A Step-by-step Tutorial to Reproduce *MonEx* Use Case Experiments

This file contains the artifacts of the *MonEx* paper experiments. In the same folder of this file, you can find the experiments folders that contain the datasets and some dependent scripts.

1 Disk & Power Usage of a *MongoDB* Cluster

1.1 Goals of experiment

Using *MonEx* framework to monitor the disk & the power usage of a *MongoDB* cluster while running an indexation workload.

1.2 Fully Reproducible?

Yes, if you are using the *Grid'5000* testbed or any testbed that allows to consult the power usage of the nodes. Otherwise, you could only reproduce the part of monitoring the disk usage.

1.3 Requirements

1.3.1 Hardware Requirements

This experiment needs a cluster of at least five nodes. Three nodes serve as shard nodes while another node holds the configuration server and the *mongo* router daemons. The fifth node is reserved for the *MonEx* framework. The main point during reproducing this experiment relies on the way the experiment is monitored using *MonEx*, but not on its results by itself. Any hardware specifications could be used, so the only impact might be obtaining a different execution time.

1.3.2 Software Requirements

- *MonEx* framework configured with *Prometheus* as a data collector.
- *Prometheus* default node exporter, and snmp exporter (for power usage)
- *MongoDB* version 3.2, with *WiredTiger* search engine
- *Ruby (1.9 or above)*, and *mongo(v1.1), yaml* ruby-gems to generate data for *MongoDB*

1.3.3 Run-time Environment

Debian jessie is used as a run-time environment.

1.4 Installation

For simplicity reasons, we refer to the experiment nodes as follows:

- node_[1-3]: the first, second & third shard nodes of *MongoDB*
- node_4: the node that holds the config. server and the router of *MongoDB*
- node_monex: the node that will be used to hold the *MonEx* framework.

1.4.1 *MongoDB* Installation & Configuration

⇒ Installing MongoDB v2.3 on the nodes_[1-4]:

```
$ apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
$ echo "deb http://repo.mongodb.org/apt/debian jessie/mongodb-org/3.2 main" \
  | tee /etc/apt/sources.list.d/mongodb-org-3.2.list
$ apt-get update
$ apt-get install -y mongodb-org
```

⇒ Configure the *WiredTiger* to be used as the default storage engine in *MongoDB*. This is done by editing the file **/etc/mongod.conf** on the nodes_[1-4], by adding the following line:

```
 storage.engine: wiredTiger
```

You must also modify the *bindIp* line from the same file to allow the nodes communication:

```
 bindIp: 127.0.0.1 => bindIp: 0.0.0.0
```

⇒ Run the different *MongoDB* machines as the following:
- run the follwing commands on the node_4:

```
mkdir ~/mongodb && chown mongodb:mongodb ~/mongodb  // create a folder for the config server
nohup mongod --configsvr --dbpath ~/mongodb --port 27019 &  // to start a config serv. daemon
nohup mongos --configdb [IP_NODE_4:27019] &   // to start a mongo router daemon
```

- run the following command on the nodes_[1-3]:

```
 nohup mongod --config /etc/mongod.conf --shardsvr --port 27017 &
```

⇒ you need to configure the cluster and to have a sharded data collection, so we run the following commands on the node_4:

```
$ mongo
mongos> sh.addShard("[IP_NODE_1:27017]") // repeat this step for node_2 & node_3
mongos> use db1   // create db1 database and jump inside
mongos> sh.enableSharding("db1")   // indicating that the db1 could be distributed
mongos> db.c1.ensureIndex({_id: "hashed"})  // create a free collection c1
mongos> sh.shardCollection("db1.c1", {_id:"hashed"})  // sharding the free collection c1
```

⇒ Create and inject a data into the configured cluster. To do so, you need to use the provided file (indexing_on_MongoDB/generateDumpsDataForMongoDB.rb) for generating data (20 million documents by default, about 80Gbyte). The script could be run on a standalone *MongoDB* and then you could create a dump of the generated data in order to be injected into our cluster using the *mongo* router. The commands of this steps are as follows:

```
$ gem install mongo v 1.1  & gem install yaml   // install the required gems
$ ruby generateDumpsDataForMongoDB.rb  // on a machine running a mongod daemon
$ mongodump  --db db1 --collection c1   // export the collection data into a bson file
```

On the node_4, you execute the following command to inject the created data into our cluster

```
$ mongorestore -d db1 -c c1  [BSON_FILE]   // inject and evenly distribute the data
```

At the end of this step, *MongoDB* is installed and configured correctly.

1.4.2 *MonEx* Installation

⇒ Clone *MonEx* and install its dependencies on the *MonEx* _node as follows:

```
git clone https://github.com/madynes/monex.git   // clone MonEx
apt install python3-pip python3-flask python3-requests r-base  // install dependencies
pip3 install flask-cors
apt-get install -t jessie-backports prometheus    // install prometheus for debian Jessie
systemctl stop prometheus
```

⇒ Install the prometheus node exporter and and curl on the nodes_[1-3] as follows:

```
apt install prometheus-node-exporter
```

You could also install the snmp exporter of prometheus if you are interested in reproducing the part of the power usage. You can install its binary version from this page:
`https://github.com/prometheus/snmp_exporter`

⇒ Edit the configuration file of prometheus (**/etc/prometheus/prometheus.yml**) in the node_monex, by adding the IP addresses of the shards (with port 9100) as follows:

```
scrape_timeout: 1s
scrape_interval: 1s
 - targets: ['IP_NODE_1:9100', 'IP_NODE_2:9100','IP_NODE_3:9100']
```

⇒ On the same node (node_monex), start prometheus, and then start the *MonEx* server with the default configuration file.

```
systemctl start prometheus
cd monex && ./monex-server monex-server-default.conf
```

1.5 Experiment Workflow

We suppose that *MongoDB* is installed correctly and that *MonEx* server is started as explained in the previous steps. Then, you can execute the following steps.

1. Before running the main workload of the experiment, notify *MonEx* that the indexation workload is to be started in a moment. Thus, the following command must be run on the node_4:

    ```
     curl -X POST MonEx_IP:5000/exp/mongoXP
    ```

2. Run the workload from the same node as follows:

    ```
    $ mongo
    mongos> use db1
    mongos> db.c1.createIndex({randNum:1})
    ```

    This will take several minutes regarding the hardware specification of the shard nodes, and the size of the generated data.

3. When the workload is accomplished, you should notify *MonEx* that the experiment is terminated. So, the following command is run in the node_4 too (you can run it manually when you the previous command is terminated):

    ```
     curl -X PUT MonEx_IP:5000/exp/mongoXP
    ```

4. From the *MonEx*-node, query the server to obtain a dataset containing the target metrics of the experiment. The commands to obtain the disk usage:

    ```
    curl -X GET -H "Content-Type: application/json" 127.1:5000/exp/mongoXP \
    -d '{"query":"rate(node_disk_io_time_ms{device=\"sda\"}[10s])", \
    "server":"prometheus","type":"duration","labels":["instance"]}' > dataset_hdd.csv
    ```

    and the command to obtain the power usage (if this are reproducing this part) metrics is:

    ```
    curl -X GET -H "Content-Type: application/json" 127.1:5000/exp/mongoXP \
    -d '{"query":"pdu","server":"prometheus","type":"duration", \
    "labels":["outlet","instance"]}' > dataset_pdu.csv
    ```

5. You can directly draw publishable figures from the obtained datasets. For example, to draw only the figure of the disk usage, you could run:

    ```
    $ cd monex
    $ ./monex-draw -F dataset_hdd.csv
    -x "Time (sec)" -y "Disk utilization (%)" -l
    ```

    For drawing a figure for the disk & power usage, you execute the following command; it is used for producing the *Fig. 2-b* that is appeared in the *MonEx* paper.

    ```
    $ ./monex-draw -F dataset_hdd.csv -F2 dataset_pdu.csv
    -c node1,node2,node3  -c2 node1,node2,node3
    -x "Time (sec)" -y "Disk utilization (%)" -y2 "Power usage (W)" -l
    ```

    Indeed, the datasets for this experiment are available in the experiment folder under the same names (dataset_hdd & dataset_pdu), so you can also run the drawing commands over the provided datasets.

2 Many-nodes Bittorrent Download

2.1 Objective

Monitoring the completion of 100 peers on a torrent network using *MonEx*.

2.2 Fully Reproducible?

Yes, this experiment could be totally reproduced independently from the testbed resources as the Distem emulator is used to emulate the target resources.

2.3 Requirements

3 Hardware Requirements

No specific hardware is needed.

4 Software Requirements

- The Distem emulator
- *Ruby (1.9 or above)*
- *MonEx* configured with *Prometheus* as a data collector.

4.0.1 Run-time Environment

Debian stretch is used as a run-time environment.

4.1 Installation

Distem could be installed on different testbeds, but it is easy to install on the *Grid'5000* testbed. To do so, you need to execute those three commands to reserve the required resources, to deploy Debian9, and to install the Distem emulator:

```
frontend$ oarsub -t deploy -l slash_22=1+{"cluster='grisou'"}nodes=11,walltime=3 -I
frontend$ kadeploy3 -f $OAR_NODE_FILE -e debian9-x64-nfs -k
frontend$ distem-bootstrap -g -U https://github.com/alxmerlin/distem.git
```

Distem installation elects a coordinator node for controlling the rest of resources. Ssh the coordinator as a root for the rest of the experiment. Indeed, you will need the provided scripts in the experiment folder, so clone the *MonEx* project and get into the experiment folder:

```
coordiantor$ git clone https://github.com/madynes/monex.git
coordinator$ cd monex/Artifacts_and_datasets/many-nodes_bittorrent/
```

Then, deploy the topology using the provided ruby script:

```
coordinator$ ruby deploy_peers.rb
```

This script deploys 101 peers running Transmission (100 leechers, 1 seeder) and deploys also a tracker. It also throttle the bandwidth between peers (you can check details on Distem website: `http://distem.gforge.inria.fr/`). The script also starts an exporter on each peer (exporter.py). Thanks to the exporter instances, the torrent completion is being available to Prometheus.

Your are ready now to install Prometheus and *MonEx* directly on the coordinator.

```
coordinator$ apt install python3-pip python3-flask python3-requests r-base prometheus
coordinator$ pip3 install flask-cors
coordinator$ systemctl stop prometheus
```

Start Prometheus and *MonEx* with the provided configuration files. You will need to open two other sessions to do so.

```
coordinator$ prometheus -config.file prometheus-config.yml
coordinator$ ../../monex-server monex_server.conf  //from inside the experiment folder
```

The *Prometheus* configuration file contains the list of peers and the *MonEx* configuration file defines the target (Prometheus) and the target metrics. Specifying the metrics here is not necessary but it is more convenient than being given when querying *MonEx*.

4.2 Experiment workflow

You are now ready to start the experiment as follows:

1. This step is to start the experiment and notifies *MonEx* that the experiment is started. The provided script does that:

   ```
   coordinator$ ./start_xp.sh 127.1:5000
   ```

   This script makes the file available to the seeder and starts a new leecher every 4 seconds. You can configure *Grafana* to monitor the experiment in real-time (see *MonEx* documentation for more info.).

2. At the end of the experiment, you must run this command to notify *MonEx* that the experiment is accomplished.

   ```
   coordinator$ curl -X PUT 127.1:5000/exp/peerxp
   ```

3. This command allows you to obtain the target metrics if the metrics are not listed on the configuration file:

   ```
   coordinator$ curl -X GET -H "Content-Type: application/json" 127.1:5000/exp/peerxp \
   -d '{"query":"percent","server":"prometheus", \
   "type":"duration", "labels":["node"]}' > dataset.csv
   ```

   But since you have specified the metric in the configuration file of MonEx, you can just execute this lightweight command:

   ```
   coordinator$ curl -X GET -H "Content-Type: application/json" 127.1:5000/exp/peerxp \
   -d '{"metric":"completion","type":"duration"}' > dataset.csv
   ```

4. Moreover, *MonEx*-draw can connect directly to the server to plot a figure, as we done for the *Fig. 3* in the *MonEx* paper.

   ```
   coordinator$ ../../monex-draw -S 127.1:5000 --metric "completion" --exp "peerxp" \
   --type "duration" -y "Completion" -x "Time (sec)" -n
   ```

5 Input/output offset sequences experiment

5.1 Objective

Using *MonEx* framework to monitor how the access patterns are uncovered over a given file when applying a random access workload. An *eBPF* program is used to uncover the access patterns, while *Fio benchmark* is used to generate the random access workload. The *eBPF* program is **not publicly available** as its paper is not yet published. Thus, it is only possible to repeat the analysis of this experiment. However, all steps to reproduce the experiment are listed below.

5.2 Requirements

5.2.1 Hardware Requirements

Only one node is needed. Any Linux distribution with a kernel version greeter than v3.4 should be used, to allow running the *eBPF* virtual machine. Additionally, no special hardware is needed.

5.2.2 Software Requirements

- Fio Benchmark v2.1.3
- eBPF virtual machine
- eBPF script to construct the access patterns (not available)x
- *MonEx* configured with *InfluxDB* as a data collector

5.2.3 Run-time Environment

Debian jessie is used as a run-time environment.

5.3 Installations

- Fio can be installed using *apt* on debian8:

  ```
  apt-get install -y fio
  ```

- *eBPF* virtual machine could be installed as a part of the *IOvisor BCC project*: `https://github.com/iovisor/bcc` (see install.md inside BCC project page).

5.4 Experiment workflow

The experiment is performed as follows:

1. We create a file using *dd* command as follows:

   ```
   $ dd if=/dev/zero of=testFile bs=16 count=9175040  // create 147 MByte file
   ```

2. We start our *eBPF* program that listens inside the kernel to new I/O requests accessing the target file

3. Notify MonEx that the experiment will begin in a moment:

   ```
   $ curl -X POST MonEx_IP:5000/exp/accessPattern
   ```

4. Start the main workload of accessing *testFile* randomly:

   ```
   $ fio --name=testFile --iodepth=16 --rw=read  --direct=0 --numjobs=1 --group_reporting
   ```

5. After the termination of the workload, notify *MonEx* that the experiment is finished.

   ```
    curl -X PUT MonEx_IP:5000/exp/accessPattern
   ```

6. *MonEx* could be then quired to obtain a dataset of the target metrics as follows:

   ```
   $ curl -X GET 127.1:5000/exp/accessPattern -H "Content-Type: application/json" -d  \
   '{"measurement":"ebpf","server":"influx","database":"monex","type":"sample"}' > dataset.csv
   ```

7. Use *MonEx*-draw to produce a publishable figure of the experiment, by consuming the provided dataset file:

   ```
   ./monex-draw -F dataset.csv -x "Time (sec)" -y "Offset (bytes)" -p -r "random" -n -s ';'
   ```

   This command is used to produce the *Fig. 4* of the *MonEx* paper.