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

This file contains the experimental artifacts of the use case experiments of the *MonEx* paper. 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 Objective

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

1.2 Requirements

The experiment has an automated script to run on the *Grid'5000* testbed, but we list the necessary steps to allow repeating it elsewhere.

1.2.1 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 last node is reserved for installing the *MonEx* framework. Additionally, the main point during reproducing this experiment is on the way we monitor the experiment using *MonEx* but not its results, so any hardware specifications could be used. Thus, the only impact is that you might obtain a different execution time.

1.2.2 Software Requirements

Many software are necessary:

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

1.3 Installation

For simplicity reasons, we refers 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.

1.3.1 *MongoDB* Installation & Configuration

⇒ Please refer to the *MongoDB* official webpage to install *MongoDB* on the nodes_[1-4] and make sure to stop the *mongod* service on all nodes after the installation:
`https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-debian/`

⇒ 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
```

⇒ Configure the *MongoDB* cluster as the following:
- run the follwing commands on the node_4:

```
nohup mongod --configsvr --port 27019 &  // to start a config serv. daemon
nohup mongos --configdb [node_4_IP:27019] // to start a mongo router daemon
```

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

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

⇒ We need to configure the cluster and to have a sharded data collection, so we run the following commands on the node_4 using *mongo* daemon:

```
$ mongos
mongos> sh.addShard([node_1:IP]) // 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, we 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 we could create a dump of the generated data in order to be injected into our cluster using the *mongo* router. The commands on this steps are as follows:

```
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, we 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.

1.3.2 *MonEx* installation

Clone the *MonEx* github project into the node_monex and install its dependencies as explained in the project page (`https://github.com/madynes/monex`). This step implies installing the default exporter of *Promtheus* into all the data nodes (nodes_[1-3]), and configuring *Prometheus* to regularly scrape those data nodes at the rate of 1 second.

1.4 Experiment workflow

We suppose that *MongoDB* is installed correctly and that *MonEx* server is started as explained in the previous steps. Then, we 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. We then run the workload from the same node as follows:

   ```
   $mongos
   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, we should notify *MonEx* that the experiment is terminated. So, the following command is run dynamically from the experiment script (you can run it manually when you the previous command is terminated):

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

4. We can query the *MonEx* server to obtain a dataset containing the target metrics of the experiment. The commands to obtain the disk usage, and the power usage metrics are as follows:

```
curl -X GET -H "Content-Type: application/json" IP:5000/exp/mongoXP \
-d '{"query":"rate(node_disk_io_time_ms{device=\"sda1\", \
"server":"prometheus","type":"duration","labels":["instance"]}' > dataset_hdd.csv

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

5. The obtained datasets from the previous step are available in the experiment folder. So if it is not possible to repeat the experiment, you can use the *MonEx*-draw over the obtained datasets to produce the figures of this experiment as follows:

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

   This command produces the *Fig. 2-b* that is appeared in the *MonEx* paper.

2 Many-nodes Bittorrent Download

2.1 Objective

We want to monitor the completion of 100 peers on a torrent network using MonEx.

2.2 Requirement

This experiment relies on Distem which is easy to deoply on Grid'5000, but it should be repeatable on any testbed.

2.3 Installation

To setup this experiment, we need to install Distem on 11 machine. On Grid'5000, it is done as follows:

```
frontend> oarsub -t deploy -l slash_22=1+{"cluster='grisou'"}nodes=11,walltime=3 -I
frontend> kadeploy3 -f $OAR_NODE_FILE -e jessie-x64-nfs -k
frontend> distem-bootstrap
```

On the coordinator, we deploy the topology using the provided ruby script:

```
coordinator> ruby deploy_peers.rb
```

This script deploy 101 virtual node running Transmission (100 leechers, 1 seeder) and a tracker. It also throttle the bandwidth between peers (you can check details on Distem website: http://distem.gforge.inria.fr/). We will 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
coordiantor> git clone https://github.com/madynes/monex.git
```

Start Prometheus and MonEx with the provided configuration files

```
coordinator> prometheus -config.file prometheus-config.yml
coordinator> ./monex-server monex_server.conf
```

The prometheus configuration file contains the list of peers and the MonEx configuration file defines the target (prometheus) and a metric. Specifying the metric here is not necessary but it is more convintent.

2.4 Experiment workflow

We are now ready to start the experiment, the provided script will start the experiment and signal MonEx

```
./start_xp.sh 127.0.0.1:5000
```

This script make the file available to the seeder and start a leecher every 4 seconds and start an exporter on every virtual node (exporter.py). At the end of the experiment, to get the metric, we use :

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

But since we have specify the metric in the configuration file of MonEx, we can juste use:

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

Moreover, monex-draw can connect directly to the server to plot a figure:

```
./monex-draw -S 127.0.0.1:5000 --metric "completion" --exp "peer" --type "sample" \
-y "Completion" -x "Time (sec)" -n
```

3 Input/output offset sequences experiment

3.1 Objective

Using *MonEx* framework to monitor how the access patterns over a given file are uncovered 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 out of scope of this tutorial as its paper is not yet submitted. Thus, it is only possible to repeat the analysis of this experiment.

3.2 Requirements

3.2.1 Hardware Requirements

The experiment must run on any Linux distribution with a kernel version greeter than v3.4 to allow running *eBPF* virtual machine. Additionally, no special hardware is needed.

3.2.2 Software Requirements

- Fio Benchmark v2.1.3
- eBPF virtual machine
- *MonEx* configured with *InfluxDB* as a data collector

3.3 Installations

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

  ```
  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 the project page).

3.4 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:

   ```
   \begin{verbatim} curl -X PUT 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/myxp -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.