ArrayDB Beta User Manual

(Rev. 2.0, 4/14/2015)

Contents

Introduction

The goal of this user manual is to provide a guide for customers to know and understand Exeray's ArrayDB functionality and high performance.
ArrayDB leverages patented Indexing technology to outperform current MPP Databases in the market. This guide also provides instructions of benchmarktesting ArrayDB's performance in terms of data inserting, retrieving and concurrency.

Test Environment

ArrayDB consists both Server and Client sides. You may install Server and Client packages either on the same machine or on different servers.

System minimum requirements for Server Side

Hardware：Pentium III 450 above, 1024MB RAM, 200G HD
Software： Linux 2.6.23 CentOS, RedHat, Fedora, x86, x86_64
File systems： ext4, or XFS
Environments have been tested: Fedora 3.1.0 x86_64, 8GB RAM, 512GB HD, ext4

System minimum requirements for Client Side

Hardware：Pentium III 450 above, 512MB RAM, 100G HD
Software： Linux 2.6.23 CentOS, RedHat, Fedora, x86, x86_64
File systems：ext4, or XFS
Environments have been tested: Fedora 3.1.0 x86_64, 8GB RAM, 512GB HD,ext4

ArrayDB Installation

You can download all binary packages for 64bit machines of ArrayDB software from our Github account (https://github.com/exeray/arraydb) and then you can install ArrayDB in just one step. You may install Server, Client or Embedded binaries either in the same machine or different servers. ArrayDB Server will listen on TCP/IP port 8888, and the process name is "rdbserv". The process should be started by *r db* user. ArrayDB provides shell scripts rdbstart.sh and rdbstop.sh for you to start and stop the Server repsectively.

ArrayDB Server Setup

In the server machine, you can execute the following script using as *root* :

```
# tar zxf arraydbservern.n.tar.gz
```

Then related files will be unzipped under arraydbservern.n directory:

```
# cd arraydbservern.n
# ./install.sh
```

The installation shell script will generate *rdb* system user, copy config file *rdb.conf* to /etc/rdb.conf and copy rdbserv, rdbadmin, rdbstart.sh and rdbstop.sh to /usr/bin.

Create database and users

After you have set up ArrayDB, you may create database and users as *rdb* user by invoking the rdbadmin script:

Under *rdb* user, do

```
# su – rdb
$ rdbadmin createdb mydb
```

mydb is your database name.

Use *rdbadmin createuser* to create your database users:

```
$ rdbadmin createuser tom:tom123
```

Will create user *tom* with password as *tom123* . The default user priviledge is readonly.

You may grant more priviliegs by doing so:
$ rdbadmin perm tom:write

This will grant *read* and write privileges to user *tom*. The default *users* generated are general users who can not query and change database System Meta data. General users can only query and change database System Meta data by upgrading to Admin account. You can achieve this by doing:

$ rdbadmin role tom:admin
This command will upgrade the user *tom* to Admin level user.

And here is RDBADMIN man page:
$ rdbadmin
rdbadmin command [argument]
rdbadmin createdb <databasename>

createa new database
rdbadmin dropdb <databasename>
drop a database

rdbadmin createuser <username:password>
createa new database user with username and password

rdbadmin dropuser <username>
drop a database user
rdbadmin role <username:admin/user>
update role of user (admin or user)
rdbadmin perm <username:read/write>
update permission of user (read: read only, write: read and write)

ArrayDB Server Startup

After you have created database and users, you may change your identity to the *rdb* account to start ArrayDB server:

```
# su – rdb
$ /usr/bin/rdbstart.sh
```

Then ArrayDB server will listen on port 8888. After Server is started up, *rdb* may still create more Databases and User Accounts. Newly created Databases will take effect immediately while newly create Users will take about 10 minutes to be recognized by *rdbserv server process* . The Server log will be under $RDB_HOME/system/log/ and you may set the value of $RDB_HOME in /etc/rdb.conf.

ArrayDB Client Setup

To install client platforms, you can excute this command as *root* user：

```
# tar zxf arraydbclientn.n.tar.gz
```

Related files will be unzipped to arraydbclientn.n.

```
# cd arraydbclientn.n
# ./install.sh
```

The installation will also

- Copy config file *rdb.conf* to */etc/rdb.conf*
- Copy *rdbclient.h* which includes APIs to */usr/include*
- *Copy* librdbclient.a which includes API libraries to */usr/lib*
- Copy *rdb* programs to */usr/bin*

Config file /etc/rdb.conf includes the following parameters:

```
RDB_HOME = /home/rdb/database
port=8888
```

RDB_HOME is the root directory which stores all ArrayDB related files and only apply to the Server. 8888 is the port number which you can change as needed.

ArrayDB Embedded Database Setup

ArrayDB embedded database is a standalone data storage software. It is not related to ArrayDB client and server programs. It is meant to be used by software which needs to store and query data in a local device or host. ArrayDB embedded database will generate lower level data files directly. Insert and Select operations will be performed on such lower level data files without clientserver interaction.

In supported platforms, you can excute this as *root* user :
tar zxf arraydbembedn.n.tar.gz
Related files will be unzipped to directory arraydbembedn.n

cd arraydbembedn.n
./install.sh

The installation will also
- Copy ArrayDB.h and *rdb.conf* files to */usr/include*
- *Copy libarraydb.a* which includes API libraries to */usr/lib*

We also provide a C++ code example *rdbtest.cc* to show you how to generate Index, insert data and query data using ArrayDB C++ class. This example also behaves as a script to benchmark performance of ArrayDB server kernel.

Platform Provisioning

Mount noatime

File Input and Output (IO) is one of the most important performance indicator for Database. We suggest that you may close the *access time* option for your file system. You may disable this in */etc/fstab* as *root* :

defaults,noatime

Installation Verification

After you install the ArrayDB Server, please make sure：

1) No other service or progresses use Port 8888
2) User *rdb* root directory /home/rdb was created
3) /etc/rdb.conf RDB_HOME pointed to correct directory
4) Following files exist and can be executed:

    /usr/bin/rdbserv

    /usr/bin/rdbadmin

    /usr/bin/rdbstart.sh

    /usr/bin/rdbstop.sh

After you install the ArrayDB Client, please make sure that the following files exist:

    /usr/bin/rdb

    /usr/include/rdbclient.h

    /usr/lib/librdbclient.a

After you install ArrayDB Embeded, please make sure that the following files exist:

    /usr/include/ArrayDB.h

    /usr/include/abax.h

    /usr/include/AbaxFormat.h

    /usr/lib/libarraydb.a

Test Execution

Test Approaches

There are three ways to benchmarking test against ArrayDB:

1. Interaction between ArrayDB Client and Server side:

Test by operating as *rdb* client user, typing RQL (SQL standard query language). Then the Server will respond when receiving RQL.

2. APIs calls

Test by writing C programs which calls ArrayDB APIs to perform related data Select, Insert operations.

3. Operations on kernel data files

Test by using ArrayDB C class to perform related data Select, Insert operations on Server Kernel data files

Baseline Performance

Following benchmarks can demonstrate the performance advantages of ArrayDB:

- Data Load/Insert
- Date Query
- Indexing performance
- Memory usage
- Join performance
- Above performance in a concurrent way

Insert performance

There are 3 ways to test Insert performance:

1) Perform batch load on the Server side ;

2) Insert single record from Client side;

3) Perform Insert operation direcly on Kernel data files.

Please make sure Server, Client and Embed bins were correctly installed and excuted.

Preparations：

1. Create the user

su – rdb

$ rdbadmin createuser test:test

$ rdbadmin perm test:write

2. Create the table

$ rdb –u test –p test –d test –h 127.0.0.1:8888

rdb> create table test (key: uid char(16), value: addr char(16));

(A) Batch Load

You can test ArrayDB Server by loading 3 million records. And the sample 3 million records can be generated by program *genrand* which comes with Client bin.

$ genrand 3000000 1 uid addr

$ mv genrand.out /tmp/3M.txt

Then in *rdb* client side (any user can run *rdb* Client side) use the following command to load 3 million records to test table:

rdb> load file /tmp/3M.txt into test format S;

Expected behavior：After about 23 minutes, rdb will tell how long it takes to load data in milliseconds.

(B) Client single record Insert

rbench program in Client package will help insert, modify and query records on Server. The following command will generate 10000 numbers randomly and insert record to *test*

table in the Server.

$ rbench –t test –r “10000:0:0” –k 0 | tee –a test.log

In *“10000:0:0”* the first “10000” the times for Insertion, the second “0” is the times for Update and the third “0” is the times for Query. The option “k 0” means clear Table *test* and regenerate table.

Expected behavior: rbench will insert data about 6000-8000 record per second including network overhead.

(C) Insertion on Kernel data files
Apply rdbtest in Embed package and perform direct insertion on Kernel data files:

$ ./rdbtest 3000000 | tee –a test.log
The result will tell the Insertion speed:
Successfully insert() 3000000 records in nnnn milliseconds IOPS=38235
Expected behavior: rdbtest performance will be around 30000-60000 records per second.

Data Query performance

（A） rbench Client query performance
$ rbench–t test –r “0:0:10000” –k 0 | tee –a test.log
This will query server 10000 times
Expected behavior: the performance will be around 4000-8000 records per second.

（B） rdbtest test Server kernel query performance
$ rdbtest 3000000 | tee –a test.log
exist(): if the record exists
getValue(): get Value for each Key input

scanned: get all records

Expected behavior:
exist(): 500,000-700,000 per second
getValue() : 400,000-500,000 per second
Index Scan: 33,500,000-36,800,000 per second

Index performance

In *rdbtest* embedded database testing, exist() and getValue() are done by hashing. And Scan operation is done by index scan. The value of IOPS is operation per second. Query by Hash Index is about 500,000-700,000 per second while query by scanning Index is about 33,500,000-36,800,000 per second.

Memory usage

During all operations, you may use *top* to monitor memory usage for Server *rdbserv* process.

$ top

PID USER PR NI VIRT **RES** SHR S %CPU %MEM TIME+ COMMAND
12360 rdb 20 0 108m **26m** 904 S 99.4 0.3 0:09.02 rdbserv
You may watch the RES value which is the actual memory usage for the process.
You may also user *rdbtest* for Embedded Database usage.

$ rdbtest 1000000
Memory used: 14 MB
ArrayDB will use much less memory, thus improving overall performance of Database.

Join Performance

You may follow these steps to test table Join performance:

1) Use *genrand* generate some files containing 1 million records each.

```
$ genrand 10000000 1
$ mv genrand.out /tmp/10M1.txt
$ genrand 10000000 1
$ mv genrand.out /tmp/10M2.txt
$ genrand 10000000 1
$ mv genrand.out /tmp/10M3.txt
```

2) Create tables

```
rdb> create table t1 ( key: uid char(16), value: addr char(16) );
rdb> create table t2 ( key: uid char(16), value: phone char(16) );
rdb> create table t3 ( key: uid char(16), value: email char(16) );
```

3) Load Data

```
rdb> load file /tmp/10M1.txt into t1 format S;
rdb> load file /tmp/10M2.txt into t2 format S;
rdb> load file /tmp/10M1.txt into t3 format S;
```

Each Load will need 23 minutes.

4) Join

```
rdb> select * join ( t1, t2, t3 ) limit 10;
```

This will Join 3 tables by joined key and you may also monitor memory usage by using *top* command.

Expected behavior: 3 million records tables will be joined and the result will be returned in 0.9 seconds.

Concurrentrequest performance

You may use *rbench* to simulate concurrent connections and requests. The concurrent testing will include following:

1) 100% users data insert concurrently
2) 100% users data select concurrently
3) 20% users data insert and 80% users data select concurrently

100% users data insert concurrently

$ rbench–t test –r "10000:0:0" -k 0 –c 4 | tee –a test.log

"c 4" means 4 threads, and every thread will request 10000 insert operations from Server. Actual concurrent threads number should align with the number of Server Cores: for example if the Server had 16 cores, the concurrent threads should be set to 16 to make sure the best performance of Insertion.

100% users data query concurrently

$ rbench–t test –r "0:0:10000"–c 4 | tee –a test.log

"c 4" means 4 threads, and every thread will request 10000 select operations from Server. Actual concurrent threads number should align with the number of Server Cores: for example if the Server had 16 cores, the concurrent threads should be set to 16 to make sure the best performance of Selection.

20% users data insert and 80% users data select concurrently

$ rbench–t test –r "2000:0:8000"–c 4 | tee –a test.log

"c 4" means 4 threads, and every thread will request 8000 select operations and 2000 insert operation from Server. The concurrent threads number should be set the same as above.
You may also use *top* to monitor memory usage here as well .
Expected behavior: The performance will increase when you add more threads and will stop improvement when threads are more than the number of Server cores.
Note ：*rdbserv* locks in the file level and we will improve this in the future.

Baseline Functionality

Following features of Database can also be tested using above methodology:

1) create database
2) create user
3) create table
4) insert data into table
5) select data from table
6) update data in table
7) delete data from table
8) join tables and select data
9) load file into table
10) drop a table
11) drop database

The man page is here to help:

rdb> help

You can enter the following commands:

help use (how to use databases)
help desc (how to describe tables)
help show (how to show tables)
help create (how to create tables)
help insert (how to insert data)
help load (how to batch load data)
help select (how to select data)
help update (how to update data)
help delete (how to delete data)
help join (how to join two or more tables)
help drop (how to drop a table completely)

MySQL benchmark testing

MySQL Load Data

Generate data files before inserting data into Mysql database by using *genrand* .

```
$ genrand 10000000 1
$ mv genrand.out /tmp/10M_MYSQL.txt
```

Create table:

```
mysql> create table t1 ( uid varchar(16) primary key, addr varchar(16) );
```

You may load data by using:

```
mysql> load data infile '/tmp/10M_MYSQL.txt' into table t1 fields terminated
by ',' lines terminated by '\n';
```

You may also use *top* to observe mysqld memory usage.

MySQL Insert Data

Here is the C example to insert 10000 records into Mysql

```
mysql_real_connect(con, "localhost", username, passwd, "test", 0, NULL, 0);
for ( i = 0; i < 1000 ; ++i )
{
   k = randomString(16);
   v = randomString(16);
   sprintf( sqlcmd, "insert into %s values ('%s', '%s' ) ", table, k, v );
   mysql_query( g_mysqlcon, sqlcmd );
}
```

MySQL Select Data

Here is the C example to select 10000 records from Mysql

```
for ( i = 0; i < 10000; ++i )
{
   k = randomString( 16 );
   v = randomString( 16 );
   sprintf( sqlcmd, "select * from %s where uid='%s'", table, k );
   mysql_query( con, sqlcmd );
   result = mysql_store_result( con );
}
```

MySQL Join Tables

Load generated data files to 3 tables in Mysql

```
mysql> create table t1 ( uid varchar(16) primary key, addr varchar(16) );
mysql> create table t2 ( uid varchar(16) primary key, phone varchar(16) );
mysql> create table t3 ( uid varchar(16) primary key, email varchar(16) );
mysql> load data infile '/tmp/10M_MYSQL_1.txt' into table t1 fields terminated
```

by ',' lines terminated by '\n';
mysql> load data infile '/tmp/10M_MYSQL_2.txt' into table t2 fields terminated by ',' lines terminated by '\n';
mysql> load data infile '/tmp/10M_MYSQL_3.txt' into table t3 fields terminated by ',' lines terminated by '\n';

Load 1 million data in MySQL will take around 1-3 hours

Perform Join operation

mysql> select * from t1, t2, t3 where t1.uid = t2.uid and t2.uid = t3.uid;

MySQL will print the Time spent for the operation.