EdgeX Data Persistence Project Group - Database Performance Testing

Approaches to comparing the

A. **Use existing harness/benchmark**. We could use an existing benchmarking apparatus like that of Yahoo Cloud Serving Benchmark: https://github.com/brianfrankcooper/YCSB. The Research Gate publication used this to compare Cassandra, MongoDB and Redis. The negative aspect of this approach is that the harness really isn't set up to test SQL databases and it is not tailored to look at the specifics of our use case.
B. **Role our own harness/tests**. This requires us to create a test harness/apparatus, the code that uses each database (core data using Mongo, core data using CouchDB, etc.) and the actual tests (and supporting data) to put into the harness. The positive effect of this is that the tests would presumable be closer to our use case needs and the harness may help lay the foundation for a performance harness for EdgeX for future needs. The obvious down side is that this requires more work.

If adopting option A, the project then entails using the existing benchmark (YSCB) and potentially adapting it to test some of the other databases (CouchDB) and/or including some of our own specific tests not already in the benchmark.

If adopting option B, then we need to create the harness, test code, and tests/data.

Option B Harness

- We'll use the existing Postman blackbox tests for core data (gives us the context we want for normal operations)
- We need a means to execute the Postman tests a multiple of times to get a sense of performance over time and as the repetition hits the system. The harness should be able to call (and measure) the Postman tests up to configurable number of times each within a defined time frame (example: execute a call for 1000 times once every 10 milliseconds. This will allow us to also determine when the system is unable to keep up.
- We need a means to capture stats from each run.
    - memory usage
    - CPU usage
    - size of database on disk before and after (how efficiently does the database store the data)
    - speed to complete the operations
- We need a means to launch multiple instances (100's? – would we ever anticipate more than 100 concurrent client threads against something like core data) of the harness and point to the same database to measure co multi-thread/concurrent requests
- The environment must be controlled well enough as to avoid outside influences or large fluctuations of the performance measure capture.

Option B Code

Core Data is already written to utilize MongoDB fitting our use case. We would require several core data instances be newly created – one each writing to utilize CouchDB, Redis and any other database we want to consider. For purposes of this examination, we would not refactor the Core Data code base

other than to change the database to use under test. Specifically, the following Core Data files would need to be replaced:

- Add a new implementation of interfaces.DBClient interface (@ edgex-go/internal/core/data/interfaces/db.go) as Mongo, Influx, etc. do (@ edgex-go/internal/pkg/db/)
- Add or modify existing configuration to core data to support the new database.
- Update edgex-go/internal/core/data/init.go and edgex-go/internal/pkg/db/db.go – to connect and use the new database type

Option B Tests & Data

The Postman blackbox tests perform simple Event/Reading operations on core data. For example, the existing blackbox tests for core data can perform:

- Write an event and reading which can be used when done many times to measure write speed average and to know when/how the system can no longer maintain throughput needs.
- Read events/readings (return a single instance or a collection of instances based on the complexity of the query) which can be used to understand query speeds.

Additional tests and/or data would need to be created to satisfy our evaluation of the databases for performance concerns based on foreseen use cases.

- Combining the above write and reads simultaneously to understand the impacts of reads writes and vice versa.
- Add tests that incorporate reading/writing an Event with more readings (1 event with 10, 25 or even more readings)
- Seeding the event and reading stores with many documents to understand the impact on reading queries or writing and updating indexes.

The following set of operations should suffice to compare the databases for use in generic EdgeX use cases:

General read/write performance

- Write a simple Event with associate Readings (1 event, 2 readings) x 1000 [1000 is arbitrary, but should be sufficiently large to get a good average]
- Write a complex Event with associated Readings (1 event, 100 readings) x 1000
- Read a simple Event (and associated Readings) based on ID x 1000
- Read a complex Event with associated Readings (1 event, 100 readings) based on ID x 1000
- Read Events/Readings using a complex query (all events for a given device, created within a time rage, and with a temperature reading attached) x 1000

Scale/Load performance

- Have 100 clients (device service emulators) call to write a simple Event with associated Readings (1 event, 2 readings) simultaneously at a frequency that can be modulated. At what frequency does the database (and core data) cease being able to keep up?

- Have 100 clients (device service emulators) call to write a complex Event with associated Readings (1 event, 100 readings) simultaneously at a frequency that can be modulated. At what frequency does the database (and core data) cease being able to keep up?
- Have 100 clients (device service emulators) call to complete a simple query by ID simultaneously at a frequency that can be modulated. What are the response times?
- Have 100 clients (device service emulators) call to complete a complex query with > 100 events simultaneously at a frequency that can be modulated. What are the response times?

Resources

Some open reports completed on database and performance comparisons (please not dates of some of these works as some are rather dated)

- https://www.researchgate.net/publication/261079289_A_performance_comparison_of_SQL_and_NoSQL_databases
- https://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-hbase
- https://www.researchgate.net/publication/281629653_Performance_Comparison_of_NoSQL_Databases_in_Pseudo_Distributed_Mode_Cassandra_MongoDB_Redis
- https://jaxenter.com/evaluating-nosql-performance-which-database-is-right-for-your-data-107481.html
- https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu/
- https://d-nb.info/1132360862/34
- Go implementation of YCSB: https://github.com/pingcap/go-ycsb