**CORTX- Performance**

**Guide to MongoDB and PerfPro Database Schema**

Updated on 30 March 2021

1. About database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases. MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB has high availability through built-in replication and failover. It is very easy to scale horizontally and add more collections. Insertion, deletion and selection has least time complexity. It has a well formatted built-in API structure with fully elastic database.

1. Structures and Collections

Since MongoDB is designed for web-based applications, it is a document-based NoSQL database. There are two ways to access a database, direct access and indirect access. In direct access one need the database on the physical machine on which one is working on. Whereas in Indirect database, the database mounted location can be accessed using database URL.

In this project, there Integrated central server has the parent database storage. The URL format is unified across official drivers from MongoDB with some options. Following is the generalized format.

mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]

1. **mongodb://** is a required prefix to identify that this is a string in the standard connection format.
2. **username:password@** is optional. If given, the driver will attempt to login to a database after connecting to a database server.
3. **host1** is the only required part of the URI. It identifies either a hostname, IP address, or unix domain socket.
4. **:portX** is optional and defaults to :27017 if not provided.
5. /database is the name of the database to login to and thus is only relevant if the username:password@ syntax is used. If not specified the “admin” database will be used by default.
6. **?options** are connection options.

**Collection** is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. **Documents** within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose. A document is JSON like a set of key value pairs. They have a **dynamic** schema. They are case sensitive and may hold different types of data.

Following table shows a comparison of MongoDB with RDBMS.

|  |  |
| --- | --- |
| RDBMS | MongoDB |
| Table | Collection |
| Tuple / Row | Document |
| Column | Field |
| Table Join | Embedded Documents |
| Primary Key | Default key ( \_id provided itself by DB ) |
| Server: Mysqld/Oracle | mongod |
| Client: mysql/sqlplus | mongo |

*Table 1. Comparison of RDBMS Vs MongoDB*

1. Querying Database

Pymongo is a python library for using mongoDB in python. pymongo.MongoClient() module connects to host Database.

client = pymongo.MongoClient(hostURL)

db = client[‘database\_name’]

// selects the database named ‘database\_name’ from the client

col = db.results // connects to collection named results

For elementary operations like insert, delete, select and find there are built in APIs for MongoDB. They are as follows.

1. col.insert\_one(data) – It inserts single document in database. ‘data’ is a dictionary of all key-value pairs to be saved as a document in the collection named ‘col’.
2. col.insert\_many(documents) – It inserts more than one documents at a time. ‘documents’ is a list of all documents / dictionaries.
3. col.find\_one\_and\_update(filter, {"$set" : update}) – It finds a document having key-value pairs mentioned in ‘filter’ dictionary and replaces the key-value pairs mentioned in ‘update’. If more than one matches the ‘filter’ it considers the latest entry.
4. col.find(filter) – It is a function which finds all the documents which matches with filter. It returns mono-iterative cursor location.[2]
5. Shell Commands

* To connect to performance DB remotely, first install mongoDB on your machine. Open command prompt (on windows) or type following in terminal (linux): mongo <db\_url>

e.g. mongo mongodb://sampada:password@cftic1.pun.seagate.com:27017,cftic2.pun.seagate.com:27017,apollojenkins.pun.seagate.com:27017/test?authSource=performance\_db&replicaSet=rs0&readPreference=primary&appname=MongoDB%20Compass%20Community&ssl=false

* To show available databases: show dbs
* To connect to database: use <db\_name> e.g. use performance\_db
* To list all collections present: show collections
* To do operations on collections: db.<collection\_name>.<operation>()

e.g. db.configurations.find() //list all the documents

e.g. db.results.find({‘Name’: ‘Hsbench’, ‘Build’: ‘515’}) //specific query

etc.

Use the mongo DB APIs for reference.[1]

1. PerfPro

* **Database location:**

Performance Team’s database is mounted on CFT Integration Central i.e. IC2.

FQDN: cftic2.pun.seagate.com

* **Database URL:**

mongodb://sampada:password@cftic1.pun.seagate.com:27017,cftic2.pun.seagate.com:27017,apollojenkins.pun.seagate.com:27017/test?authSource=performance\_db&replicaSet=rs0&readPreference=primary&appname=MongoDB%20Compass%20Community&ssl=false

* **Current Users and passwords:**

|  |  |
| --- | --- |
| User | Password |
| perfpro | PerfPro |
| sampada | password |
| radha | password |

* **Database: performance\_db**
* **Collection ‘results\_2’** –
* Purpose: stores data from benchmark logs
* Gets Updated from:

s3bench\_DBupdate.py, cosbench\_DBupdate.py, hsbench\_DBupdate.py

* S3Bench Schema:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | String | Benchmark name | S3bench |
| Operation | String | Operation done by benchmark | Write/Read |
| Object\_Size | String | Object sizes used in workload | 1Kb, 4Kb, 100Kb, 1Mb, 5Mb, 36Mb, 64Mb, 128Mb, 256Mb |
| Build | String | CortX build number | e.g. 515 |
| Version | String | Github branch build version | Cortxv1/release/beta |
| Config\_ID | ObjectID | \_id of corresponding configurations entry in ‘configurations’ collection |  |
| HOST | String | S3 machine client of PC |  |
| IOPS | Float | Aggregate Requests per second observed | e.g. 29.0 |
| Latency | Dict | Aggregate latency value of min/max/avg observed | {'Max': 1.78, 'Avg': 1.643, 'Min': 1.041} |
| Log\_File | String | Log file name generated by benchmark |  |
| TTFB | Dict | Time To First Byte observed by client | {'Max': 1.78, 'Avg': 1.643, 'Min': 1.041} |
| Throughput | Float | Throughput in MBps | e.g. 576.67 |
| Timestamp | String | Time when entry has updated | '2020-11-27 17:56:13’ |
| Count\_of\_nodes | Int | Number of nodes taking part in CORTX cluster. | 3 / 6 / 9 / ...n |
| Count\_of\_Clients | Int | Number of clients used to run benchmarks | 1 / 2 / 3 / 4 … n |
| Branch | String | Branch of the build deployed on CORTX cluster. | Main / Release / Custom |

* COSBench Schema:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | String | Benchmark name | Cosbench |
| Operation | String | Operation done by benchmark | write/read |
| Object\_Size | String | Object sizes used in workload | 4 KB, 100 KB, 1 MB, 5 MB, 36 MB, 64 MB, 128 MB, 256 MB |
| Buckets | Int | Number of buckets in the workload | 1/10/50 |
| Objects | Int | Number of objects in each bucket | 1000 for 1 bucket, 100 for 10 & 50 buckets |
| Sessions | Int | Total number of sessions | 100 |
| Build | String | CortX build number | e.g. 515 |
| Version | String | Github branch build version | Cortxv1, release, beta |
| Config\_ID | ObjectID | \_id of corresponding configurations entry in ‘configurations’ collection |  |
| HOST | String | S3 machine client of PC |  |
| IOPS | Float | Aggregate Requests per second observed | e.g. 29.0 |
| Latency | Dict | Aggregate latency value of min/max/avg observed | {'Max': 1.78, 'Avg': 1.643, 'Min': 1.041} |
| Log\_File | String | Log file name generated by benchmark |  |
| TTFB | Dict | Time To First Byte observed by client | {'Max': 1.78, 'Avg': 1.643, 'Min': 1.041} |
| Throughput | Float | Throughput in MBps | e.g. 0.31812 |
| Timestamp | String | Time when entry has updated | '2020-11-27 17:56:13’ |
| Count\_of\_nodes | Int | Number of nodes taking part in CORTX cluster. | 3 / 6 / 9 / ...n |
| Count\_of\_Clients | Int | Number of clients used to run benchmarks | 1 / 2 / 3 / 4 … n |
| Branch | String | Branch of the build deployed on CORTX cluster. | Main / Release / Custom |

* HSBench Schema:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | String | Benchmark name | Hsbench |
| Operation | String | Operation done by benchmark | write/read |
| Object\_Size | String | Object sizes used in workload | 1Kb, 4Kb, 100Kb, 1Mb, 5Mb, 36Mb, 64Mb, 128Mb, 256Mb |
| Buckets | Int | Number of buckets in the workload | 1/10/50 |
| Objects | Int | Total number of objects distributed in all buckets | 1000 for 1 & 10 buckets, 5000 for 50 buckets |
| Sessions | Int | Total number of sessions | 100 |
| Build | String | CortX build number | e.g. 515 |
| Version | String | Github branch build version | Cortxv1, release, beta |
| Bucket\_Ops | List of dicts | Bucket operations provided by HSBench; each list is each bucket ops mode executed by HSBench in sequence | Detailed example below |
| Config\_ID | ObjectID | \_id of corresponding configurations entry in ‘configurations’ collection |  |
| HOST | String | S3 machine client of PC | Client FQDN |
| IOPS | Float | Aggregate Requests per second observed | e.g. 29.0 |
| Latency | Dict | Aggregate latency value of min/max/avg observed | {'Max': 1.78, 'Avg': 1.643, 'Min': 1.041} |
| Log\_File | String | Log file name generated by benchmark |  |
| TTFB | Dict | Time To First Byte observed by client | {'Max': 1.78, 'Avg': 1.643, 'Min': 1.041} |
| Throughput | Float | Throughput in MBps | e.g. 0.31812 |
| Timestamp | String | Time when entry has updated | '2020-11-27 17:56:13’ |
| Count\_of\_nodes | Int | Number of nodes taking part in CORTX cluster. | 3 / 6 / 9 / ...n |
| Count\_of\_Clients | Int | Number of clients used to run benchmarks | 1 / 2 / 3 / 4 … n |
| Branch | String | Branch of the build deployed on CORTX cluster. | Main / Release / Custom |

* BucketOPs list schema:
* There are following modes executed sequentially by HSBench.

|  |  |
| --- | --- |
| BCLR | Initial clearing of objects in buckets if any are present |
| BDEL | Initial deletion of buckets if any of them are present |
| BINIT | Initializing/creating buckets as per the workload |
| PUT | Writing objects into buckets |
| LIST | Listing objects in buckets |
| GET | Reading objects from buckets created initially |
| DEL | Deleting all the objects |
| BCLR | Clearing/deleting newly created buckets during benchmark run |

* For each of the above operations following metrics are available.

|  |  |
| --- | --- |
| Mode | Above bucket operations such as BDEL, BCLR |
| Seconds | Time in seconds took to execute the particular mode |
| Ops | Number of operations covered |
| Mbps | Throughput observed within the particular mode |
| Iops | IOs per second observed |
| MinLat | Minimum latency observed |
| AvgLat | Average latency across the particular mode |
| MaxLat | Maximum latency observed |

* **Collection ‘configurations\_2’ –**
  + Purpose: Store configurations provided by user in config.yml file
  + Gets updated from addConfiguration.py file
  + Schema:

|  |  |  |  |
| --- | --- | --- | --- |
| REIMAGE | String | Weather cluster is reimaged or not | yes/no |
| NODE1 | String | Node1 used during run | FQDN |
| NODE2 | String | Node2 used during run | FQDN |
| NODE3 | String | Node3 used during run | FQDN |
| CLIENT1 | String | S3 client1 used during run | FQDN |
| CLIENT2 | String | S3 client2 used during run | FQDN |
| BUILD\_URL | String | URL to CORTX build location from where iso image is taken |  |
| OS\_TYPE | String | The OS deployed on cluster | CENTOS/ RHEL/ CORTX |
| CLUSTER\_PASS | String | Auth password for both nodes is set to this field if reimage = yes | seagate1 |
| CHANGE\_PASS | String | If true, changes cluster password during execution | yes/no |
| PRVSNR\_CLI\_REPO | String | GitHub URL to CORTX Provisioner which imports CLI commands |  |
| PREREQ\_URL | String | URL to install prerequisites needed for CORTX Provisioner to install & run prvsnr CLI |  |

* **Collection ‘systemresults\_2’ –**
  + Purpose: Collect system related data generated during PerfPro and dump here
  + Gets updated from system\_monitoring/systemMonitoring.py
  + Current available Keys:

['\_id', 'Timestamp', 'benchmark', 'CPU', '%user', '%nice', '%system', '%iowait', '%steal', '%idle', 'kbmemfree', 'kbmemused', '%memused', 'kbbuffers', 'kbcached', 'kbcommit', '%commit', 'kbactive', 'kbinact', 'kbdirty', 'DEV', 'tps', 'rd\_sec/s', 'wr\_sec/s', 'avgrq-sz', 'avgqu-sz', 'await', 'svctm', '%util', 'rtps', 'wtps', 'bread/s', 'bwrtn/s', 'rxpck/s', 'txpck/s', 'rxkB/s', 'txkB/s', 'rxcmp/s', 'txcmp/s', 'rxmcst/s', 'Device', 'Time', 'IFACE', 'HOST']

**References**

1. [Website] Mongo Shell commands manual, MongoDB, <https://docs.mongodb.com/manual/reference/mongo-shell/>
2. [Website] MongoDB Python APIs, MongoDB, <https://docs.mongodb.com/drivers/python>
3. [Website] Mongo DB official manual, MongoDB, <https://docs.mongodb.com/manual/>