 2008-11-10 15:32:04 CLIENT2 Feeding transactions for 7200 seconds
```

The client logs are located in the node specific directories. The log names are furnished with the test run identifier.

6 Test Input and Result Database (TIRDB)

Test Input and Result Database (TIRDB) is the data storage for pre-defined input data and the benchmark result data of TATP. The schema of TIRDB is shown in the figure below (Fig. 4).

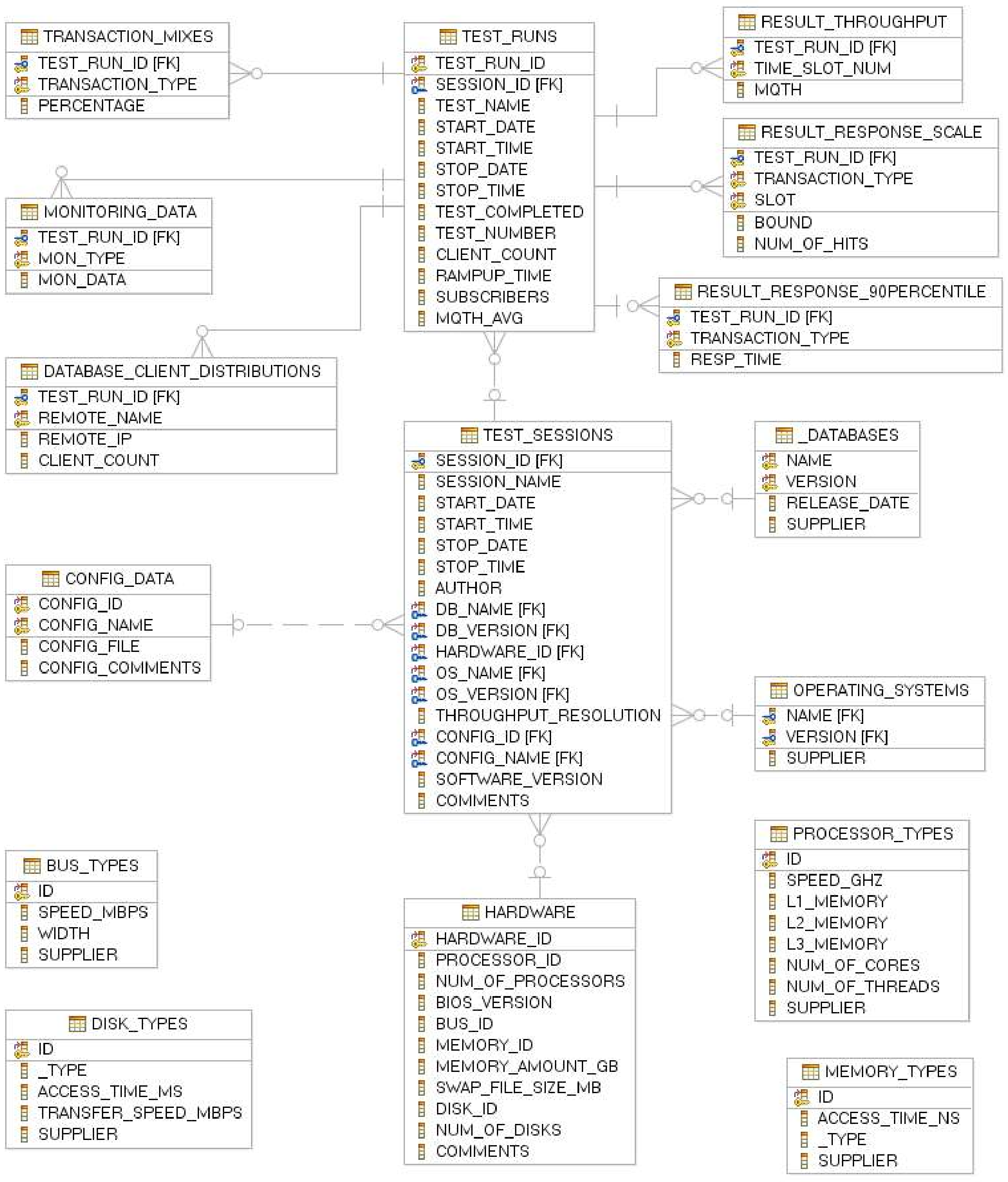

Fig. 4. Test Input and Result Database (TIRDB).

6.1 Pre-Defined Data

TIRDB includes a set of pre-defined input data that is fairly constant in nature and is fed into the system before the first TATP benchmark session. For each TATP session, the user has to input a certain number of parameters in the DDF and TDF files and, for most of these values, the corresponding value has to be present in TIRDB. The relevant data are shown below. The data required by TATP at run time are marked with "mandatory":

Information about the target database Server:

```
CREATE TABLE hardware (
   hardware_id CHAR(32) NOT NULL,--mandatory
   processor_id CHAR(32),
   num_of_processors INTEGER,
   bios_version CHAR(32),
   bus_id CHAR(32),
   memory_id CHAR(32),
   memory_amount_gb INTEGER,
   swap_file_size_mb INTEGER,
   disk_id CHAR(32),
   num_of_disks INTEGER,
   comments CHAR(255),
   PRIMARY KEY (hardware_id));

CREATE TABLE processor_types (
   id CHAR(32) NOT NULL,
   speed_ghz FLOAT,
   l1_memory CHAR(32),
   l2_memory CHAR(32),
   l3_memory CHAR(32),
   num_of_cores INTEGER,
   num_of_threads INTEGER,
   supplier CHAR(32),
   PRIMARY KEY (id));

CREATE TABLE memory_types (
   id CHAR(32) NOT NULL,
   access_time_ns FLOAT,
   _type CHAR(32),
   supplier CHAR(32),
   PRIMARY KEY (id));

CREATE TABLE disk_types (
   id CHAR(32) NOT NULL,
   _type CHAR(32),
   access_time_ms FLOAT,
   transfer_speed_mbps INTEGER,
   supplier CHAR(32),
   PRIMARY KEY (id));

CREATE TABLE bus_types (
   id CHAR(32) NOT NULL,
   speed_mbps INTEGER,
   width INTEGER,
   supplier CHAR(32),
   PRIMARY KEY (id));
```

Information about the operating system of the target database Server:

```
CREATE TABLE operating_systems (
   name CHAR(32) NOT NULL,    --mandatory
   version CHAR(32) NOT NULL, --mandatory
   supplier CHAR(32),
   PRIMARY KEY (name, version));
```

Information about the target databases:

```
CREATE TABLE _databases (
   name CHAR(32) NOT NULL,    --mandatory
   version CHAR(32) NOT NULL, --mandatory
   release_date DATE,
      supplier CHAR(32),
      PRIMARY KEY (name, version));
```

The delivery package includes an example SQL script file containing some example information. Change the data in the file to correspond to your environment. The file can be found in *tirdb/initDataExample.sql.*

6.2 Benchmark Input Data

The DDF and TDF files define one or several benchmark runs. Relevant data from these definitions is stored in TIRDB for later reference. The TIRDB tables that store this data are TEST_SESSION, TEST_RUNS, TRANSACTION_MIXES, DATABASE_CLIENT_DISTRIBUTIONS and CONFIG_DATA. Logically, a record in the table TEST_SESSIONS corresponds to a combination of a DDF and TDF files and a record in the table TEST_RUNS corresponds to a single benchmark run defined in a TDF file. The table CONFIG_DATA is used to store the configuration file content of the target database. The path to the configuration file with possible comments is given in the DDF file.

6.3 Benchmark Result Data

6.3.1 Throughput results

All the result data from a TATP benchmark run are stored in TIRDB (if TIRDB is specified in *tatp.ini*). This includes the Mean Qualified Throughput (transactions per second) and the response time data per transaction type.

The following sample query produces a full summary of all test runs, including the test time information, some of the test setting parameters, and the resulting MQTh values.

```
SELECT se.session_id, session_name, test_run_id, test_name,
te.start_date, te.start_time, te.stop_time, db_name, db_version,
config_name, subscribers, mqth_avg
   FROM test_sessions se JOIN test_runs te
   ON se.session_id = te.session_id
   ORDER BY se.session_id desc;
```

The query can be expanded with additional columns, and restricted by way of columns of the TEST_SESSIONS and TEST_RUNS tables.

The table RESULT_THROUGHPUT is used to store the MQTh timeline results with a given time resolution (for example, 1 second).

RESULT_THROUGHPUT.test_run_id is a foreign key referencing TEST_RUNS. This column associates the result records with a certain benchmark run. RESULT_THROUGH-PUT.time_slot_num (RT.tsn) identifies the time interval from the test execution time along with TEST_SESSIONS.throughput_resolution (TS.tr) in the following way:

The time interval for a record of RESULT_THROUGHPUT is
[RT.tsn*TS.tr, (RT.tsn+1)*TS.tr-1]

The throughput resolution, TS.tr, is given, optionally, in the TDF file. By default, the resolution is 1 second.

For example, the following query will produce the full time series of MQTh(t), for test run 111.

```
SELECT time_slot_num, mqth
   FROM result_throughput
   WHERE test_run_id=111;
```

6.3.2 Response time results

During the run, TATP collects the response time data in two ways: as distribution histograms and as 90-percentile response time calculated for the whole run.

90-percentile response times

90-percentile response time is a value (expressed in microseconds) that corresponds to a response time that 90% of transactions underpass.

For example, the following query will return 90-percentile response times for all transaction types, produced in test run 111:

```
SELECT transaction_type, resp_time
   FROM result_response_90percentile
   WHERE test_run_id = 111;
```

Response time distributions

Response time distribution is represented as logarithmic histograms, in table RESULT_RESPONSE_SCALE. The column.*test_run_id* is a foreign key referencing TEST_RUNS. This column associates the result records with a certain benchmark run. Response time is measured per transaction type, thus the column.*transaction_type* is part of the primary key of the table as well. The response time histograms are represented as series of slots, one slot per row. For each run and transaction type, a slot is identified with a slot number (*slot*), starting with 1. For each slot, TATP inserts an upper bound of the response time (*bound*) in microseconds. Finally,.*num_of_hits* records the number of transactions (of a certain type) having the response time within the bounds of this slot. The bounds of a slot having number i are:

$[\text{boundary}_{i-1}, \text{boundary}_i)$

where $\textit{boundary}_{i-1}$, $\textit{boundary}_i$ are the values in ‘boundary’ column in the rows identified by ‘slot’ = i and ‘slot’ = i – 1, respectively. The lowest boundary RTS.boundary_0 is always zero.

In general, the format of the RESULT_RESPONSE_SCALE table allows storing an arbitrary set of time slots.TATP uses a predefined set of slot with logarithmically increasing sizes. A decimal order of magnitude is divided into 7 logarithmic intervals with a base 1.39 with the following boundaries:

1.0; 1.39; 1.93; 2.68; 3.72; 5.17; 7.2; 10.0

Five decimal orders of magnitude are currently accounted, starting from 10 microseconds and with the upper limit of 1 second. An additional interval [0; 10 microseconds) is added so there are 36 intervals in total.

The benefit of a logarithmic set of histogram time slots is that it is possible to cover any large interval with a small number of slots with fixed boundaries. Costly estimation of histogram boundaries is not required during the test run time.

To retrieve the response time histogram for transaction type GET_ACCESS_DATA in test run 111, and in the time range below 1 millisecond, use the following query:

```
SELECT bound, num_of_hits
   FROM result_response_scale
   WHERE test_run_id=111
     AND transaction_type = 'GET_ACCESS_DATA'
     AND bound <= 10000;
```

Sample TIRDB query scripts are included in *tirdb/.*

6.3.3 Test run status and monitoring

When a benchmark run is finished successfully, the value of TEST_RUNS.test_completed bit is set to 1. Otherwise (in case of erroneous interruption) the bit is 0.

There is a special table MONITORING_DATA in TIRDB to store system monitoring related data of the benchmark run. It can be used, for example, to store relevant MMC (Microsoft Management Console) generated performance data in Windows environments.

7 References

[1] "Telecommunication Application Transaction Processing (TATP) Benchmark Description", IBM Corporation, 2009.

Appendix 1. TATP Benchmark Suite Internals

Main node and remote nodes

The run time components of TATP in a configuration with multiple client nodes are shown in Fig. A- 1. If there are no remote nodes, the components of the Main Node are present.

There are three executable program images used in each execution. There are called 'tatp', statistics' and 'client'. You start a TATP execution by invoking the 'tatp' program at all the nodes. The program runs the functions of Main Control or Remote Control, depending on the command line options. Main Control starts the Statistics and Client processes. Remote Control starts the Client processes only. The Clients from both Main Node and Remote Nodes communicate the test results to the Statistics process (in the Main Node). The Client process in the Main Node and the Remote Nodes are identical.

Multiple clients run inside a single process, utilizing a separate thread for each Client connection. A process-based configuration with multiple single-threaded Client processes is also possible.

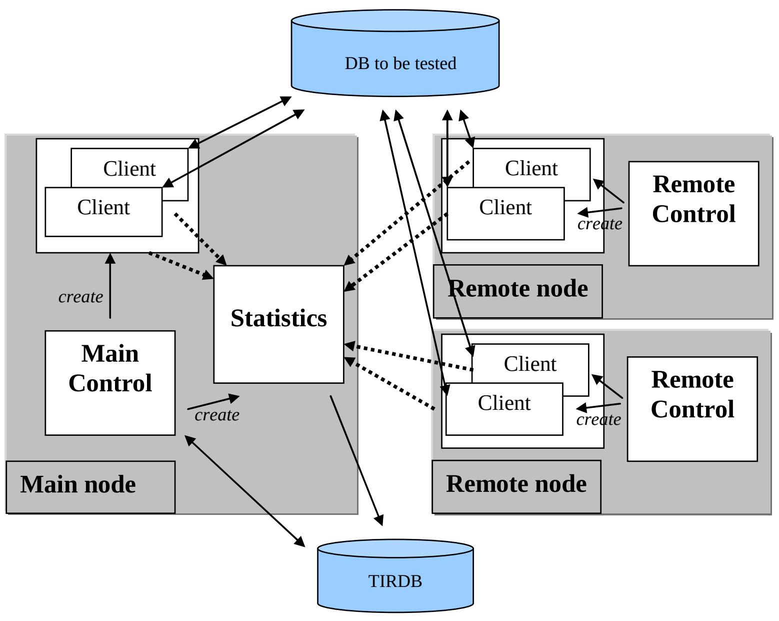

Fig. A- 1. Overall Architecture of the TATP Benchmark Suite with remote nodes..

The Main Control component controls the execution of TATP by reading all the input data and starting the benchmark (main control) and counting and storing the benchmark result (Statistics). When the Main Control is started, the Remote Control processes in Remote Nodes have to be running (in fact, Remote Controls can run constantly and need not be stopped between sessions). Fig. A- 2 depicts the process level architecture of the Main Node.

Operation of the Main Node

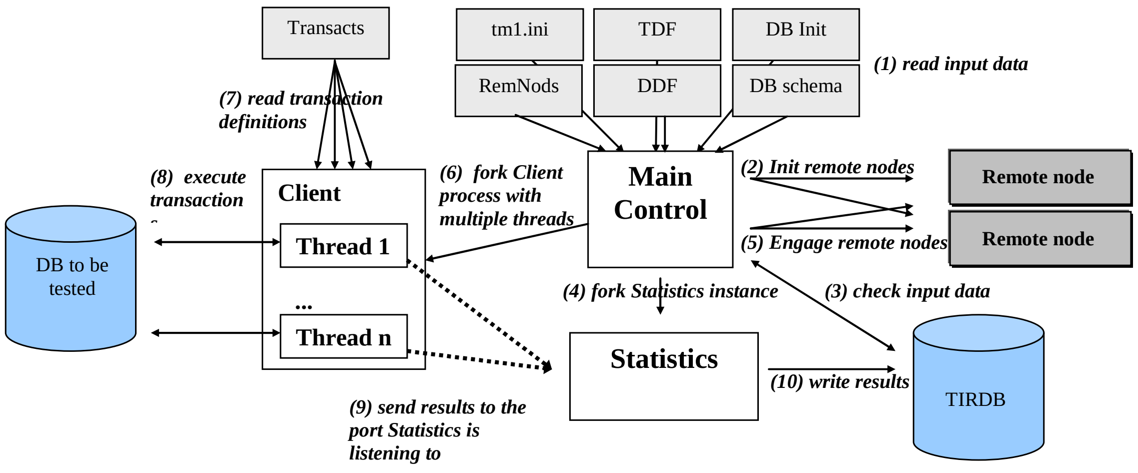

Fig. A- 2. Process level architecture of Main Node.

The runtime configuration of the Main Node consists of three processes, namely, *Main Control*, *Statistics* and *Client*. *Client* process starts multiple threads, the number of which in the Main Node is defined per test run in the TDF file (see Section *Input Files / Test Definition Files* for details). The main responsibilities of the Main Node processes are as follows (see the figure above for the numbers in parentheses):

- ***Main Control*** (1) reads the input files, (2) initializes the Remote Control processes (in Remote Nodes), (3) checks DDF input data against TIRDB, (4) starts the Statistics process, (5) asks the Remote Control processes to start the benchmark, and (6) creates the Main Node Client processes.

- ***Client*** (7) starts multiple threads, one thread per client connection as configured by Main Control. Each thread reads transaction definitions from a file, (8) executes the transactions (based on the transaction mix defined) against the target database, and (9) writes transaction execution information to *Statistic's* communication port .

- ***Statistics*** (9) collects the results from individual Clients from all the nodes (Main Node and Remote Nodes), calculates the overall results of the benchmark, and finally (10) writes the results to TIRDB.

Operation of the Remote Node

The Remote Node controls the execution of TATP clients by starting a Client according to the request received from the Main Control (from the Main Node). The Remote Control acts as the client dispatcher and it can run constantly waiting for the client start requests (however, only one benchmark running at a given time can be active). The input file handling is left out, and the input data needed by the Remote Control is communicated to it by the Main Control process. Fig. A- 3 depicts the process level architecture of the Remote Node.

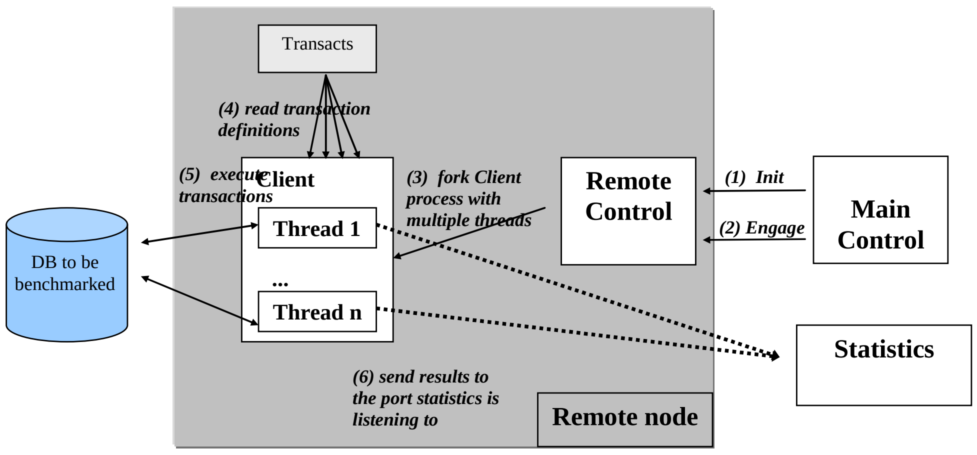

Fig. A- 3. Process level architecture of Remote Node.

The main responsibilities of the Remote Node processes are as follows (see the figure above for the numbers in parentheses):

- ***Main Control from the Main Node*** (1) initializes the Remote Control process and (2) asks the Remote Control processes to start the benchmark.
- ***Remote Control*** (3) creates the Remote Node Client processes.
- ***Client*** (4) starts multiple threads, one thread per client connection as configured by Main Control. Each thread reads transaction definitions from a file, (5) executes the transactions (based on the transaction mix defined) against the target database, and (6) writes transaction execution information to statistic's (in Main Node) communication port.

The various components of the TATP configuration can be run on different platforms (hardware/OS). Therefore, the type of the TATP Benchmark Suite Package(s) (for example, Windows or Linux) should be chosen depending on the planned test client environment. The target database system can be run on any platform offering ODBC-based remote access.