Manual for MySQL Target Algorithm Evaluator

Steve Ramage
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4, Canada
{seramage}@cs.ubc.ca

May 10, 2015

Contents

1 Introduction

The MySQL Target Algorithm Evaluator is an implementation of AEATK's `TargetAlgorithmEvaluator` interface, that supports distributed execution through the use of a centralized MySQL Database. This document is designed to read by two types of users:

1. Users who would like to use the workers as part of an existing AEATK application
2. Developers of AEATK applications who would like to make them compatible with the workers.

1.1 License

The MySQL Target Algorithm Evaluator has been released under the AGPLv3 license. Please contact Frank Hutter `hutter@cs.ubc.ca` to discuss other licensing opportunities.

2 Usage

2.1 Prerequisites

2.1.1 User Prerequisites

To use the MySQL Target Algorithm Evaluator it is expected that you are familiar with the following concepts:

1. You are familiar with the basics of the AEATK *wrapper* interface. Specifically you are aware that an algorithm will be invoked via the command line given an instance, cutoff time, seed and a parameter configuration, and the wrapper of the algorithm will return a status flag as well as measurements of runtime, runlength, quality, and some additional run data. This is key, since this project essentially allows for distributing the execution of these wrappers.

2. You know how to connect to a MySQL Server using the `mysql` command.

3. You can execute basic `SELECT` and `UPDATE` statements as necessary. A good tutorial on SQL is available here: http://www.w3schools.com/sql/sql_syntax.asp

4. You can create a database in MySQL.

5. It is expected that you read through this entire document (end-to-end). In 90%-95% of cases, the MySQL Target Algorithm Evaluator can be used without a second thought but you should be familiar with this document, as occasionally you will need to understand the architectural limitations.

2.1.2 System Prerequisites

It is assumed that you have access to a MySQL Server running version 5.5. Version 5.1 may or may not work. MariaDB and MySQL 5.6 should work but have never been tested. Additionally you must have access to a user and database that allows for `SELECT, INSERT, UPDATE, DELETE` as well as `CREATE TABLE` permissions.

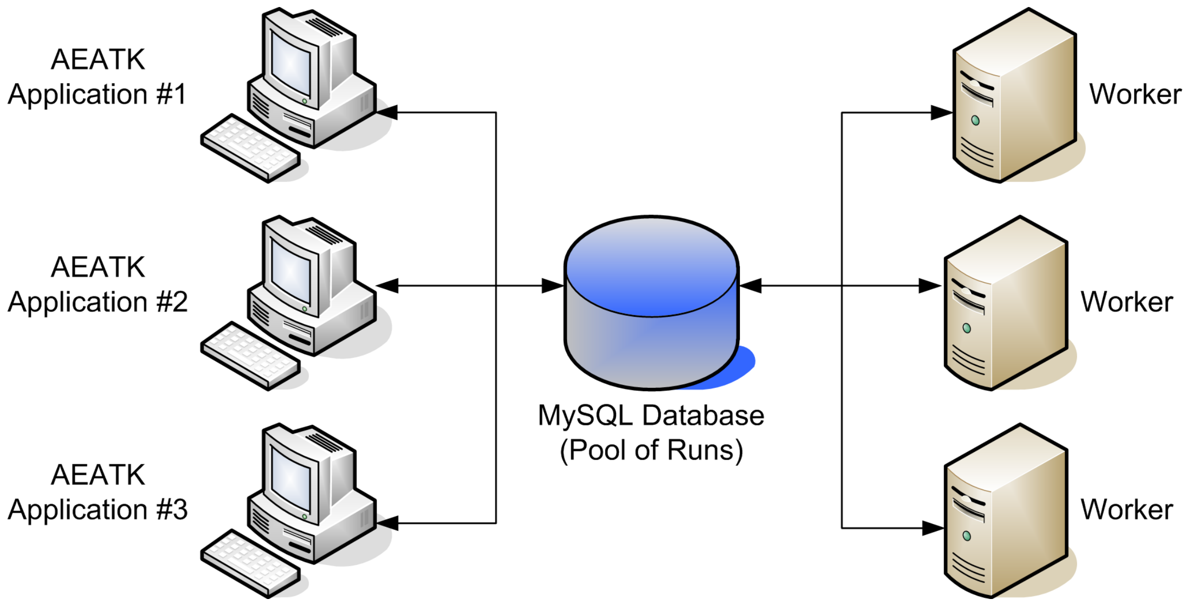


Figure 1: Architecture of the MySQL Target Algorithm Evaluator

Important!

Good performance of the MySQL Target Algorithm Evaluator depends *heavily* on the MySQL Server being tuned correctly. Unfortunately the default configuration of MySQL Server is incredibly inappropriate. More advanced users are encouraged to read Chapter 8 of *High Performance MySQL 3rd Edition*. Assuming that you are only interested in using MySQL for this application and are not running anything else, we would recommend the following changes (if you are using MySQL for something else, you will need to do your own research).
The three most important settings to change are:

- `innodb_buffer_pool_size` which should be set to 80% of the RAM you would like MySQL to use (we use 16 GB). For more information see:
  http://dev.mysql.com/doc/refman/5.5/en/innodb-parameters.html#sysvar_innodb_buffer_pool_size
- `innodb_log_file_size` which should be set to about 1 GB or higher. For more information see:
  http://dev.mysql.com/doc/refman/5.5/en/innodb-parameters.html#sysvar_innodb_log_file_size
- `max_allowed_packet` which should be set to 16 MB or higher (we set it to 64 MB). For more information see:
  http://dev.mysql.com/doc/refman/5.5/en/server-system-variables.html#sysvar_max_allowed_packet

Additionally Section 7 starting on page 25 has a sample configuration that we use.

2.2 Overview

Both users and developers should be aware of the basic architecture, which is pictured in Figure 1. In essence, all communication between the application and the worker is done through the database. In fact it is almost entirely *anonymous*. The database structure is designed to be simple and in fact is nothing more than a glorified list of runs to execute. The application is entirely unaware about which worker, if any, executed it's run, and the worker is unaware about which application requested the run or why. This anonymity has certain benefits like allowing for simple caching, as well as allowing users to manually intervene in the database.

2.2.1 Database Overview

Beyond a simple database (in the `CREATE DATABASE` meaning of the term), the MySQL Target Algorithm Evaluator also has a concept of pools. A pool is essentially a family of tables that serve as a light weight database. For instance you might create a pool for each job id on the cluster to manage experiments, or whatever else you'd like. Workers are assigned to a pool and will operate on all runs in the pool. When executing, all table names will be prefixed with the pool name, so for instance the `runs` table with pool `test` will actually be called `test_runs`.

For an exact detailed listing of the tables and columns consult the database. The meaning of individual columns are heavily based on AEATK concepts and largely just a one-to-one mapping to a field in AEATK. The following however is an overview of what is important.

Suggestion

Using PHPMyAdmin (www.phpmyadmin.net) for interacting with the database can make many things much easier. Ask your system administrator to install it for you.

commandTable This table simple lists all AEATK applications commands that were executed if available. Data is never read from this table, and mainly exists in case the command that generated the data has been forgotten.

version This table simply lists the version of the MySQL Target Algorithm Evaluator that created the table. The hash is actually a hash of the file that creates the tables. If the hash that exists in the database differs from the current one a warning will be logged. This might indicate that the structure has changed and is incompatible.

algoExecConfig This table contains information about executing the algorithm, it's location, it's configuration space, etc. It is inserted into once. All of the fields are informational and are just used to recreate the appropriate AEATK object in the worker. For AEATK users, it is the `AlgorithmExecutionConfiguration` object and the `ParameterConfigurationSpace` object that is stored here.

runs This table is where most of the magic happens[1]. The remaining information necessary to reconstruct an call to the wrapper is here. Other columns of note are:

[1]For database aficionados, this table is roughly a denormalized version of three sets of information, the information necessary to execute the run, the result of the run, and information controlling the run status. It has been denormalized for both performance and ease of use for users. This table is hit very heavily

Column	Description
runHashCode	The hash is a SHA1 hash of key information that identifies the run. The hash is only used by the AEATK Application portion. It is *approximately* (You should consult the code for the authoritative calculation) the instance name, seed, cutoff, parameter configuration, and the execConfig all concatenated as a string and then and hashed. Some other options also affect the hash, such as the runPartition (described below).
status	Controls the current status of a run (either `NEW`, `ASSIGNED`, or `COMPLETE`).
priority	Allows for rudimentary control of the priority of the run. In the AEATK application you can set the **--mysqldbtae-priority** option to control this, or manually alter it in the table. Higher priorities are selected first for processing.
killJob	Flag that signifies whether the application requested the run be terminated.
retryAttempts	Stores the number of times a worker was assigned the run since the last time it was requested.
runPartition	A unique integer that allows for different AEATK applications to interact with the same pool *without* using the same cache. Each one can be assigned a different **--mysqldbtae-run-partition** which will cause different hashes to be generated.
worstCaseEndtime	An approximate, pessimistic and informational-only estimate of the latest time a run could complete.
worstCaseNextUpdate_WhenAssigned	If the run is `ASSIGNED` and this time is in the past, then we expect that the run will not be completed, and may be reclaimed.

Important!
Executing queries against this table while the workers are running needs to be done with care as you may inadvertently stall the workers, especially when this table contains millions of records.

workers This table controls the status of the workers, and allows for online management of the workers. Many of the fields are strictly informational including the `hostname`, `jobID`, `status`, `version`, `startTime`, `startWeekYear`, `orginalEndTime`. Additionally other fields contain the suffix `_UPDATEABLE` this means that if you change this value AND change the value of `upToDate` to 0, the worker will re-read this value and change it's behaviour.

Many of these columns values can be initially set on the command line of the worker via the appropriate argument.

Some columns of note are (Note we have removed the `_UPDATEABLE` suffix here for space reasons):

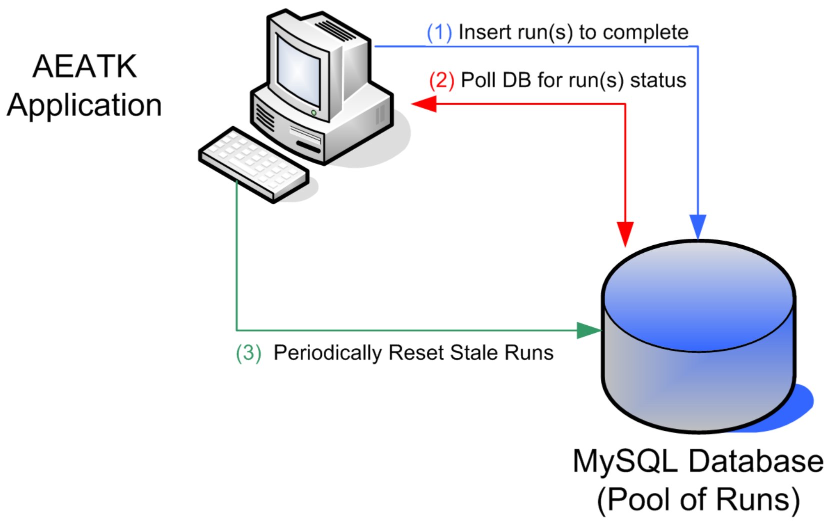

AEATK Application
(1) Insert run(s) to complete
(2) Poll DB for run(s) status
(3) Periodically Reset Stale Runs
MySQL Database (Pool of Runs)

Figure 2: Master - Database Interactions

Column	Description
autoAdjustBatchSize	If true then the worker will slowly increase the number of runs it requests (the `runsToBatch` column) from the database in batches if it is idle, if it can't get as many runs as it requests it will request fewer runs. The minimum and maximum size are controlled by the other columns.
delayBetweenRequests	This is loosely how often the workers will poll for new runs, if all runs the worker was assigned finished before this, it will wait until this amount of time expires before continuing.
poolIdleTimeLimit	If the sum of all the `workerIdleTime` values within the past few `startWeekYears` exceeds this value, and this worker becomes idle, it will terminate to avoid wasting cluster time.
concurrencyFactor	This is the number of workers that can simultaneously hold a lock on the `runs` table. Setting this too high can result in performance problems.
pool	If changed the workers will restart on another pool, if that pool doesn't exist the worker MAY die depending on other options.
worstCaseNextUpdate_ WhenRunning	If this value is ever in the past and the worker has status `RUNNING`, then we assume the worker has died.

2.2.2 AEATK Application

Because neither the workers nor the application are aware of each other, it helps to think about their responsibilities / processes separately. Figure 2 contains a diagram of the responsibilities and interactions of the application and the database.

1. The runs are inserted into the database. The MySQL Target Algorithm Evaluator uses hashing to reuse runs. If a run already exists it is reused as is. At a certain points a run may be interrupted or killed, if the run was killed or the wrapper signalled an **ABORT**, the cached run will NOT be used, and instead it will be run again.

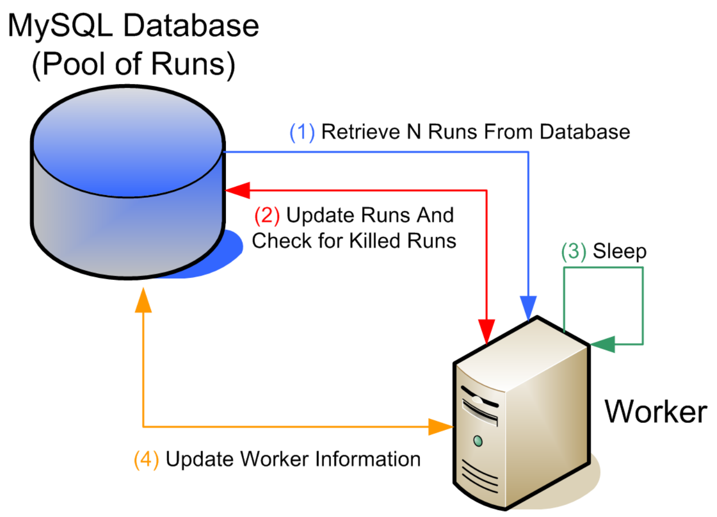

MySQL Database
(Pool of Runs)
(1) Retrieve N Runs From Database
(2) Update Runs And Check for Killed Runs
(3) Sleep
Worker
(4) Update Worker Information

Figure 3: Worker - Database Interactions

2. Periodically the database is queried to see if the runs were completed (i.e., **status** is `COMPLETED`, if so they are returned.

3. Periodically the application checks the `runs` and `workers` tables for columns that have not been updated by the correct time (given by the corresponding `worstCaseNextUpdate` time). Entries in the `workers` table will have their status set to `DONE`, where as entries in the `runs` table will be set to `NEW` status.

2.2.3 Worker

Figure 3 outlines the interactions between the workers and the database.

1. The worker retrieves N runs from the database which can be controlled via the **--runs-to-batch** and related options on the worker command line.

2. The worker processes each runs one at a time, periodically updating the database and checking if the run should be killed. For AEATK developers, this happens when the worker's `TargetAlgorithmEvaluator` triggers the observer. If observation isn't supported by the `TargetAlgorithmEvaluator` then this will not occur.

3. The worker then sleeps out the remaining time on delays between requests.

4. The worker periodically checks the workers table to see if its row has and `upToDate` set to 0, if so it refreshes the parameters, and takes any action as appropriate. After this step, if any runtime limits have been reached the worker will exit.

2.3 Basic

This section will outline the basic usage of the MySQL Target Algorithm Evaluator with a sample AEATK application, that is included with the MySQL Target Algorithm Evaluator package. The application performs random evaluations of the branin function (see http://www.sfu.ca/~ssurjano/branin.html) and reports the minimum value found out of all evaluated. Other AEATK applications should have the same command line interface, and so by following the same steps here in the other application you should be able to run that application in a distributed setting.

As stated previously it is assumed that you are familiar with the wrapper execution, if not please consult Section 5 of the SMAC Manual. The wrapper included is based on the wrapper included within SMAC, and requires python to be installed. The branin function has a global function minimum of 0.397887 which occurs at $(-\pi, 12.275), (\pi, 2.275)$ and $(9.42478, 2.475)$.

NOTE: This wrapper has been modified to sleep for 0.5 seconds to provide us with a more interesting example.

LOCATION: The files for this example are in the `example` subfolder.

We can verify this by executing the wrapper directly:

```
$./branin-sleep.py null 1 0 -1 -x1 -3.141 -x2 12.275
Result of this algorithm run: SUCCESS, 0, 0, 0.397891, 0
```

You can run the included program `branin-search` which will by default perform the search locally.

2.3.1 Running the Program Locally

```
./branin-search
[INFO ] Starting Random Search for minimum of Branin Function
[INFO ] Branin Search has completed 1 runs out of 100
[INFO ] Branin Search has completed 26 runs out of 100
[INFO ] Branin Search has completed 51 runs out of 100
[INFO ] Branin Search has completed 76 runs out of 100
[INFO ] Best value for the Brainin Function found was: 0.593666 and occurred at
-x1 '-3.142700170192641' -x2 '12.720123424738636'. Search took 55.796 seconds
```

If you run `branin-search` you should see the output given above and it should take approximately 1 minute to execute. This program is running the algorithm via the command line, and with the above arguments is being done sequentially.

2.3.2 Starting the workers

The first step is to create the database that will be used:

```
$mysql -h <hostname> -u <username>
mysql>CREATE DATABASE braninSearch;
Query OK, 1 row affected (0.03 sec)
```

Before running our application with the MySQL Target Algorithm Evaluator we should start some workers to be able to process them. We can start them with the following command [2]:

```
$mysql-worker --pool example --mysql-username <username> --mysql-password
 <password> --mysql-hostname <hostname> --mysql-port <port> --mysql-database
braninSearch --delay-between-requests 1
```

[2]The default port for MySQL is 3306

Suggestion

The final argument **--delay-between-requests** is optional, it changes how often the database is polled. In general the default value is conservative but good enough. We are changing it here because it is too conservative for the scale and duration of this tutorial. In general it is recommended that you leave this value alone, especially at first. For more information see FAQ 3.3.7.

The example target algorithm, `branin-sleep`, is executing a `sleep()` call and not actually using CPU processing time. As a result we can easily process these jobs even on a single core machine. I would recommend starting 8 workers to process these runs.

2.3.3 Running with the MySQL Target Algorithm Evaluator

We can change the `TargetAlgorithmEvaluator` implementation on the command line using the **--tae** argument, and supply the necessary arguments to the Target Algorithm Evaluator. This and other Target Algorithm Evaluator options are the same across this application, SMAC, and other applications. For more information see FAQ 3.1.1.

```
$./branin-search --tae MYSQLDB --mysqldbtae-pool example --mysqldbtae-username
 <username>  --mysqldbtae-password <password> --mysqldbtae-hostname <hostname>
--mysqldbtae-port <port> --mysqldbtae-database braninSearch
```

Important

Note that the worker and AEATK application use different arguments for specifying the same thing. This is because the worker is itslef an AEATK application, and consequently could use the MySQL Target Algorithm Evaluator. By default the workers will use the command line Target Algorithm Evaluator, but they can use any one, even a plugin.

Suggestion

You can remove the need to specify the arguments for MySQL on the command line everytime using option files, see FAQ 3.3.9 for more information.

After this starts executing you can verify that the runs were inserted into the database:

```
mysql> SELECT status,COUNT(*) FROM example_runs GROUP BY status;
+----------+----------+
| status   | COUNT(*) |
+----------+----------+
| NEW      |       64 |
| ASSIGNED |       14 |
| COMPLETE |       22 |
+----------+----------+
3 rows in set (0.00 sec)
```

The above indicates we are roughly 20% completed the runs, eventually when all runs are completed the `branin-search` application should exit, and you should see the full output that is similar to the following:

```
$./branin-search --tae MYSQLDB --mysqldbtae-pool example --mysqldbtae-username
```

```
<username> --mysqldbtae-password <password> --mysqldbtae-hostname <hostname>
 --mysqldbtae-port <port> --mysqldbtae-database braninSearch

[INFO ] Starting Random Search for minimum of Branin Function
[INFO ] Branin Search has completed 14 runs out of 100
[INFO ] Branin Search has completed 44 runs out of 100
[INFO ] Branin Search has completed 71 runs out of 100
[INFO ] Branin Search has completed 99 runs out of 100
[INFO ] Best value for the Brainin Function found was:
0.593666 and occurred at -x1 '-3.142700170192641' -x2 '12.720123424738636'.
Search took 10.294 seconds
```

So in this case we got a 5.4× speed up. In general for very short runs the over head of the database can be pretty extreme, but is largely independent of the number of runs. For more information on how performance changes see FAQ 3.5.2.

NOTE: If we run the command again it completes almost instanteously:

```
[INFO ] Best value for the Brainin Function found was: 0.593666 and occurred at -x1
```

This is a result of the caching, as all runs are complete. You can delete the runs in the database to try the search again, or you can change the **--seed** argument to the `branin-search` program.

2.4 Installation

Up until this point we have skipped talking about how to install the MySQL Target Algorithm Evaluator, in the above example however we didn't need to install it because it came with the MySQL Target Algorithm Evaluator. Lets look at another example, the `algo-test` utility included in SMAC (available at http://aclib.net/smac/. Within SMAC are a number of sample scenarios, any one of them well do fine. The algo-test utility essentially does a sample algorithm run, and is useful for verifying that SMAC can run the algorithm. It is an AEATK application, and so we can change the TAE on the command line. First lets verify that we can run it locally:

```
./util/algo-test --scenario-file ./example_scenarios/branin/branin-scenario.txt
Starting algo-test with 128 MB of RAM
...
[INFO ] Run 0 on Instance(1):no_instance has status =>  RUNNING, 0.0
[INFO ] Run Completed
[INFO ] Run 0 on Instance(1):no_instance with config: -x1 '2.5' -x2 '7.5' _
 had the result => SAT, 0.0, 0.0, 24.129964, 1,
```

Now if we check the version of the `algo-test` utility we will see the following:

```
$./util/algo-test -v
Starting algo-test with 128 MB of RAM
**** Version Information ****
Algorithm Execution & Abstraction Toolkit ==> v2.08.01-master-776 (296fd9ab6768)
Java Runtime Environment ==> Java HotSpot(TM) 64-Bit Server VM (1.7.0_40)
```

```
OS ==> Linux 3.13.0-39-generic (amd64)
Random Forest Library ==> v1.05.01-master-108 (7fba58fe4271)
SMAC ==> v2.08.01-master-740 (217d25a4724f)
```

Now copy the *contents* of the `mysqldbtae-plugin` folder into the lib folder and run the same command again (still executing locally), and observe that you now see a row for the MySQL Database Target Algorithm Evaluator:

```
$./util/algo-test -v
Starting algo-test with 128 MB of RAM
**** Version Information ****
Algorithm Execution & Abstraction Toolkit ==> v2.08.01-master-776 (296fd9ab6768)
Java Runtime Environment ==> Java HotSpot(TM) 64-Bit Server VM (1.7.0_40)
MySQL Database Target Algorithm Evaluator ==>
v0.92.00b-development-128 (b6d8a7fa90aa)
OS ==> Linux 3.13.0-39-generic (amd64)
Random Forest Library ==> v1.05.01-master-108 (7fba58fe4271)
SMAC ==> v2.08.01-master-740 (217d25a4724f)
```

Now following the exact same procedure as before we can use the MySQL Target Algorithm Evaluator. First we start a worker:

```
$./lib/mysql-worker --pool example2 --mysql-username <username> --mysql-password <p
```

Then we execute the utility the same as before with appropriate arguments:

```
$./util/algo-test --scenario-file ./example_scenarios/branin/branin-scenario.txt
--tae MYSQLDB --mysqldbtae-pool example2 --mysqldbtae-username
<username> --mysqldbtae-password <password> --mysqldbtae-hostname <hostname>
 --mysqldbtae-port <port> --mysqldbtae-database <database>
Starting algo-test with 128 MB of RAM
...
[INFO ] Run 0 on Instance(1):no_instance has status =>  SAT, 0.0
[INFO ] Run Completed
[INFO ] Run 0 on Instance(1):no_instance with config: -x1 '2.5' -x2 '7.5'
 had the result => SAT, 0.0, 0.0, 24.129964, 1,
```

We can confirm that this executed in the database by running the same query:

```
mysql> SELECT status,COUNT(*) FROM example2_runs GROUP BY status;
+----------+----------+
| status   | COUNT(*) |
+----------+----------+
| COMPLETE |        1 |
+----------+----------+
1 row in set (0.00 sec)
```

2.5 DZQ

Included with the MySQL Target Algorithm Evaluator is the `dzq` utility, it is a utility that wraps arbitrary command line calls within the AEATK framework and executes them using workers. At first glance this may seem quite circular and unnecessary, but in some cases this provides an advantage:

- If your cluster only lets you schedule some fixed number of jobs, this can allow you to avoid having to statically partition the jobs into smaller jobs.
- If your jobs may have very small runtimes, this can lower the dispatch overhead (which in our case can be several minutes per job).
- If your jobs may randomly fail non-deterministically, and if the scripts supply proper exit codes (0 for success, > 0 for failure), you can monitor those jobs in the database as easily manage the failures, etc...
- You don't have a queueing system of your own but would like one that is somewhat familiar [3].

For more information run `dzq --help` on the command line, note that it essentially contains a built in wrapper decoder that maps the standard wrapper format to something that will call arbitrary commands. By default it uses the `CLI` Target Algorithm Evaluator, you will need to use the `MYSQLDB` one to get full benefits.

2.6 Repair

Included with the MySQL Target Algorithm Evaluator is the `mysql-repair` utility. This utility will upgrade some older versions of the database to the new format, and repair the hash codes of entries. The command line is pretty straight forward, again consult **--help** for more information. Additionally a section of the FAQ has

3 FAQ

3.1 Troubleshooting

3.1.1 How can I verify that the MySQL Target Algorithm Evaluator is available?

This can depend on how well the applications honors certain AEATK conventions [4]. The first step would be to run the application with the **-v** argument and see if MySQL is listed. You can also look at the **--help** argument and look at the allowed values for the **--tae** argument, and also whether any arguments start with **-mysqldbtae-** . If none of these are accessible then it is entirely up to the author of the application, and some applications may not expose this on the command line.

[3]The author is not recommending that this be used as a queueing system, it is not, but if you were using the MySQL workers anyway

[4]These conventions are evolving, and undocumented, so don't blame the author of your particular application

3.2 I have really big instances, configurations, etc, and they don't seem to work.

There are a couple of limits that you could run into when you have extremely large instances or configuration spaces. They generally fall into three camps, changing the column size limits, changing the batch insert size, and changing MySQLs configuration. The advice that follows presumes that this is occurring either during insertion of a runs in the AEATK application, or logging the result of a run on the worker.

There are generally two things that need to change in order for you to get this to work, the first is that you may or may not be hitting a limit on the column of a table. For instance if your problem instance is larger than 8172 characters [5], you may get an error when inserting it.You generally can safely increase the the limits of the table without directly causing a problem with the workers. However if you do any subsequent queries involving the column that performance may then tank (for instance if you were to change problemInstance to a TEXT, and then want to get an average runtime grouped by problemInstance, MySQL may perform horribly or may actually not let you [6]. Additionally the author believes that an InnoDB row can be at most 64 KB (or some value like this), excluding `TEXT` and `BLOB` columns.

The next limit you may hit involves the MySQL setting `max_allowed_packet`. A single packet cannot exceed this value in bytes. When inserting multiple runs the MySQL Target Algorithm Evaluator will batch the insertions in sizes controlled by **--mysqldbtae-batch-insert-size**, at the time of writing the value of this defaulted to a conservative 500 (rows typically involve no more than 1 KB), you could lower this value almost to 1, although you pay the overhead of increased latency for database traffic. If even 1 row cannot be inserted, then you will most likely have to change the `max_allowed_packet` setting, which at least on my system was 16 MB. It seems that in MySQL 5.5, this value can be increased up to 1 GB (http://dev.mysql.com/doc/refman/5.5/en/server-system-variables.html#sysvar_max_allowed_packet).

If this still isn't enough, there isn't much you can do other than offload the work to a shared file system.

3.2.1 What happens if there is a problem on the worker?

The MySQL Target Algorithm Evaluator is designed to be robust against failures, and there are a number of possible failure scenarios on the worker. In general, if the evaluation of a target algorithm signals a failure, this will be written to the database as a failed run. If during the evaluation of a run an `Exception` is raised, this will be logged as an **ABORT** and the exception will be encoded in the additional run data on the field. When the run is decoded on the AEATK Application side the ABORT will be propagated and the reason logged.

If some other `Throwable` occurs at some point in the evaluation, or when the worker shuts down it makes a generic attempt to reassign all runs back **status** column to `NEW`. At several points in the shutdown procedure the worker will make an attempt to reassign the runs back to `NEW`.

Important

If some runs get reassigned back to `NEW` in the above manner, then their priority will be set to `LOW`. The reason for this is to prevent the run from acting like a poison pill and cycling through every worker and causing it to terminate.

In some cases such as the worker experiencing a `SIGSEGV`, or the machine shutting down, the run will be stuck assigned to the worker. There are two processes that will recover from this. Periodically (controlled by the **--dead-job-check-frequency**, with a default of two minutes. The AEATK application will check for workers that haven't checked in recently, and for jobs that are stuck in assigned, and will reassign them to

[5]This was the limit at time of writing

[6]The author was too lazy to check which it is.

`NEW`. Workers will also perform this check, but only if they have no work to do. In both of these cases the jobs keep there existing priority.

3.2.2 How can I verify that everything is working with the workers?

The way that the author does this primarily is through repeatedly executing, the `SELECT status, COUNT(*) FROM <POOL>_runs GROUP BY status;` query at various intervals, and seeing if the number of runs seem to me making forward progress. Another safe query to execute is `SHOW PROCESSLIST`, in large bulk workloads it should consist largely of connections that are Sleeping. A run can either be sleeping when the *Command* is `Sleep` or when the query is `SELECT SLEEP();`

In intensive or online applications, the thing to watch out for is the *Time* column, except for sleeping queries the time here should generally be very small. In some cases a careless query against the **runs** table can cause the workers to stall, in this case you should try the `KILL QUERY` SQL command with the value in the *Id* column.

3.2.3 Why is my AEATK Application and/or the workers stuck?

There are a number of ways that an application can seemingly become "stuck", here are a number of things to check for

- It is possible that one or more runs are stuck in either `NEW` or `ASSIGNED`, you can verify this via: `SELECT status, COUNT(*) FROM <POOL>_runs GROUP BY status;`.
- Check to ensure that there are alive workers, you can check this in the workers table, but this can be misleading, you may also want to verify the processes are still running directly on their hosts.
- If a run got changed to have a priority of `LOW` it can be a while before it gets completed. You can reset the run priority manually with an UPDATE query, by primary key.
- Check that the database isn't being floored or queries are not backed up for instance with `SHOW PROCESSLIST`. If you see a bunch of queries with huge time values that aren't sleeping (see also FAQ 3.2.2) , it may be possible that a particular query is taking too long to execute and should be killed.
- Check that your database server isn't out of disk space. MySQL will essentially pause every query once it runs out of disk space.

3.2.4 I'm still stuck what other things should I look at?

If you are still having a problem it is helpful to look at the `jstack` utility included with Java to get a thread dump of both a worker process and the AEATK application, it may shed light on the problem.

3.3 Usage

3.3.1 How do I control how long the workers run for?

There are a number of options that exist that can control the workers execution time. Using the **--help-level** `ADVANCED` on the `mysql-worker` will allow you to see the full set of options. The following options will control execution time, but you should consult the help screen for authoritative information:

--time-limit the maximimum amount of time the worker can execute for (it won't attempt jobs that could go longer than this).

--idle-time-limit the amount of time that is allowed to occur on the worker without it doing any work before it shus down.

--pool-idle-time-limit the amount of time that is allowed to occur across workers before workers will be shutdown if idle. This limit only looks at workers that existed within the past 2 weeks.

--num-uncaught-exceptions the number of uncaught exceptions that a worker will allow to exist before shutting down. By default the logical worker will die, and another one will restart within the same process.

Most of the above can be changed directly in the **workers** table. Another way the workers can be shutdown is the **--mysqldbtae-shutdown-workers-on-complete** on an AEATK application and when it "finishes"[7].

3.3.2 How do I manage runs that signal ABORT?

The **ABORT** signal implies that the experiment and AEATK master needs to shutdown. If this occurs during a run, it will be propagated to the client. If the run is re-requested the run will be retried from scratch. You can manually set runs that have a runResult of **ABORT** back to `NEW` in the database if you'd like.

3.3.3 How do I manage runs that signal CRASHED?

The **CRASHED** signal implies that this run crashed. This will be propagated to the client, and the client will continue processing it normally. Runs that are **CRASHED** do count as cache hits, and will not be retried from scratch. Many CRASHED runs are due to certain configurations or runs having an intrinsic problem, but some seem to be occasional random glitches that are never reproducible. AEATK applications generally have an option **--retry-crashed-runs** that will automatically cause a run to be resubmitted. Unfortunately with the MySQL Target Algorithm Evaluator, this option is ignored because it will just be a cache hit. Consequently one should set the option **--retry-crashed-runs** on the worker side, for it to be retried.

3.3.4 How do I manage runs that may be KILLED?

Some applications may decide after observing a series of runs for some amount of time, that they no longer want the runs to run until all are completed. They may request that some subset of runs be killed. Because of the anonymized nature of the AEATK applications and workers, it's possible that one AEATK application may request a run be killed that another application doesn't want killed. In this case the run is killed, and both applications will see this. The application that did not request the kill may in fact not even be able to handle this case properly because it never kills runs. This can cause the invariants in the application that didn't request the kill to be broken. The moral of this story is that if you are running multiple AEATK applications that may kill runs, you should set the **--mysqldbtae-run-partition** to be unique for each individual run, this will allow them to share the same pool of workers, but will cause them not to use the same cache of runs, and so there will be no collisions. Section 4 discusses a possible way this could be improved.

[7]Typically this is on application shutdown, but for those developing applications it is when `notifyShutdown()` is called on the TAE

3.3.5 How safe is it to run queries directly against the database?

The MySQL Target Algorithm Evaluator is designed to allow for manual interference by you, in some cases while the database is running. The biggest risk is that you may cause the application to grind to a halt if you do something very expensive. This could be something as drastic and silly as an `ALTER TABLE`, or could just be any expensive query against a very large table. You can check the impact of your statement by using the `SHOW PROCESSLIST` command, and ensuring that the workers are healthy, like as described in FAQ 3.2.2.

You certainly can reset runs back to `NEW` while workers are running the biggest risk is a complicated `WHERE` clause would start stalling the workers. If a worker is currently executing or assigned that run it will think the run has been killed (to update a row a worker requires that it be in **status** `ASSIGNED` and the **workerUUID** matches the workers own, otherwise the `UPDATE` clause won't select the row.

Some things to keep in mind that have come up are that the hash is the only thing that the client really looks at when identify a match, so conceivably you could change the row such that the hash would no longer match up, but it is the hash that matters. The worker never looks at the hash, and does everything by `runID`, so you could alter the runs here.

3.3.6 Can I change the Target Algorithm Evaluator used by the workers to something custom?

Yes, simply install your Target Algorithm Evaluator implementation into the workers like you would any other utility, and then change the **--tae** option on the worker command line.

3.3.7 How can I increase performance of the MySQL Target Algorithm Evaluator?

There are a number of knobs that you can turn that will affect the performance (throughput and/or latency). This section may be out of date, so you may want to consult **--help** for the most authoritative information.

Important

Like all performance tuning tasks you should have a clear objective and an actual need for improvement. It is not recommended that you simply change these settings without due care, as it may negatively affect your application performance or in some cases may make the MySQL Server unusable by others. In particular we have experienced situations when the database entirely stalls and not a single worker is able to make forward progress.

Large Bulk Inserts If you are inserting a large number of rows at any given time by default they will be sliced into inserts of size 500, you can increase this with the **--mysqldbtae-batch-insert-size**. The reason all runs are not inserted all at once is because there is a limit to the size of a single SQL query can take (`max_allowed_packet`). If set to high, the program may fail inserting at some runs. See also FAQ 3.2. Another useful setting to change is **--mysqldbtae-max-jobs-to-poll** this controls how many jobs to poll for completion at any one time, this needs again to be below the max packet size. The actual size of the packet is unclear, and in many cases the MySQL Target Algorithm Evaluator is unaware of what the size is. The**--mysqldbtae-batch-insert-size** option also depends on how large your instance, and configuration space is. The **--mysqldbtae-max-jobs-to-poll** shouldn't be affected by any of this, and assuming the packets were ASCII text then it's just about <250 bytes of overhead, and then ∼60 bytes per run. [8].

[8]The command-line interface gives a MUCH more conservative estimate of 10K characters overhead and 2500 bytes per run, the estimate here is much tighter.

Decrease Latency of Runs If you would like your runs to be more responsive then the following settings may help you. With these settings you do risk negatively affecting your applications performance and the performance of your database server. The biggest change in the AEATK application you could make is **--mysqldbtae-poll-delay**, if you have many small requests (as opposed to a few large requests), you may also want to increase **--mysqldbtae-poll-threads**.

On the worker you could lower the **--delay-between-requests** option, and also the **--runs-to-batch** option. It might also help to turn off **--auto-adjust-batch-size**. There are a few other options that on the worker you can play with, but they have increased risk.

Increase Throughputs of Runs To increase the throughput of runs, which are very short, you may want to increase both **--runs-to-batch** and **--max-runs-to-batch**.

3.3.8 Is there a way to have the workers shutdown when the AEATK application is completed?

Yes, see the **--mysqldbtae-shutdown-workers-on-complete** option.

3.3.9 How can I avoid having to supplying the arguments to the database on every execution?

AEATK applications have support for *Option Files*. These files are read on start up and options specified within them will change the default values (but can still be overridden), you can include any subset of these options or other options in these files. For the MySQL Target Algorithm Evaluator, there are two different files that are read. The first file is read by a standard AEATK application, and should be placed in a file (relative to your user accounts home directory): `.aeatk/mysqldbtae.opt`

```
mysqldbtae-database = <database>
mysqldbtae-username = <username>
mysqldbtae-password = <password>
mysqldbtae-hostname = <hostname>
mysqldbtae-port = <port>
```

The second file is read by the workers and should be placed in a file (relative to your user accounts home directory): `.aeatk/mysqlworker.opt`

```
mysql-database = <database>
mysql-username = <username>
mysql-password = <password>
mysql-hostname= <hostname>
mysql-port = <port>
```

Suggestion

The `mysqldbtae.opt` file will be read by every program when it uses the MySQL DB Target Algorithm Evaluator, and you can put any option you'd like in these files. Simply find the argument from **--help** and put it here, without the preceeding two dashes. It is recommended that you only put options under the *MySQL Target Algorithm Evaluator Options* that are for the plugin, as other options may change depending on the application.

When these changes are complete, you can remove all the corresponding options on the command line, unless you need to overwrite them. For instance to run the branin search program in 2.3.2, you could simply do:

```
./branin-search --tae MYSQLDB --mysqldbtae-pool example --evaluations 100
```

And to run the workers in Section 2.3.2, one would simply do:

```
$mysql-worker --pool example
```

3.4 Migration & Repair

3.4.1 When do I have to / How can I migrate an older version of the database to work with this version?

You only need to upgrade the database when something has been renamed or new functionality necessitates new columns and/or a change in hash function. At the time of the initial public release (September 2014), most of the issues that would have required this have been dealt with, so unless you had a pre-release you shouldn't need to upgrade.

NOTE: Prior to running the utility you should make a back up of your data with a utility like `mysqldump`.

If you do need to upgrade you should just be able to run:

```
./mysqldbtae-repair --mysql-host <host> --mysql-username <username>
 --mysql-password <password> --mysql-database <database> --pool <pool>
```

Or to upgrade every pool in the database try:

```
./mysqldbtae-repair --mysql-host <host> --mysql-username <username>
 --mysql-password <password> --mysql-database <database> --all-pools true
```

3.4.2 The migration utility failed, what should I do?

The only part of a migration that cannot easily be done manually is the hash calculation, thankfully the utility will only do more invasive stuff if the database suggests that it needs it.

First I would ensure that all of the tables have the correct structure, through manually constructing the correct `ALTER TABLE` statements. If you unzip the `mysqldbtae.jar` file you should see two `.sql` files. The first `tables.sql` contains what a tables should look like, the second `migration.sql` contains some sample `ALTER TABLE` clauses that will restructure tables. You only need to ensure that all the tables have the same names, column names, types, order as the values in the `tables.sql` file.

Once this is done, it's important to populate the `version` table with the correct value for the hash. The best way to get the current value of the hash, is to create a new pool (by starting the worker or an AEATK application, on a new pool) and look at the value of the hash here. At the time of writing it is a hash of the `tables.sql` file. The **version** column should be the same as well and the id should be the lowest value in the table (you mine as well delete everything else).

After this point, simply run the migration utility again. It should see the corrected tables version information, and simply repair hashes. If repairing of the hashes failed then this could happen for a couple of reasons.

1. Two or more rows now hash to the same value, which would be very bad, and these rows will have to be deleted manually.
2. Out of memory, in this case simply rerun the repair utility with the **--reset-all-run-hashes** set to `false`. This will cause the utility to only repair runs that hadn't been repaired previously, controlled by looking at which runs are PAUSED.

3.4.3 How can I speed up the repair of hashes?

Once the first repair utility gets to the point of repairing the runs table, it is safe to start additional versions (perhaps on other machines), provided that they have **--reset-all-run-hashes** set to `false`.

3.4.4 How do I import runs manually?

You should simply be able to import the data straight into the tables, with the correct values and run the repair utility. The MySQL Target Algorithm Evaluator encodes the parameter settings in a weird forward, but you should simply be able to write the value manually in the -name 'value' format, and it should work.

3.4.5 How do I export runs?

It would not be very hard to write a utility that does this, but no one has asked for it [9].

3.5 Performance

3.5.1 Is there a way to use runs with existing captimes more effectively?

Yes, most AEATK applications should have an option **--use-dynamic-cutoffs** that if set to true, will cause the captime of the scenario or `AlgorithmExecutionConfiguration` object to be used, the run will be monitored and once it exceeds the captime that was requested it will be terminated.

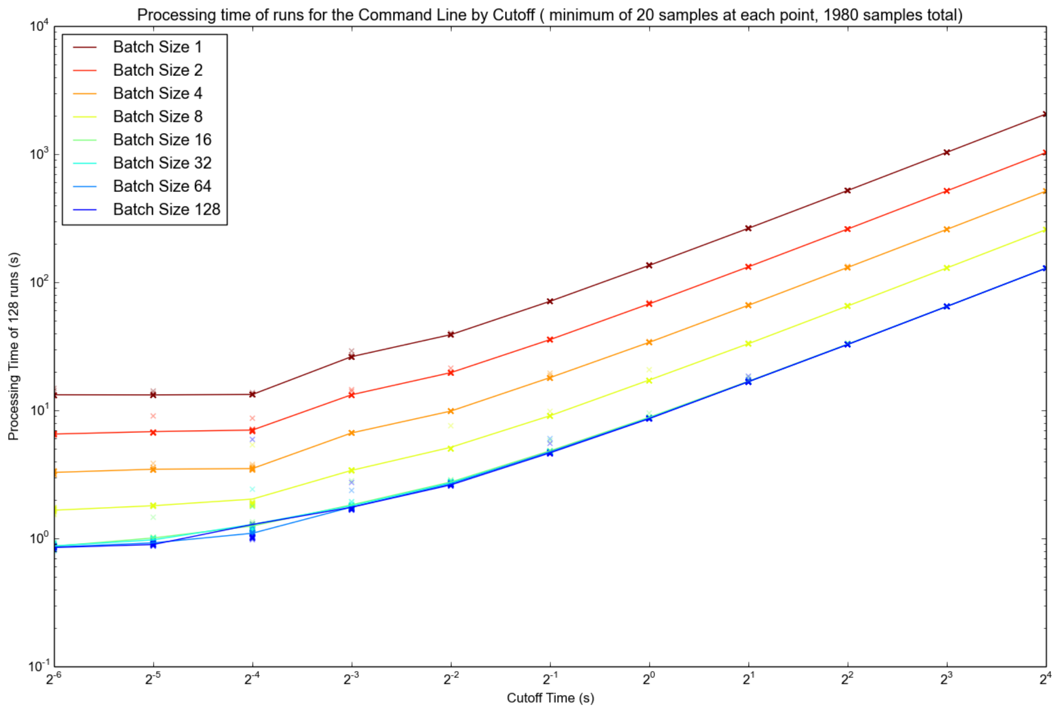
3.5.2 Do you have pretty plots that may or may not tell me something important about the performance of the MySQL Target Algorithm Evaluator?

Yes I do, most of the important performance considerations exist when doing runs that have very small execution times, and very small batch sizes. The following experiment considered how scheduling 128 runs with differing levels of batch size, that took varying levels of cutoff time. These experiments considered how the MySQL Target Algorithm Evaluator differs from the CLI Target Algorithm Evaluator, and were conducted on 16 core machines, each either directly executing a script that sleeps, or submitting them to the database to be executed by a worker running locally (each combination was run 20 times). The batch size controls how many runs were submitted to be done synchronously, so a batch size of 1 implies that only 1 core of 16 was in use at the time. The mysql runs were done with fairly aggressive settings of a 500 ms poll delay, and workers polling once per second. Both sets of run were executed on a shared cluster in shuffled order. The MySQL runs were done in parallel using the same MySQL server concurrently.

Figure 4 on page 21 suggests the following:

1. The command line runs have much lower variance that the MySQL runs, probably due to the fact that there is less reliance on polling.

[9] which suggests maybe that this isn't a frequently asked question



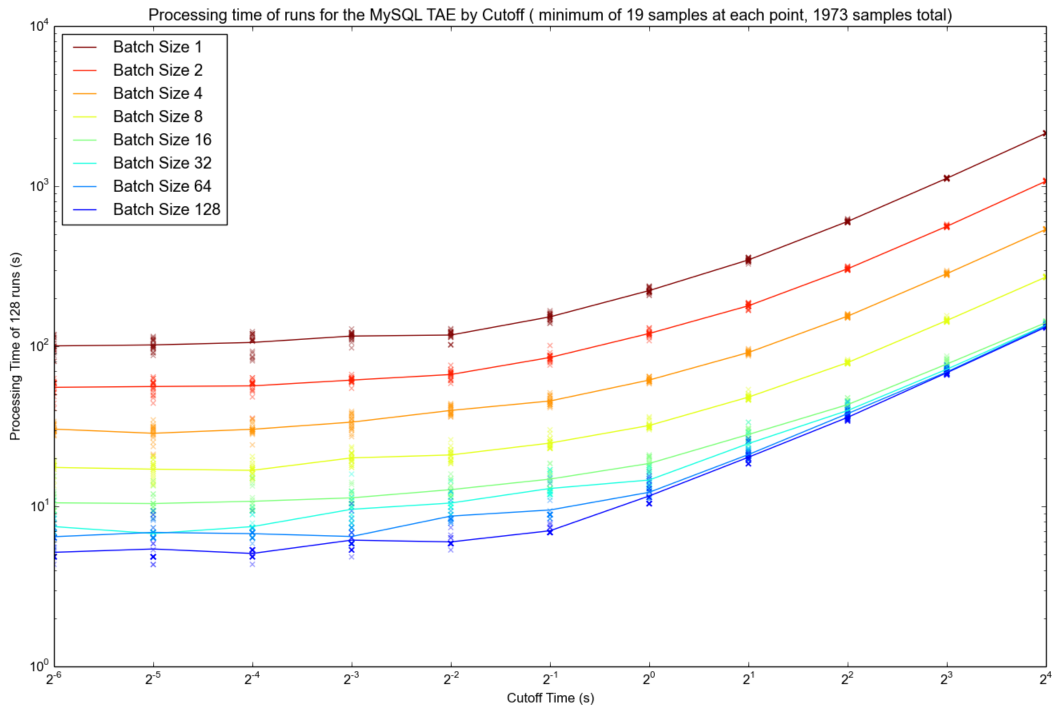


Figure 4: Processing Time of different batch sizes as a function of cutoff CLI (top/left) versus MySQL (bottom/right)

2. Increasing the batch size past 16 doesn't make much difference for the command line Target Algorithm Evaluator, but for the MySQL Target Algorithm Evaluator it makes a difference. This is probably due to the reduction in number of trips required to the database and the way that polling is implemented.

3. For the command line, runs less than about 2^{-4} seconds are dominated by overheads, where as for MySQL it is less clear but the graph tends to flatten around 2^{-2}.

In Figure 5 we can see how much slowdown the competing TAEs experience, in this case we have grouped the runs by cutoff, instead of batch size and suggests the following:

1. The slowdown experienced by runs seems mostly affected by the cutoff time, in the Command Line Target Algorithm Evaluator, except for some increased variation in the 0.0625 (s) case, the lines are mostly distinctively flat.

2. In the MySQL Target Algorithm Evaluator case, the slowdowns are a factor of 10 slower for smaller batches, and even for the longest runs of 16.0 (s) still noticeably above $1\times$.

3. In the MySQL case, batch size does have noticeable impact on overhead, up until a batch size of 16, the process becomes less efficient, and afterwards becomes more efficient. In general the batches of 128 have less slowdown than batches of 1. The improved efficiency after a batch size of 16 makes sense due to less polling of the database and potentially more efficient worker utilization, the decline prior to 16 is unclear at this time.

4 Known Issues & Future

Instance Specific Information not hashed Instance specific information won't form part of the hash. This is because the key to an instance is an instance name, but it's unclear whether this should be true in a database or not.

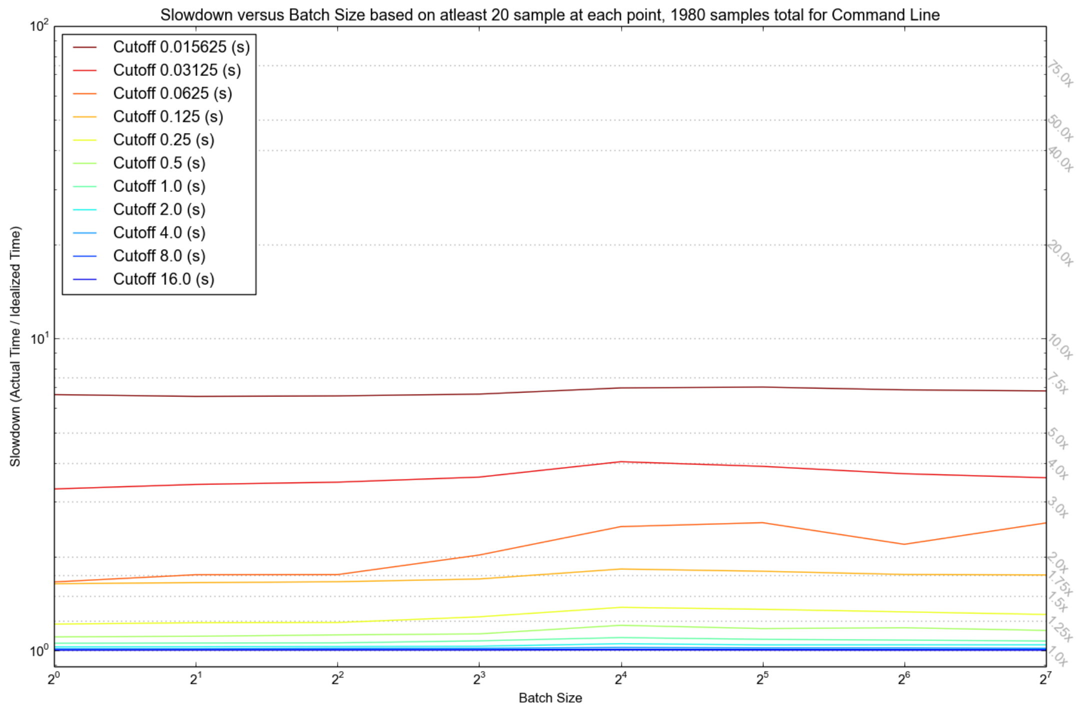
Instance features not supported Instance features are not supported (instance features that exist in the AEATK application, will not be available on the worker). This could be fixed if needed.

Better Killed Run Support could include the following mechanism. Add a new column, max_observered_runtime. This column simply shows the maximum observer runtime thus far. When we are observing runs, we always observe this column for the runtime, and when we update the database we always take the MAX of the current value and our value. When retrieving runs from the database, if we see that the run is **KILLED**, if we didn't request that it is killed, we will resubmit it as `NEW`. However, we will still report the previously seen value, and if it is killed by us we will then kill it in the database.

Workers may hang if there is a high load on AEATK application This occurs if the AEATK application has high CPU utilization and attempts to reclaim jobs from dead workers. This may hang other workers who attempt to do a similar check.

5 Thoughts for Developers

This section outlines random thoughts, musings, ideas that people who have to investigate the internals of the MySQL Target Algorithm Evaluator might find useful. It should go without saying that you should be



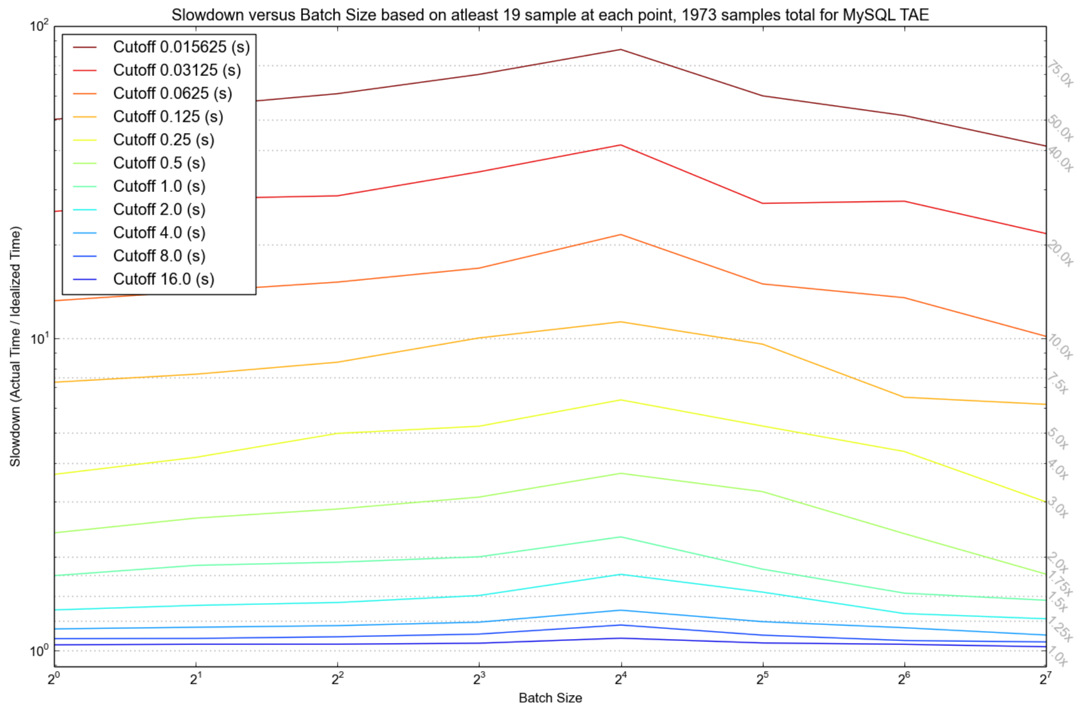


Figure 5: Slowdown of different cutoff times as a function of batch size CLI (top/left) versus MySQL (bottom/right)

very familiar with the AEATK API before attempting to digest this, and you should be very familiar with MySQL, and at least somewhat familiar with Java Concurrent Programming, lest this whole section appear to be incoherent ramblings.

- The use of `GET_LOCK()` prior to retrieving the runs on a worker is unfortunately due to a mysterious performance issue that I could not solve. Essentially randomly sporadic freeze occurs that causes database performance to tank. For more details see: http://forums.mysql.com/read.php?24,607838,607838#msg-607838.

- I would personally recommend avoiding adding more complicated logic to the retrieval of runs from the database by the worker. For instance more priority levels, or a more complicated ordering mechanisms would make the queries much more expensive. Presently it is believed that the SELECT query can be answered by only an index lookup only. One rule in MySQL, is that you can only have one column in an `ORDER BY` clause for an index to work efficiently, and we use it for FIFO ordering of runs.

- I have no idea why I didn't put most things in the `RunToken` instead of in the Map for `MySQLPersistenceClient`, it does make zero sense.

- A large literature exists suggesting that queues in MySQL are a bad idea. In some ways they are correct but note that most of the objections have been mitigated. Firstly we don't use `SELECT... FOR UPDATE`. Secondly the updates are atomic and done in a single batch [10]. Fourthly there are advantages to using MySQL, and because the data being collected generally only has a short life span most of the standard objections do not apply (for instance it isn't the case that the tables grow without bound over months or years).

- It is strongly recommended that you read *High Performance MySQL 3rd Edition* if you are going to tinker with performance.

- Most of the defaults and the settings are old and very, very conservative. They were chosen by simply observing good performance, noticing that the database locked up, and then doubling their values. The `GET_LOCK()` strategy seems to have been a silver bullet for this, and so it's possible the settings could be made far more aggressive.

- The indexes on the **runs** table seem to be counter intuitive, their purpose is primarily to act as covering indexes for queries (http://en.wikipedia.org/wiki/Database_index#Covering_index), so even though they aren't very effective and segmenting the data, they can make things very fast. In fact while writing this manual a performance bug was detected related to this. During a migration of a very large database, the estimated time to complete the migration (repair hashes), decreased linearly for several hours implying that the runs were being processed at a constant rate. Eventually when 55% of the runs were corrected, MySQL decided to switch indexes, this caused the queries to become a thousand times more expensive. We fixed this using an `USE INDEX` query but note the difference:

```
mysql> SELECT runID,1 AS priority
 FROM dsmac_cli_runs_runs USE INDEX (status2)
```

[10]Which is odd because in MySQL you can't SELECT and UPDATE a table at the same time, but for some reason this query works, I suspect it's because an internal temporary table is made

```
  WHERE status="PAUSED"
   AND priority="NORMAL"
    ORDER BY runID
    LIMIT 1;

+----------+----------+
| runID    | priority |
+----------+----------+
| 12825485 |        1 |
+----------+----------+
1 row in set (0.00 sec)

mysql> SELECT runID,1 AS priority
 FROM dsmac_cli_runs_runs
 WHERE status="PAUSED"
  AND priority="NORMAL"
   ORDER BY runID
   LIMIT 1;

+----------+----------+
| runID    | priority |
+----------+----------+
| 12825735 |        1 |
+----------+----------+
1 row in set (5.77 sec)
```

6 Acknowledgements

Special thanks to Daniel Geschwender who implemented a number of features that made them much better to use. Also to Chris Thronton, Alex Fréchette, Richy Chen, Alim Virani, Chris Cameron, and Frank Hutter for feedback. Finally to people within #mysql on Freenode, who were very helpful in sorting out the performance problems.

7 Appendix I - Sample MySQL Configuration

Important!
The following is our current MySQL Configuration for MySQL 5.5. It *disables* binary logging and as such if the database loses power there would be no backup or redundancy available. This is appropriate for our use cases, but may not be appropriate for yours.

```
#This my.cnf file was created by Steve Ramage <seramage@cs.ubc.ca>
#It is based on the configuration of our local arrowdb database server
#and a read through of High Performance MySQL 3rd Edition, Chapter 8.
```

```
#Important Note: Binary logging is disabled, it is largely expected that
#few if any users would benefit from any kind of replication and/or
#point-in-time recovery. as such we disabled it


[client]
port            = 3306
socket          = /var/run/mysqld/mysqld.sock

# Here is entries for some specific programs
# The following values assume you have at least 32M ram

# This was formally known as [safe_mysqld]. Both versions are currently parsed.
[mysqld_safe]
socket          = /var/run/mysqld/mysqld.sock
nice            = 0

[mysqld]
#
# * Basic Settings
#
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3306
basedir         = /usr
#datadir                = /var/lib/mysql
datadir         = /data/db/mysql
tmpdir          = /tmp

lc-messages-dir = /data/db/mysql


skip-external-locking

#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
#bind-address           = 127.0.0.1
bind-address            = 0.0.0.0

# This replaces the startup script and checks MyISAM tables if needed
# the first time they are touched
myisam-recover         = BACKUP

server-id=1
slow-query-log=1
default-storage-engine=InnoDB

#We can expect MANY clients to connect at times with many threads

max_connections=22000
max_connect_errors=10000
thread_cache_size=256


#InnoDB Options
innodb = FORCE

#Set this to 80% of RAM
innodb_buffer_pool_size = 16G

innodb_log_file_size = 1G
innodb_log_buffer_size = 8M

innodb_file_per_table = 1
innodb_flush_method = O_DIRECT
innodb_flush_log_at_trx_commit=2

innodb_file_format=Barracuda
innodb-file-per-table = true

#Couldn't set innodb_io_capacity as I wasn't able to login to disk directly
#Suspect RAID6 may perform poorly, unsure though

#Other settings taken from arrow (not tuned)
tmp_table_size=128M

sort_buffer_size=16M
join_buffer_size=256M
query_cache_size=128M
query_cache_limit=2M
max_allowed_packet=64M
table_cache=2048


#Key Buffer Size  (MyISAM options)
```

```
myisam_sort_buffer_size=256M
read_rnd_buffer_size=4M
read_buffer_size=2M
key_buffer = 256M
key_buffer_size=64M

#Query Cache

#Taken from High Performance MySQL (3rd Edition p 343)

tmp_table_size = 32M
max_heap_table_size = 32M


open_files_limit = 65535

[mysqldump]
quick
quote-names
max_allowed_packet      = 16M

[mysql]
no-auto-rehash # faster start of mysql but no tab completitition

[isamchk]
key_buffer              = 16M

#
# * IMPORTANT: Additional settings that can override those from this file!
#   The files must end with '.cnf', otherwise they'll be ignored.
#
#!includedir /etc/mysql/conf.d/
```