Magma Unified SMO OpenSchema Project for Facebook Connectivity

Task 3: Performance Evaluation of Magma Data Lake Implementation

Outline

- Summary and Conclusion
- OpenSchema Deployment & Scalability Requirements
- Magma Data Lake Test Plan
- Benchmarking Magma Data Lake WRITE Performance
- Magma Data Lake Limitation for OpenSchema

Summary Conclusion

Testing Hypothesis: Existing Magma Data lake implementation is not cost effectively scalable to support OpenSchema requirements.

- OpenSchema is a WRITE heavy operation
  - Test results show that as metrics complexity increases, WRITE latency of Magma Data Lake significantly increases too (up to 7x).
    - Fixing this issue by horizontally scaling the cluster, will add additional and *unnecessary cost* since all other components will scale too.
    - Even if the WRITE latency increase is linear, then UEs may fail to successfully push metrics which will cause *retries* which lead to additional cyclical loads or *lost metrics*.
    - Essentially a Prometheus based Magma Data Lake is a monitoring solution and not meant to act as a data lake for complex data structures as OpenSchema requires.
  - Magma Data Lake Comparison test results shows that a simple MongoDB setup preforms much faster WRITEs (average of ~2x faster).
    - Also improving ETL service and tuning DB and cloud infrastructure can reduce this delay even further.
- Is sum, due to Magma Data Lake lower WRITE performance and Architecture limitations, it is not recommended as a suitable solution for the SMO OpenSchema and its scalability requirements and validates the need for a more a scalable database solution design.

OpenSchema Deployment Requirements

- Deployment
  - On-Prem or Cloud Deployment
  - Server Deployment < 1 day
  - Uses all Open-Source tools (Low Cost)
  - Client SDK for Android and iOS – Integration < 1 day
  - Implemented as an optional extension to Magma Unified SMO

- Device Registration
  - Provisioning by UUID – Issued shared secret key
  - Optional extension API for UE provisioning

OpenSchema Scalability Requirements

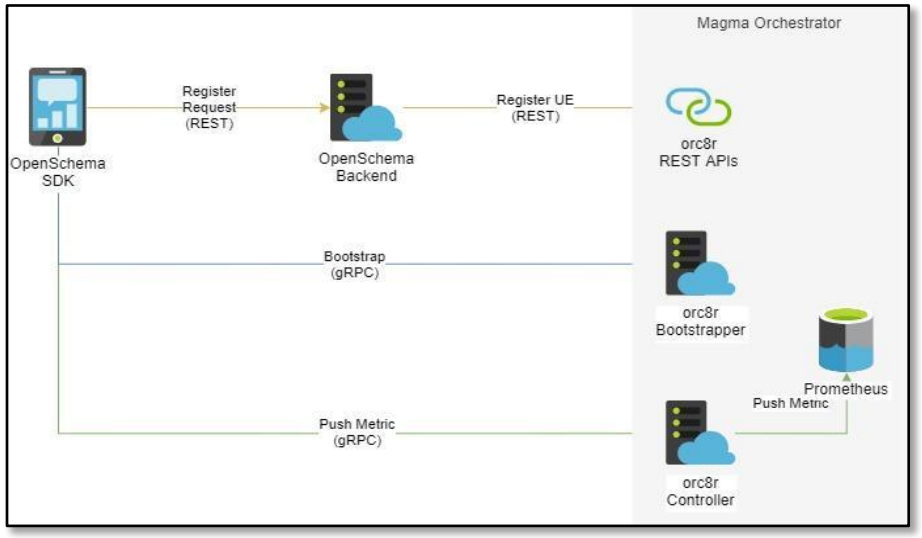
- Scaling Assumptions
  - Average 200KB transmitted per device
  - 30 million active devices per day
- Collection & Storage
  - Connections per Hour: 7.5M (Average) / 15M (Peak)
  - Average data stored per day: 6 TB
  - Ability to horizontally scale (elasticity on collection and storage)
- Data Validation
  - Format – match SDK Version
  - Size – limit to theoretical max per device / per day

Performance Test Plan

- Benchmark Magma Data-Lake WRITE Operation
  - Motivation: OpenSchema is WRITE heavy operation.
    - Hundreds of thousands of UEs,
    - Push metrics multiple times a days,
    - Each time 100s of metrics, 100KBs. Potentially TBs of WRITE per day.
  - Test 1: Test Magma Data Lake WRITE performance for variety of metrics complexities.
  - Test 2: Compare Magma Data Lake WRITE performance against MongoDB.
- Review and Analysis of Prometheus Based Magma Data Lake architecture and its limitation for OpenSchema

Test 1: Magma Data Lake WRITE performance(Setup)

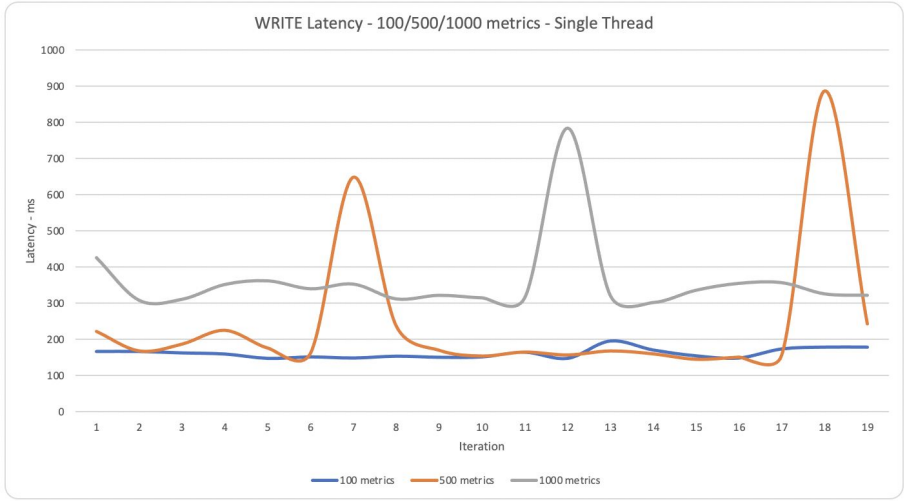
- Fig and Table show the setup and configuration used for this test
- OpenSchema SDK used a custom backend which allowed for automatic UE registration.
- The OpenSchema SDK was modified for pushing payloads of 100, 500, 1000, and 5000 metrics.

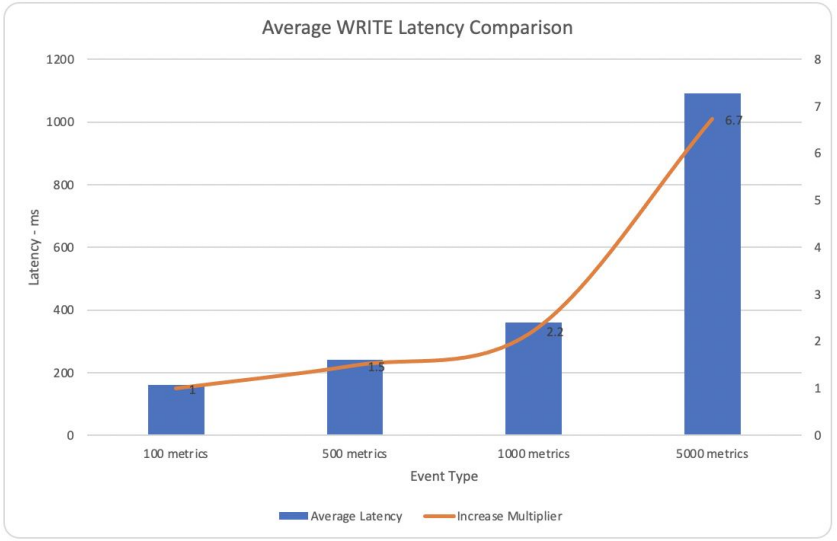


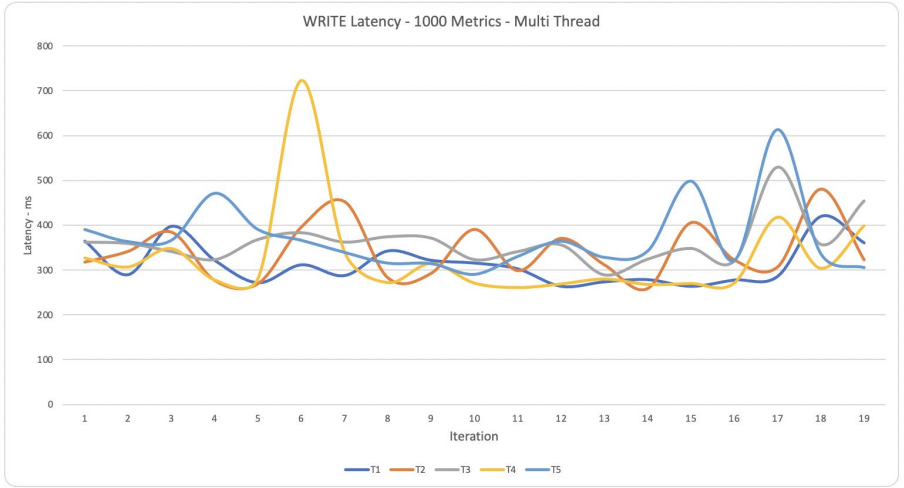
Magma Data Lake	Prometheus within Magma Orc8r
Magma Controller	Magma Orc8r 3x t3.large (23.22.121.164, 54.174.41.72, 52.72.197.188)
Communication method	gRPC Interface
Client device	Google Pixel 3a XL (Android 9)
KPI Collection Tools	Android logcat

Test 1: Magma Data Lake WRITE performance(Results)

- Graphs below compares WRITE latency of multiple metrics events in Magma Data Lake
- Summary of Observations:
  - Results are relatively consistent for 100 metrics. For more complex events with 500-1000 metrics we often see spikes in WRITE latency
  - In multi threaded test latency is relatively consistent(85% of results are within one standard deviation of the average)
  - As the metrics get more complex average latency significantly increases: 1000 metrics latency is 2.2x of 100 metrics latency and 5000 metrics latency is 6.7x times 100 metrics latency.
  - As metrics get more complicated and larger this could lead to potential performance degradation.

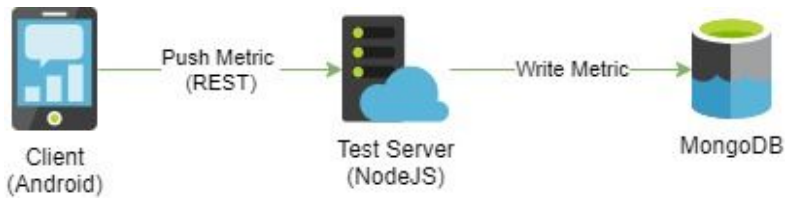






Test 2: Magma Data Lake Vs MongoDB WRITE Performance (Setup)

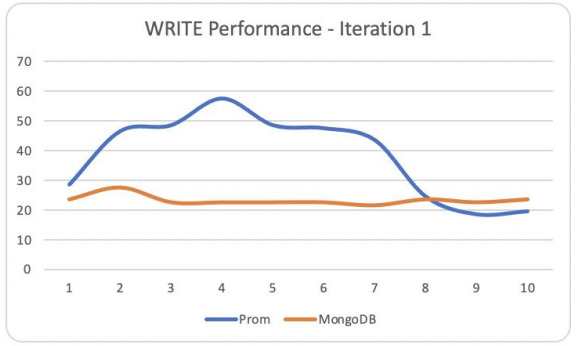
- Fig and Table show the setup and configuration used for this test.
- MongoDB and NodeJS ETL service are deployed on a micro AWS server and are not optimized for highest performance.
- The OpenSchema SDK was modified for the purpose of acting as a test client, pushing several iterations of single metric events.

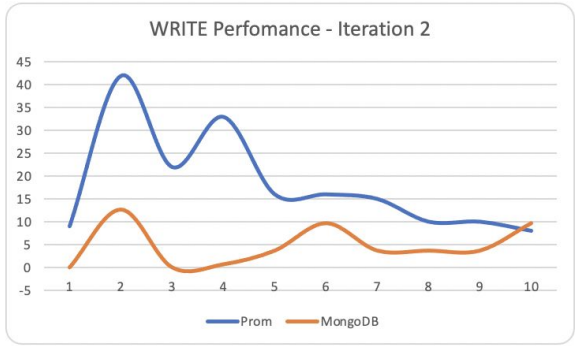


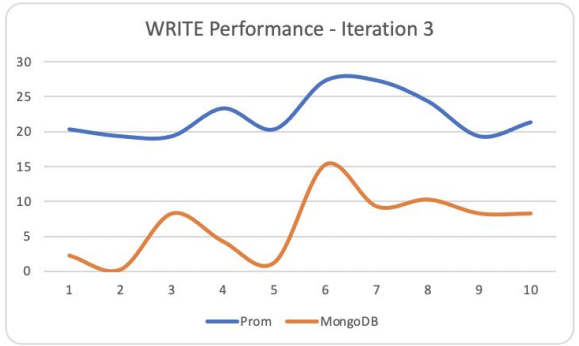
Prometheus	Prometheus within Magma Orc8r
Magma Controller	Magma Orc8r 3x t3.large (23.22.121.164, 54.174.41.72, 52.72.197.188)
Communication method	gRPC Interface
Client device	Google Pixel 3a XL (Android 9)
KPI Collection Tools	Android logcat
MongoDB	MongoDB Community Server AWS EC2 t2.micro (us-west-1, 13.52.214.86)
OpenSchema ETL	NodeJS Service AWS EC2 t2.micro (us-west-1, 13.52.214.86)
Communication method	REST Interface
Client device	Google Pixel 3a XL (Android 9)
KPI Collection Tools	Android logcat

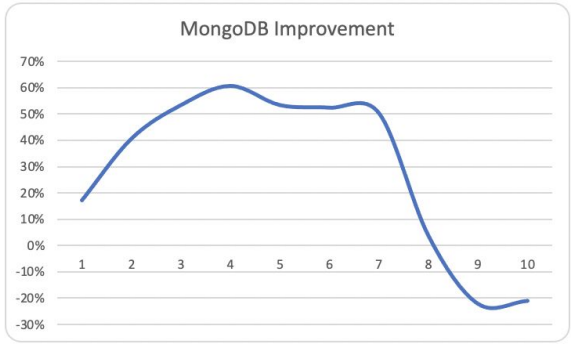
Test 2: Magma Data Lake vs MongoDB WRITE Latency(Results)

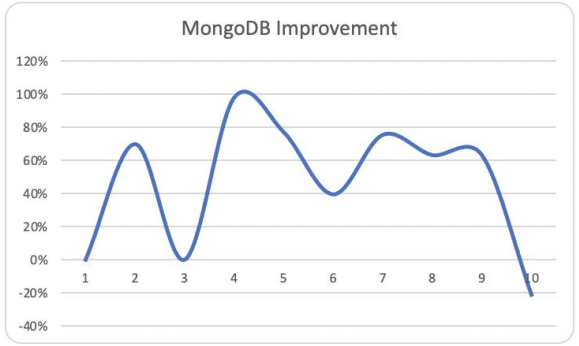
- Graphs below compares WRITE latency of single metrics events in Magma Data Lake and MongoDB
- Summary of Observations:
  - Average WRITE latency for Mongo is 14.2ms and for Magma Data Lake 29.23ms (Almost 2x slower with Magma Data Lake)
  - 85% of the times MongoDB is faster than Magma Data Lake
  - 62% of the time MongoDB is 50% or more faster than Magma Data Lake
  - Magma Data Lake results have almost 3 times more dispersion with 9.4 standard deviation version 3.4 for MongoDB
  - In this test we used a basic MongoDB + NodeJS on an AWS micro box. MongoDB latencies can be improved by tuning these parameters.

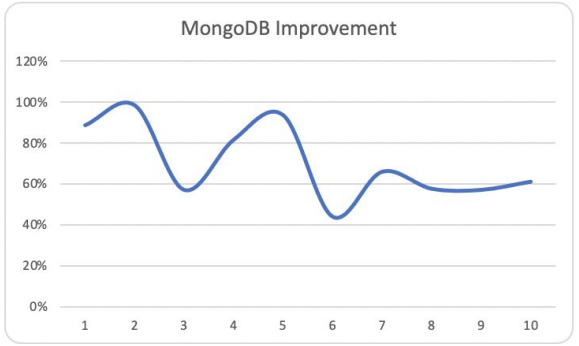












Prometheus Limitation for OpenSchema

- Reviewing Prometheus documentation, the tool is primary designed for different purposes than what OpenSchema needs.
  - Prometheus is a **monitoring** solution, not meant to act as a **data lake** which OpenSchema needs.
  - Prometheus uses pull mechanism which does not work for UEs and push gateway should be implemented. However using push gateway is only recommended in very limited scenarios and adds additional cost.
  - Prometheus doesn't allow for metrics with complex data structures (e.g. multiple columns with many different values).
    - According to Prometheus documentation: "*Do not use labels to store dimensions with high cardinality (many different label values), such as user IDs, email addresses, or other unbounded sets of values*".
  - On the Analytics side, PromQL is generally hard to comprehend and not easy to use specially for complex set of metrics.

Source:
https://prometheus.io/docs/practices/naming/#labels