Auto Bencher
The Automated Benchmarking Tool

Presented by Yu-Shan Lin
DataLab, NTHU
June 14th 2021

Outline

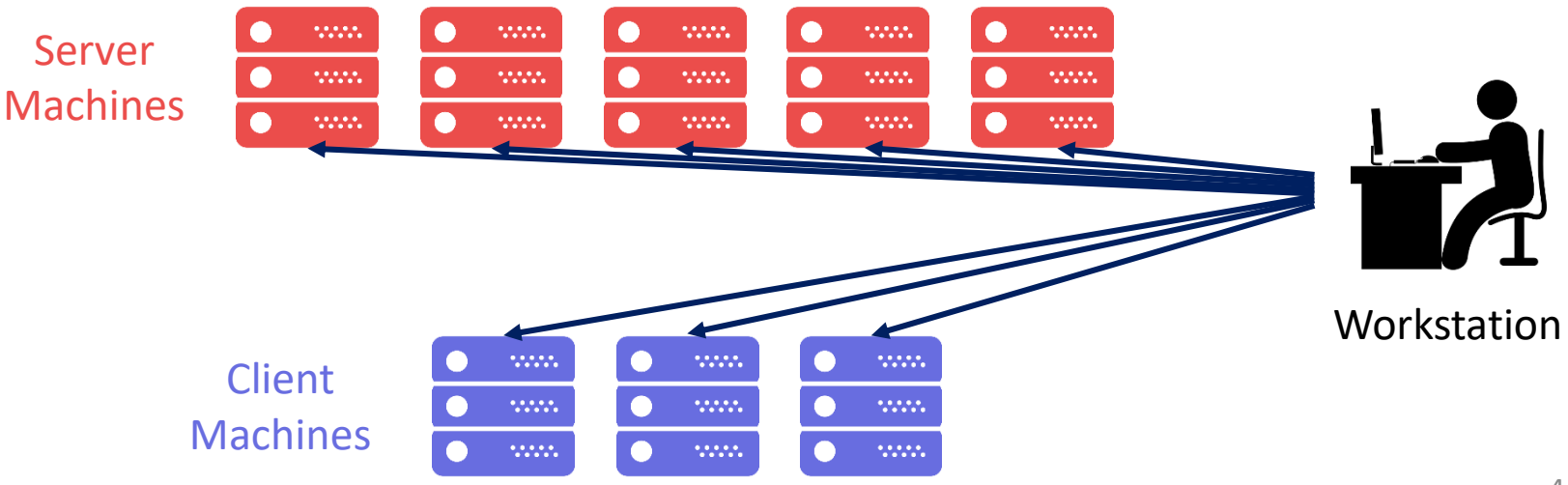
- Why do we need a tool?
- Main Workflow
  - Setting up Environments
  - Loading Testbed
  - Benchmarking
- Other Functions

Outline

- Why do we need a tool?
- Main Workflow
  - Setting up Environments
  - Loading Testbed
  - Benchmarking
- Other Functions

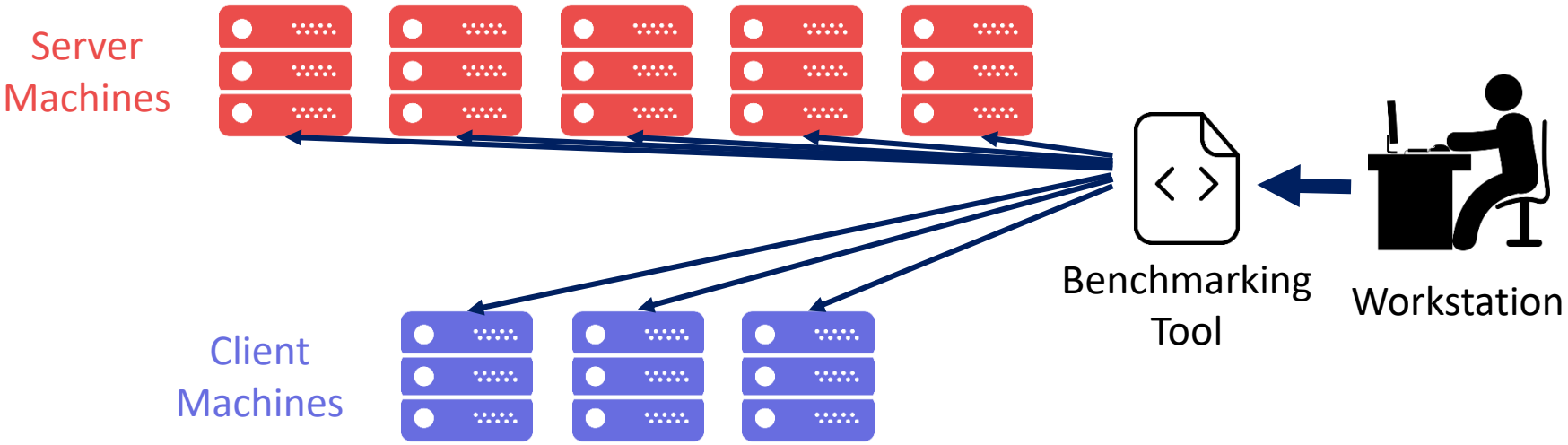
Running Distributed DBMS

- ElaSQL is a **distributed** DBMS
  - To benchmark the system, we have to start up all the servers and clients **simultaneously** while checking if there is any error showing up.



Benchmarking Tools

- We designed and wrote a lot of scripts and tools to avoid such complicated management.

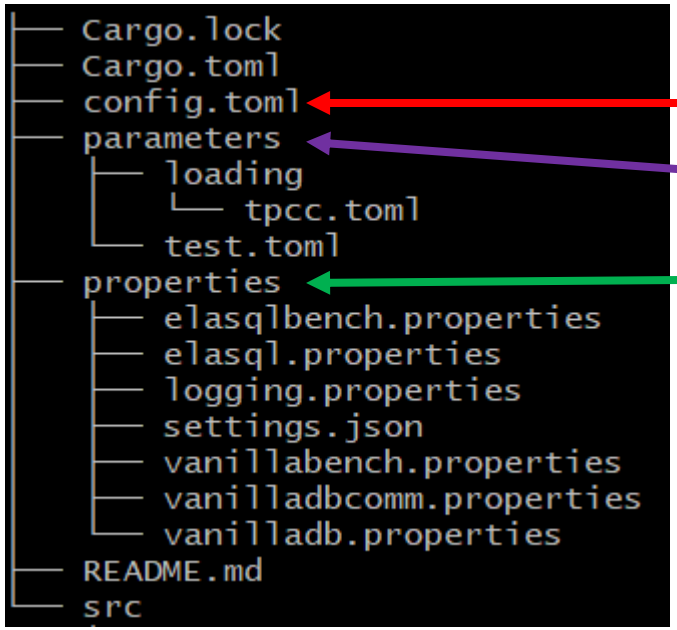


Outline

- Why do we need a tool?
- Main Workflow
  - Setting up Environments
  - Loading Testbed
  - Benchmarking
- Other Functions

Download Auto Bencher

```
git clone https://github.com/SLMT/auto-bencher
```



Setting Up Environment

- Let's say you have a complete new environment.
  - Just a fresh Linux system.
  - Without installing Java runtime.
- We first have to…
  - Setup JDK
  - Setup workspaces for putting logs, reports, and databases
  - …on all machines.
- The auto bencher will do all above for you!

Downloading a JDK

1. Download Oracle JDK from the office website
   - Remember to choose the one for Linux
   - Better be in **Java SE 8**. We experienced a few problems when using other versions of JDKs.
   - Name looks like: “jdk-8u211-linux-x64.tar.gz”

2. Put it in somewhere in the auto bencher
   - E.g., “package/jdk-8u211-linux-x64.tar.gz”

Configuring The Auto Bencher

In ‘config.toml’:

```
[system]
user_name = "db-team"
remote_work_dir = "auto-bencher-workspace"

[jdk]
use_custom_jdk = true # Reserved. Not used for now.
dir_name = "jdk1.8.0_211"
package_path = "package/jdk-8u211-linux-x64.tar.gz"

[jdk.vmargs]
sequencer = "-Xmx4g -Xms4g"
servers = "-Xmx4g -Xms4g"
clients = "-Xmx4g -Xms4g"

[machines]
# [Optional] if no sequencer is set, the system will pick one of servers as the
sequencer.
sequencer = "192.168.1.100"
servers = ["192.168.1.11", "192.168.1.12"]
clients = ["192.168.1.12"]
```

Initializing Environments

```
> cargo run init-env
```

- ‘cargo’ is the project manager of Rust.
  - Like ‘npm’
- This command complies the source code and then executes the binary.
- It only needs to run once unless the configuration is changed.

Screenshot of ‘init-env’

```
   Finished dev [unoptimized + debuginfo] target(s) in 34.10s
    Running `target/debug/auto-bencher init-env`
INFO  auto_bencher::subcommands::init_env > Starts initializing the environment
INFO  auto_bencher::subcommands::init_env > Checking local jdk: package/jdk-8u211-linux-x64.tar.gz
INFO  auto_bencher::subcommands::init_env > Checking node '192.168.1.100' ...
INFO  auto_bencher::subcommands::init_env > Creating a working directory on 192.168.1.100
INFO  auto_bencher::subcommands::init_env > Checking java runtime on 192.168.1.100
INFO  auto_bencher::subcommands::init_env > Node '192.168.1.100' checked
INFO  auto_bencher::subcommands::init_env > Checking node '192.168.1.11' ...
INFO  auto_bencher::subcommands::init_env > Creating a working directory on 192.168.1.11
INFO  auto_bencher::subcommands::init_env > Checking java runtime on 192.168.1.11
INFO  auto_bencher::subcommands::init_env > Node '192.168.1.11' checked
INFO  auto_bencher::subcommands::init_env > Checking node '192.168.1.12' ...
INFO  auto_bencher::subcommands::init_env > Creating a working directory on 192.168.1.12
INFO  auto_bencher::subcommands::init_env > Checking java runtime on 192.168.1.12
INFO  auto_bencher::subcommands::init_env > Node '192.168.1.12' checked
INFO  auto_bencher::subcommands::init_env > Checking node '192.168.1.13' ...
INFO  auto_bencher::subcommands::init_env > Creating a working directory on 192.168.1.13
INFO  auto_bencher::subcommands::init_env > Checking java runtime on 192.168.1.13
INFO  auto_bencher::subcommands::init_env > Node '192.168.1.13' checked
INFO  auto_bencher                        > Auto Bencher finishes.
```

Outline

- Why do we need a tool?
- Main Workflow
  - Setting up Environments
  - Loading Testbed
  - Benchmarking
- Other Functions

Loading Testbed

- The next step is to generate testing database for benchmarking.
- We have to…
  - Set the properties files
  - Send jar files to all machines
  - Start up servers and clients
  - Check if they finish
  - **Backup** the database for later use
- The auto bencher will do all above for you!

Uploading ElaSQL Binaries

1. Compiles ElaSQL to runnable jars.
2. Names the server and client as ‘server.jar’ and ‘client.jar’ respectively.
3. Creates a directory in `auto-bencher/jars`.
   - The name of the directory does not matter.
4. Uploads the jars to your workstation and put them into the directory.

Configuring ElaSQL

- Next, we need to set the properties files of ElaSQL
  - How many warehouses?
  - Which system to use?
  - How large is the buffer pool?

- But there are too many files to set.
  - vanilladb.properties
  - elasql.properties
  - elasqlbench.properties
  - …. etc.

Parameter Files

- To simplify the configuration, you only have to write down the properties you wish to change.
  - The rest of properties will be filled up using default values.
- The auto bencher will generate properties files and then put your configurations into them.

Setting Parameter for Loading TPC-C

In ‘parameters/loading/tpcc.toml’:

```
[auto_bencher]
jar_dir = "test"
server_count = "2" # without the sequencer
server_client_ratio = "1.0" # client / server
max_server_per_machine = "1"
max_client_per_machine = "2"

[vanilladb]
"org.vanilladb.core.storage.buffer.BufferMgr.BUFFER_POOL_SIZE" = "1024000" # 4GB
"org.vanilladb.core.storage.file.io.IoAllocator.USE_O_DIRECT" = "true"

[vanillabench]
"org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE" = "2"
"org.vanilladb.bench.tpcc.TpccConstants.NUM_WAREHOUSES" = "3"

[elasql]
"org.elasql.server.Elasql.SERVICE_TYPE" = "1"
"org.elasql.remote.groupcomm.client.BatchSpcSender.BATCH_SIZE" = "1"
```

Parameters for the auto bencher

Setting Parameter for Loading TPC-C

In ‘parameters/loading/tpcc.toml’:

```
[auto_bencher]
jar_dir = "test"
server_count = "2" # without the sequencer
server_client_ratio = "1.0" # client / server
max_server_per_machine = "1"
max_client_per_machine = "2"

[vanilladb]
"org.vanilladb.core.storage.buffer.BufferMgr.BUFFER_POOL_SIZE" = "1024000" # 4GB
"org.vanilladb.core.storage.file.io.IoAllocator.USE_O_DIRECT" = "true"

[vanillabench]
"org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE" = "2"
"org.vanilladb.bench.tpcc.TpccConstants.NUM_WAREHOUSES" = "3"

[elasql]
"org.elasql.server.Elasql.SERVICE_TYPE" = "1"
"org.elasql.remote.groupcomm.client.BatchSpcSender.BATCH_SIZE" = "1"
```

Setting Parameter for Loading TPC-C

In ‘parameters/loading/tpcc.toml’:

```
[auto_bencher]
jar_dir = "test"
server_count = "2" # without the sequencer
server_client_ratio = "1.0" # client / server
max_server_per_machine = "1"
max_client_per_machine = "2"

[vanilladb]
"org.vanilladb.core.storage.buffer.BufferMgr.BUFFER_POOL_SIZE" = "1024000" # 4GB
"org.vanilladb.core.storage.file.io.IoAllocator.USE_O_DIRECT" = "true"

[vanillabench]
"org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE" = "2"
"org.vanilladb.bench.tpcc.TpccConstants.NUM_WAREHOUSES" = "3"

[elasql]
"org.elasql.server.Elasql.SERVICE_TYPE" = "1"
"org.elasql.remote.groupcomm.client.BatchSpcSender.BATCH_SIZE" = "1"
```

These will be put in the properties files.

Default Properties

- The default properties files are put in ‘properties’.
- ‘properties/settings.json’ indicates existence and the ids of properties files.
  - If you have a new properties files, just put it in this directory and update `settings.json`.

Starting the Loading Process

```
> cargo run load [DB Name] [Parameter File]
```

- [DB Name]: the name of database
  - We can load multiple databases for different benchmarks.
- [Parameter File]: the path to the parameter file
  - E.g., ‘parameters/loading/tpcc.toml’
- This only has to be done once unless you want to change the testbed.

Result of ‘load’

```
[db-team@netdb10 demo]$ cargo run load tpcc parameters/loading/tpcc.toml
   Compiling auto-bencher v0.1.0 (/home/db-team/slmt/demo)
    Finished dev [unoptimized + debuginfo] target(s) in 1.97s
     Running `target/debug/auto-bencher load tpcc parameters/loading/tpcc.toml`
 INFO  auto_bencher::subcommands::load > Preparing for loading testbed into 'tpcc'...
 INFO  auto_bencher::subcommands::load > Using parameter file 'parameters/loading/tpcc.toml'
 INFO  auto_bencher::preparation       > Preparing the benchmarker directory...
 INFO  auto_bencher::subcommands       > Connecting to machines...
 INFO  auto_bencher::subcommands       > Killing existing benchmarker processes...
 INFO  auto_bencher::threads::server   > Preparing servers...
 INFO  auto_bencher::threads::server   > All servers are ready.
 INFO  auto_bencher::threads::client   > Starting clients...
 INFO  auto_bencher::threads::client   > All clients are running. Waiting for finishing...
 INFO  auto_bencher::threads           > All clients finished properly. Stopping server threads...
 INFO  auto_bencher::threads           > All threads exits properly.
 INFO  auto_bencher::subcommands::load > Loading testbed finished.
 INFO  auto_bencher                    > Auto Bencher finishes.
```

Outline

- Why do we need a tool?
- Main Workflow
  - Setting up Environments
  - Loading Testbed
  - Benchmarking
- Other Functions

Benchmarking

- Now all things are set, we can start benchmarking.
- We have to…
  - Set the properties files
  - Send jar files to all machines
  - Start up servers and clients
  - Check if they finish
  - Collects the results from clients
- Again, the auto bencher will do all above for you!

Setting Parameters (Example)

In ‘parameters/tpcc.toml’:

```
[auto_bencher]
jar_dir = "test"
server_count = “2" # without the sequencer
server_client_ratio = "1.0" # client / server
max_server_per_machine = "1"
max_client_per_machine = "2"

[vanilladb]
"org.vanilladb.core.storage.buffer.BufferMgr.BUFFER_POOL_SIZE" = "1024000" # 4GB
"org.vanilladb.core.storage.file.io.IoAllocator.USE_O_DIRECT" = "true"

[vanillabench]
"org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE" = "2"
"org.vanilladb.bench.BenchmarkerParameters.WARM_UP_INTERVAL" = "30000"
"org.vanilladb.bench.BenchmarkerParameters.BENCHMARK_INTERVAL" = "60000"
"org.vanilladb.bench.BenchmarkerParameters.NUM_RTES" = "10"

"org.vanilladb.bench.tpcc.TpccConstants.NUM_WAREHOUSES" = "3"

[elasql]
"org.elasql.server.Elasql.SERVICE_TYPE" = "1"
"org.elasql.remote.groupcomm.client.BatchSpcSender.BATCH_SIZE" = "5"
```

Important!

Jobs

- Unlike loading testbed, we can assign multiple parameter values in a file at a time.

- The auto bencher will treat each combination of parameters as a job.
  - It runs each job sequentially and puts its report in an independent directory.

Setting Multiple Jobs (Example)

In ‘parameters/tpcc.toml’ **(4 combinations => 4 jobs)**:

```
[auto_bencher]
jar_dir = “impl1 impl2"
server_count = “2" # without the sequencer
server_client_ratio = "1.0" # client / server
max_server_per_machine = "1"
max_client_per_machine = "2"

[vanilladb]
"org.vanilladb.core.storage.buffer.BufferMgr.BUFFER_POOL_SIZE" = "1024000" # 4GB
"org.vanilladb.core.storage.file.io.IoAllocator.USE_O_DIRECT" = "true"

[vanillabench]
"org.vanilladb.bench.BenchmarkerParameters.BENCH_TYPE" = "2"
"org.vanilladb.bench.BenchmarkerParameters.WARM_UP_INTERVAL" = "30000"
"org.vanilladb.bench.BenchmarkerParameters.BENCHMARK_INTERVAL" = "60000"
"org.vanilladb.bench.BenchmarkerParameters.NUM_RTES" = "10 20"

"org.vanilladb.bench.tpcc.TpccConstants.NUM_WAREHOUSES" = "3"

[elasql]
"org.elasql.server.Elasql.SERVICE_TYPE" = "1"
"org.elasql.remote.groupcomm.client.BatchSpcSender.BATCH_SIZE" = "5"
```

Two implementations of ElaSQL

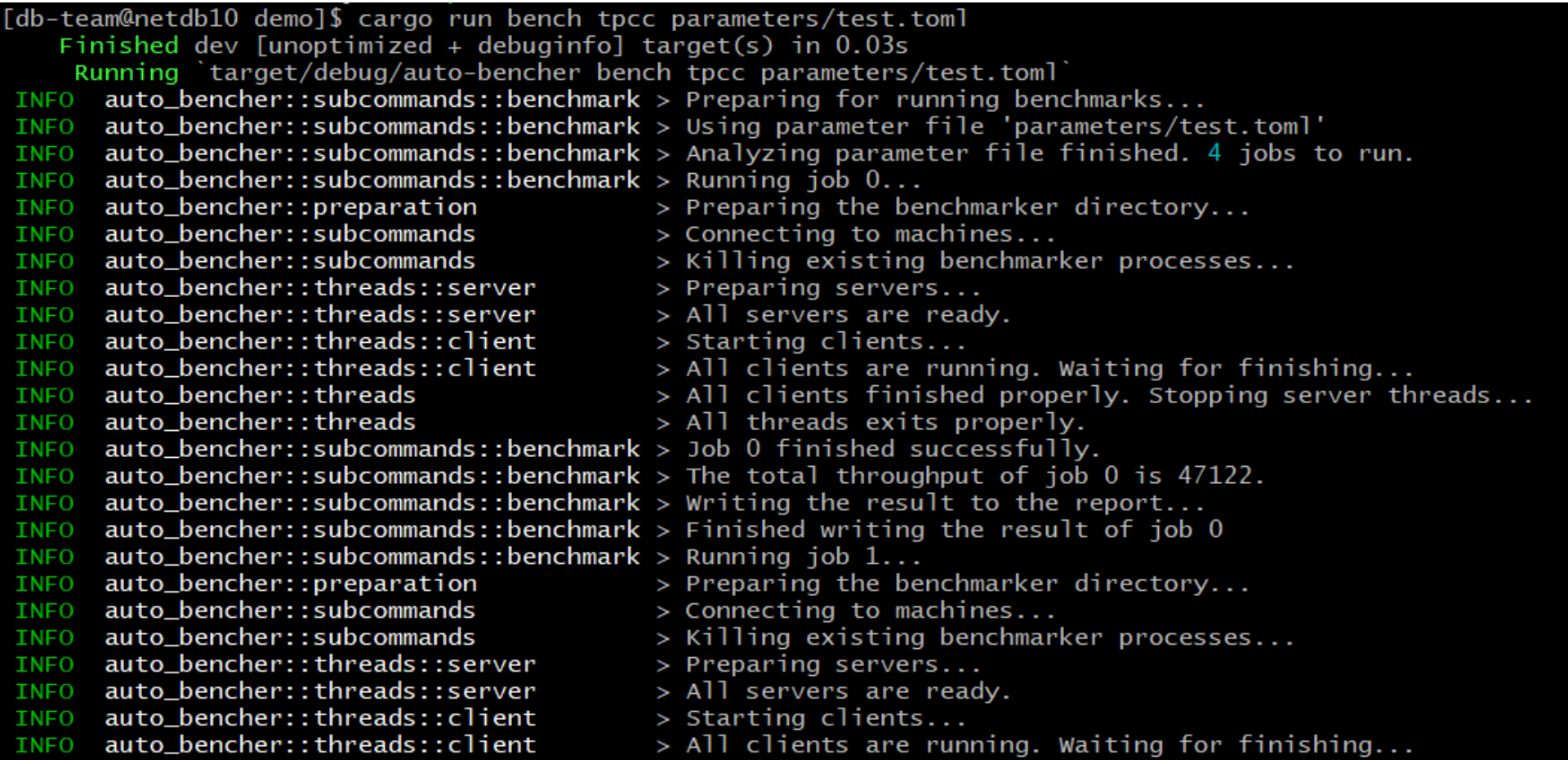
Two #s of RTEs

Starting the Benchmarking

```
> cargo run bench [DB Name] [Parameter File]
```

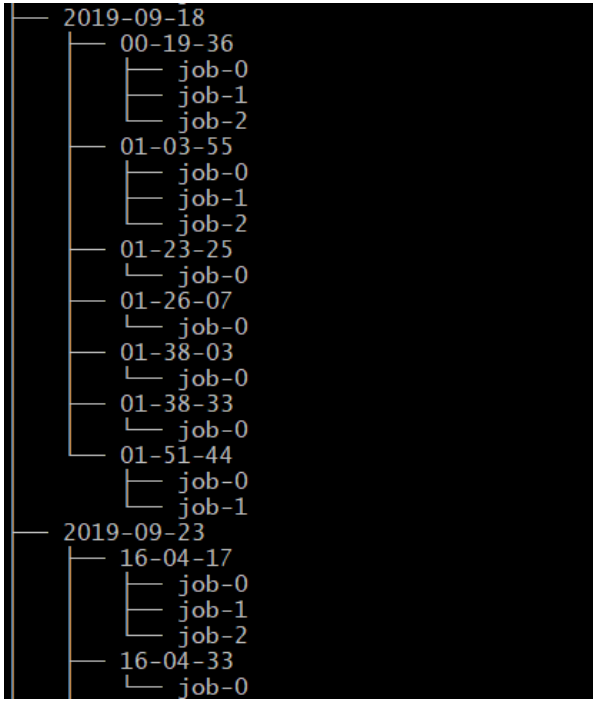
- [DB Name]: the name of database
  - The specified database must be loaded before benchmarking.
- [Parameter File]: the path to the parameter file
  - E.g., ‘parameters/tpcc.toml’

Screenshot


[db-team@netdb10 demo]$ cargo run bench tpcc parameters/test.toml
Finished dev [unoptimized + debuginfo] target(s) in 0.03s
Running `target/debug/auto-bencher bench tpcc parameters/test.toml`
INFO auto_bencher::subcommands::benchmark > Preparing for running benchmarks...
INFO auto_bencher::subcommands::benchmark > Using parameter file 'parameters/test.toml'
INFO auto_bencher::subcommands::benchmark > Analyzing parameter file finished. 4 jobs to run.
INFO auto_bencher::subcommands::benchmark > Running job 0...
INFO auto_bencher::preparation > Preparing the benchmarker directory...
INFO auto_bencher::subcommands > Connecting to machines...
INFO auto_bencher::subcommands > Killing existing benchmarker processes...
INFO auto_bencher::threads::server > Preparing servers...
INFO auto_bencher::threads::server > All servers are ready.
INFO auto_bencher::threads::client > Starting clients...
INFO auto_bencher::threads::client > All clients are running. Waiting for finishing...
INFO auto_bencher::threads > All clients finished properly. Stopping server threads...
INFO auto_bencher::threads > All threads exits properly.
INFO auto_bencher::subcommands::benchmark > Job 0 finished successfully.
INFO auto_bencher::subcommands::benchmark > The total throughput of job 0 is 47122.
INFO auto_bencher::subcommands::benchmark > Writing the result to the report...
INFO auto_bencher::subcommands::benchmark > Finished writing the result of job 0
INFO auto_bencher::subcommands::benchmark > Running job 1...
INFO auto_bencher::preparation > Preparing the benchmarker directory...
INFO auto_bencher::subcommands > Connecting to machines...
INFO auto_bencher::subcommands > Killing existing benchmarker processes...
INFO auto_bencher::threads::server > Preparing servers...
INFO auto_bencher::threads::server > All servers are ready.
INFO auto_bencher::threads::client > Starting clients...
INFO auto_bencher::threads::client > All clients are running. Waiting for finishing...

Where is the Result?

- The auto bencher will collects the reports generated by clients and put them in a directory under ‘reports’.
  - With the date and the time



Reports

- In addition to clients' reports, the auto bencher generates two more reports.
  - Timelines for each job.
    - job-x-timeline.csv (x = job number)
  - Throughput summary for each parameter combination.
    - throughput.csv

Screenshot for a Timeline Report

	A	B	C	D	E	F	G	H
1	time	throughput						
2	30	1218						
3	33	2268						
4	36	2356						
5	39	2530						
6	42	2588						
7	45	2389						
8	48	2249						
9	51	2415						
10	54	2194						
11	57	2242						
12	60	2369						
13	63	2382						
14	66	2462						
15	69	2357						
16	72	2251						
17	75	2294						
18	78	2171						

Screenshot for a Throughput Report

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	job_id	jar_dir	max_client	max_serve	server_clie	server_cou	BATCH_S	SERVICE_	BENCHMA	BENCH_T	NUM_RTE	WARM_U	NUM_WA	BUFFER_	USE_O_DI	throughput
2	0	test	2	1	1	2	5	1	60000	2	10	30000	3	1024000	TRUE	47122
3	1	test	2	1	1	2	5	1	60000	2	20	30000	3	1024000	TRUE	66235
4	2	test	2	1	1	2	5	2	60000	2	10	30000	3	1024000	TRUE	43027
5	3	test	2	1	1	2	5	2	60000	2	20	30000	3	1024000	TRUE	71196
6																
7																
8																
9																
10																
11																
12																
13																

Outline

- Why do we need a tool?
- Main Workflow
  - Setting up Environments
  - Loading Testbed
  - Benchmarking
- Other Functions

all-exec

```
> cargo run all-exec [CMD]
```

- This command executes the shell command you specified on all the machines.
- [CMD]: The command to execute
  - E.g., ‘echo meow’

Screenshot of ‘all-exec’

```
[db-team@netdb10 demo]$ cargo run all-exec 'free'
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/auto-bencher all-exec free`
 INFO  auto_bencher::subcommands::all_execute > Executing the command on 192.168.1.100
              total        used        free      shared  buff/cache   available
Mem:       32640132     1585592    29199624      992696     1854916    29782184
Swap:       8191996           0     8191996

 INFO  auto_bencher::subcommands::all_execute > Executing the command on 192.168.1.11
              total        used        free      shared  buff/cache   available
Mem:       32677424     6974084    20518120     1566272     5185220    23839952
Swap:       4079612           0     4079612

 INFO  auto_bencher::subcommands::all_execute > Executing the command on 192.168.1.12
              total       used       free     shared    buffers     cached
Mem:       32678700   12040140   20638560    1696884        888    3218784
-/+ buffers/cache:    8820468   23858232
Swap:       4079612          0    4079612

 INFO  auto_bencher::subcommands::all_execute > Executing the command on 192.168.1.13
              total       used       free     shared    buffers     cached
Mem:       24421388    8542144   15879244    1160720          0    4431296
-/+ buffers/cache:    4110848   20310540
Swap:       4079612      52736    4026876

 INFO  auto_bencher                          > Auto Bencher finishes.
```

pull

```
> cargo run pull [PATTERN]
```

- This command pulls the files with the pattern that you give from the workspaces of all the machines.

- [PATTERN]: The pattern to match
  - E.g., ‘*.log’

Screenshot of ‘pull’

```
[db-team@netdb10 demo]$ cargo run pull '*.log'
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/auto-bencher pull '*.log'`
 INFO  auto_bencher::subcommands::pull > Pulling files from 192.168.1.100...
 INFO  auto_bencher::subcommands::pull > Pulling files from 192.168.1.11...
 INFO  auto_bencher::subcommands::pull > Pulling files from 192.168.1.12...
 INFO  auto_bencher::subcommands::pull > Pulling files from 192.168.1.13...
 INFO  auto_bencher                    > Auto Bencher finishes.
[db-team@netdb10 demo]$ tree pulls/
pulls/
├── client-0.log
├── client-1.log
├── server-0.log
├── server-1.log
└── server-seq.log

0 directories, 5 files
```

Debug Messages

- In addition to basic information, the auto bencher can show more detailed logs.

- To show debug-level message:

```
> RUST_LOG=auto_bencher=DEBUG cargo run [ACTION]
```

- To show trace-level message:

```
> RUST_LOG=auto_bencher=TRACE cargo run [ACTION]
```

Screenshot of Debug Messages

```
[db-team@netdb10 demo]$ RUST_LOG=auto_bencher=DEBUG cargo run bench tpcc parameters/test.toml
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/auto-bencher bench tpcc parameters/test.toml`
 INFO  auto_bencher::subcommands::benchmark > Preparing for running benchmarks...
 INFO  auto_bencher::subcommands::benchmark > Using parameter file 'parameters/test.toml'
 INFO  auto_bencher::subcommands::benchmark > Analyzing parameter file finished. 4 jobs to run.
 INFO  auto_bencher::subcommands::benchmark > Running job 0...
 INFO  auto_bencher::preparation            > Preparing the benchmarker directory...
 INFO  auto_bencher::subcommands            > Connecting to machines...
 INFO  auto_bencher::subcommands            > Killing existing benchmarker processes...
 INFO  auto_bencher::threads::server        > Preparing servers...
DEBUG auto_bencher::connections::server     > Sending benchmarker to sequencer...
DEBUG auto_bencher::connections::server     > Sending benchmarker to server 1...
DEBUG auto_bencher::connections::server     > Sending benchmarker to server 0...
DEBUG auto_bencher::connections::client     > No previous results are found on '192.168.1.13'
DEBUG auto_bencher::connections::server     > Resetting the db of server 1...
DEBUG auto_bencher::connections::server     > Resetting the db of server 0...
DEBUG auto_bencher::connections::server     > Starting server 0...
DEBUG auto_bencher::connections::server     > Starting server 1...
DEBUG auto_bencher::connections::server     > Starting sequencer...
DEBUG auto_bencher::threads::server         > The sequencer is ready.
DEBUG auto_bencher::threads::server         > Server 1 is ready.
DEBUG auto_bencher::threads::server         > Server 0 is ready.
 INFO  auto_bencher::threads::server        > All servers are ready.
 INFO  auto_bencher::threads::client        > Starting clients...
DEBUG auto_bencher::connections::client     > Starting client 1...
DEBUG auto_bencher::connections::client     > Starting client 0...
DEBUG auto_bencher::connections::client     > Client 0 is running.
DEBUG auto_bencher::connections::client     > Client 1 is running.
 INFO  auto_bencher::threads::client        > All clients are running. Waiting for finishing...
```