Apache Polaris (incubating) New Persistence Performance Report

2025-03-19

Introduction

This report presents the performance comparison between two Apache Polaris (incubating) implementations:

- `EL-PG` - The original Apache Polaris implementation (`6a345913`)
- `MongoDB` - The implementation with the new persistence layer and MongoDB (#1189)

The benchmarks were executed using the Gatling-based benchmark suite described in the project's README, focusing on both sequential (see below) and concurrent workloads.

To facilitate a comprehensive performance comparison between the evaluated implementations, a benchmark with only sequential query was included. It is acknowledged that the Eclipselink+Postgresql implementation currently exhibits a limitation, as documented in Polaris issue #1123, which restricts its ability to reliably process concurrent queries. Therefore, the sequential benchmark provides a baseline for assessing its performance characteristics in isolation, allowing for a meaningful comparison against new persistence implementation.

Benchmark Methodology

The benchmarking process begins by creating a dataset against an empty Polaris instance. This initial step ensures that the system is tested from a clean state. Following the dataset creation, two mixed workloads are executed. The first workload consists of a mixed read/write workload with a 50/50 ratio, providing a balanced test of both read and write operations. The second mixed workload has a 99/01 read/write ratio, which predominantly tests the read performance under minimal write conditions.

Environment

The benchmarks were run against a single-server deployment of Polaris on an EC2 `m5d.2xlarge` instance (8 vCPU, 32 GiB RAM, 300 GB NVMe SSD).

The server was run as a standalone JVM process. The associated database (Postgresql/MongoDB) was run in a Docker container. The database storage location

was configured to point to the local NVMe SSD. The Polaris server was also deployed on that local SSD.

For benchmarks that required Postgresql, the database was configured with the same `postgresql.conf` file that is provided in the `getting-started/eclipselink/` folder of the Polaris repository. I.e. Postgresql was configured with serializable isolation.

MongoDB was run with default settings.

Test Dataset Structure

Namespaces are distributed in a binary-tree shape, as described in the project's README.

Sequential Benchmark Parameters

The test dataset for benchmarks with sequential queries has the following number of entities:

- 500 catalogs
- 8191 namespaces
- 8192 tables
- 8192 views

The sequential benchmarks used the following configuration parameters:

```
export GC_OPTS='-XX:+UseG1GC -Xms8G -Xmx8G -XX:+AlwaysPreTouch
-Xlog:gc*=debug,safepoint*:file=/tmp/gc.log:uptime,tags,level:filecount=1,fil
esize=200M'
export POLARIS_OPTS='-Dquarkus.otel.sdk.disabled=true
-Dpolaris.bootstrap.credentials=POLARIS,root,s3cr3t
-Dquarkus.log.console.level=ERROR'

export CLIENT_ID=root
export CLIENT_SECRET=s3cr3t
export NUM_CATALOGS=500
export DEFAULT_BASE_LOCATION=file:///tmp/polaris2
export NAMESPACE_WIDTH=2
export NAMESPACE_DEPTH=13
export NUM_NAMESPACE_PROPERTIES=10
export NUM_NAMESPACE_PROPERTY_UPDATES=1
export NUM_TABLES_PER_NAMESPACE=2
export NUM_COLUMNS=10
export NUM_TABLE_PROPERTIES=10
export NUM_TABLE_PROPERTY_UPDATES=1
export NUM_VIEWS_PER_NAMESPACE=2
export NUM_VIEW_PROPERTY_UPDATES=1
```

Concurrent Benchmark Parameters

The test dataset for benchmarks with concurrent queries has the following number of entities:

- 500 catalogs
- 65535 namespaces
- 65536 tables
- 65536 views

The concurrent benchmarks used the following configuration parameters:

```
export GC_OPTS='-XX:+UseG1GC -Xms8G -Xmx8G -XX:+AlwaysPreTouch
-Xlog:gc*=debug,safepoint*:file=/tmp/gc.log:uptime,tags,level:filecount=1,fil
esize=200M'
export POLARIS_OPTS='-Dquarkus.otel.sdk.disabled=true
-Dpolaris.bootstrap.credentials=POLARIS,root,s3cr3t
-Dquarkus.log.console.level=ERROR'

export CLIENT_ID=root
export CLIENT_SECRET=s3cr3t
export NUM_CATALOGS=500
export DEFAULT_BASE_LOCATION=file:///tmp/polaris2
export NAMESPACE_WIDTH=2
export NAMESPACE_DEPTH=16
export NUM_NAMESPACE_PROPERTIES=10
export NUM_NAMESPACE_PROPERTY_UPDATES=1
export NUM_TABLES_PER_NAMESPACE=2
export NUM_COLUMNS=10
export NUM_TABLE_PROPERTIES=10
export NUM_TABLE_PROPERTY_UPDATES=1
export NUM_VIEWS_PER_NAMESPACE=2
export NUM_VIEW_PROPERTY_UPDATES=1
```

The key difference between sequential and concurrent benchmarks was the namespace depth: 13 for sequential vs 16 for concurrent, resulting in a larger dataset for concurrent tests.

Benchmark Types

The following benchmark types were executed:

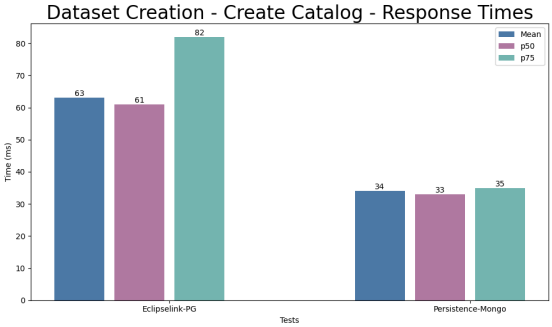
- Dataset Creation Workload: 100% writes
- **50/50 Mixed Workload**: 50% reads, 50% writes (`READ_WRITE_RATIO=0.5`)
- **99/01 Mixed Workload**: 99% reads, 1% writes (`READ_WRITE_RATIO=0.99`)

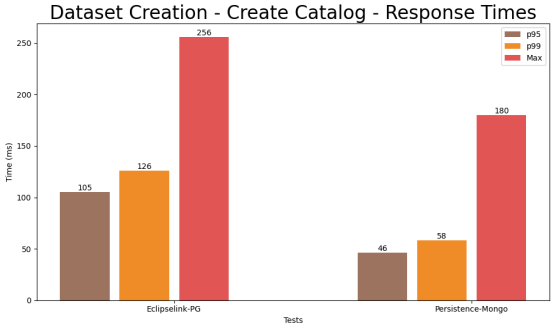
Each benchmark type was executed in both sequential and concurrent modes. In sequential mode, only one HTTP query is executed at a time. In concurrent mode, up to 50 simultaneous HTTP queries are executed.

Sequential Benchmark Results

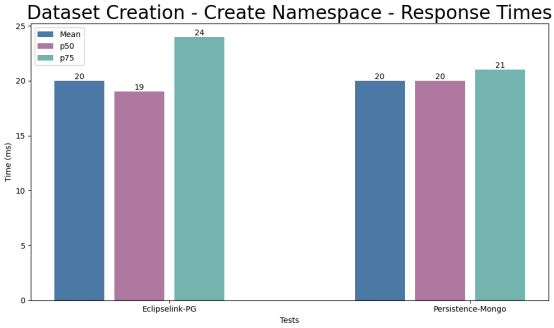
Dataset creation

Catalog Creation Performance





Namespace Creation Performance



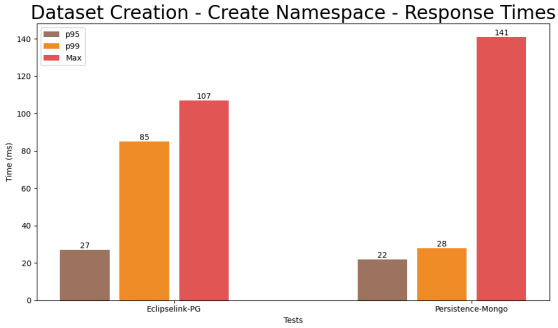
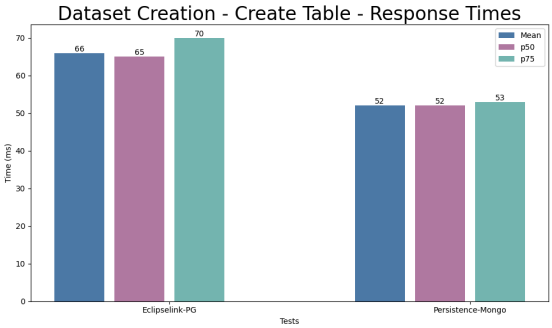
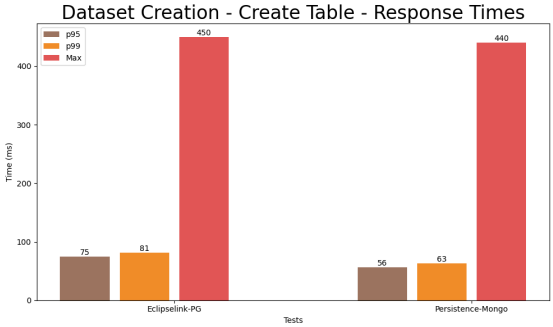
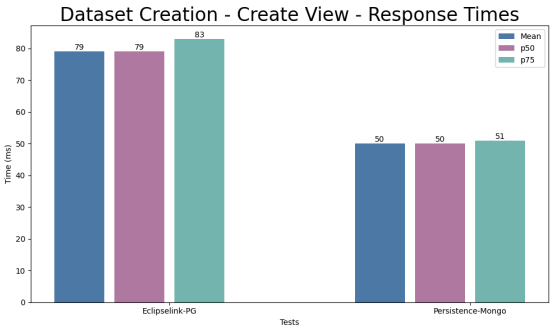


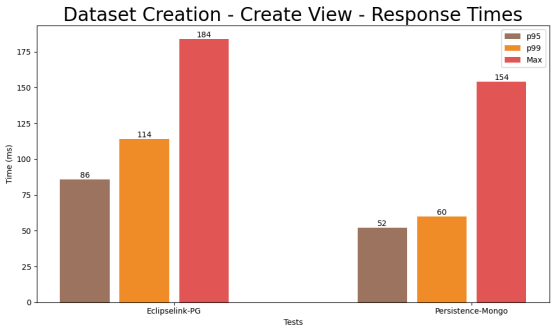
Table Creation Performance





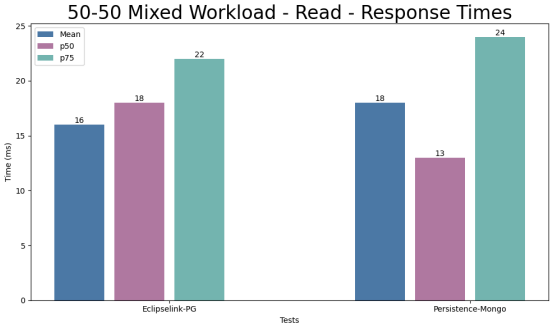
View Creation Performance

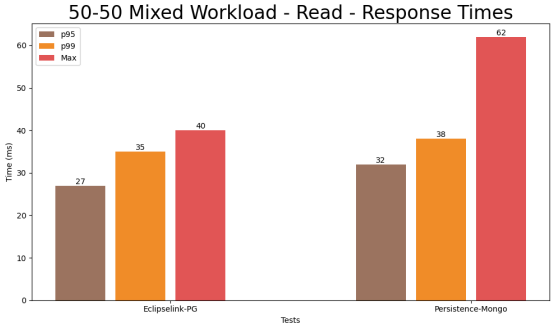




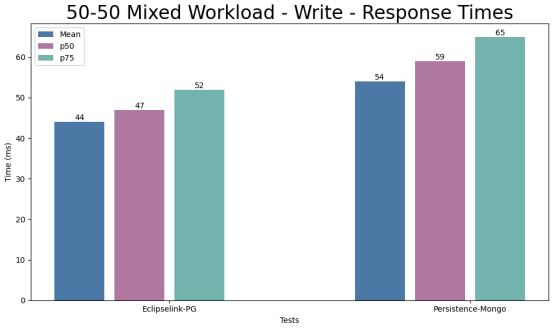
50/50 mixed workload

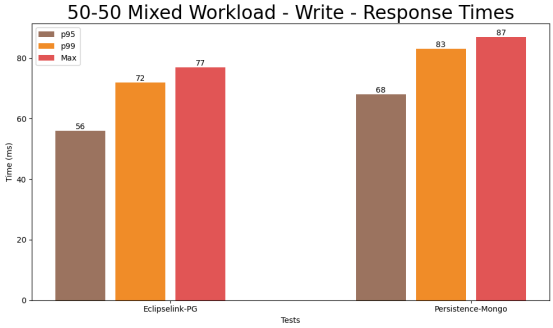
Read Performance





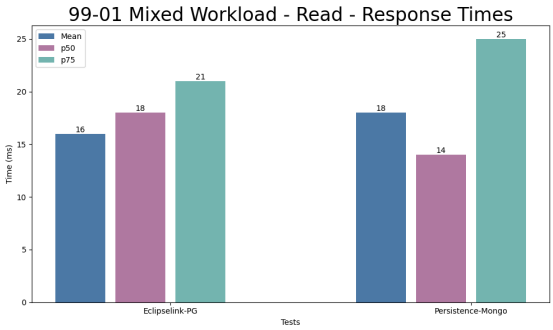
Write Performance

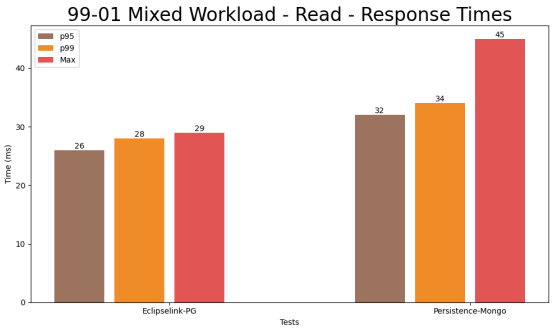




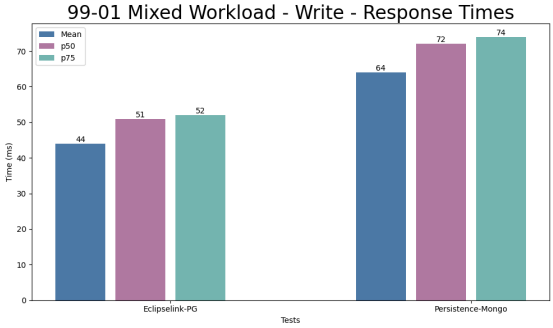
99/01 mixed workload

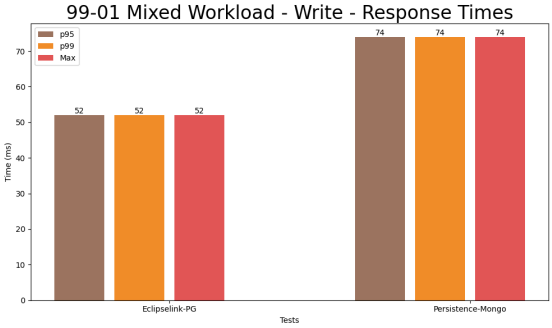
Read Performance





Write Performance

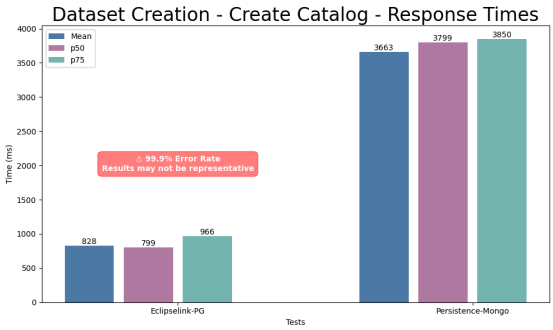


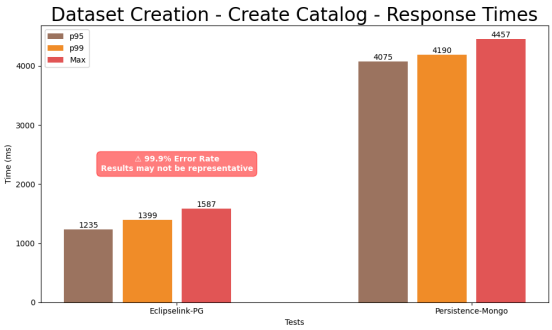


Concurrent Benchmark Results

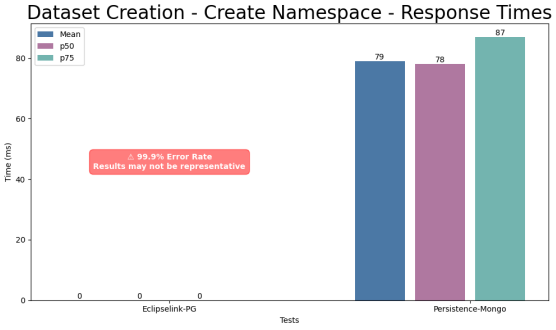
Dataset creation

Catalog Creation Performance





Namespace Creation Performance



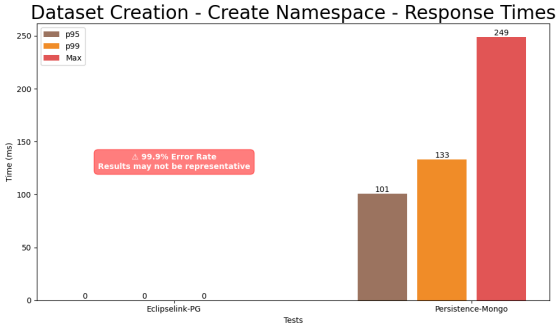
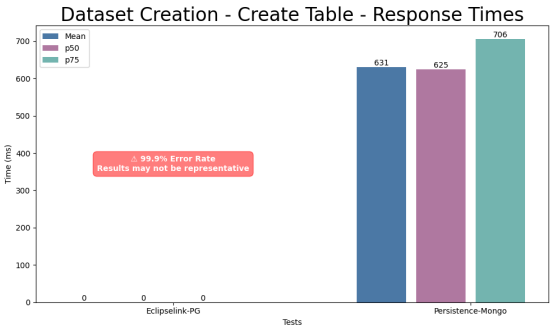
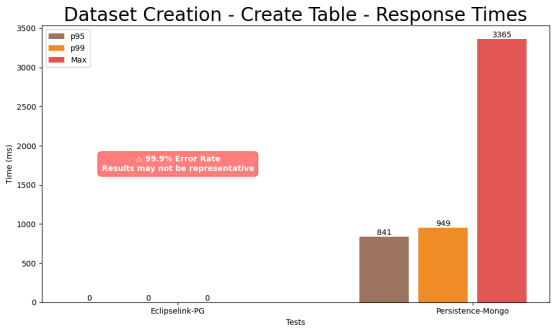
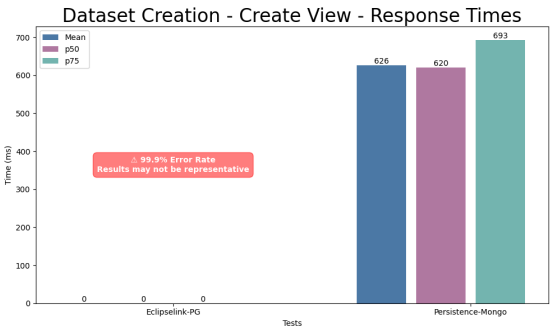


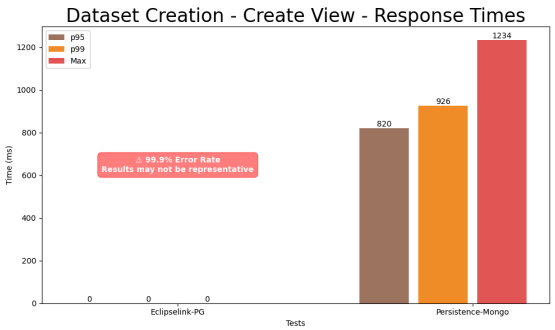
Table Creation Performance





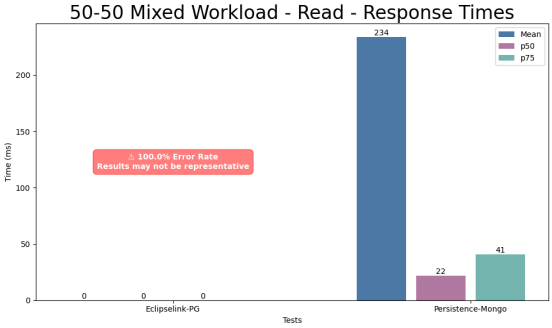
View Creation Performance

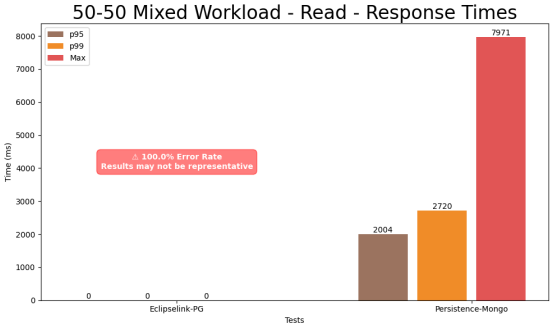




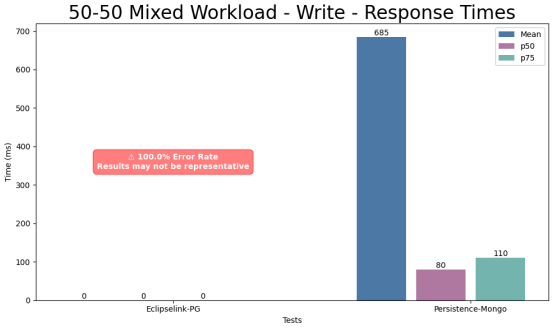
50/50 mixed workload

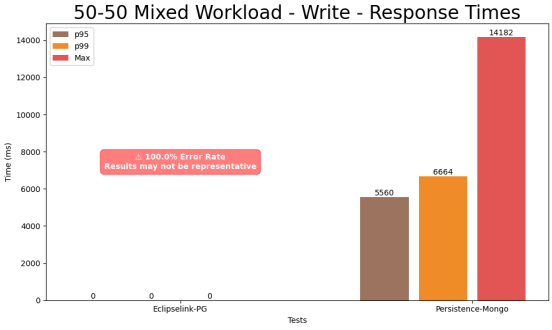
Read Performance





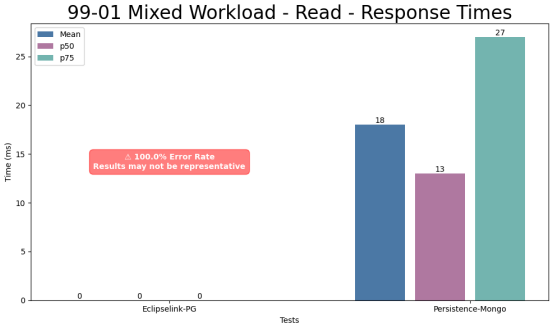
Write Performance

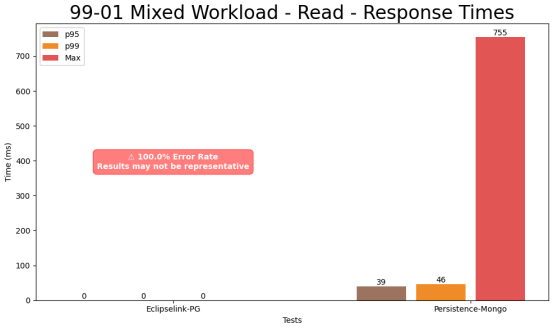




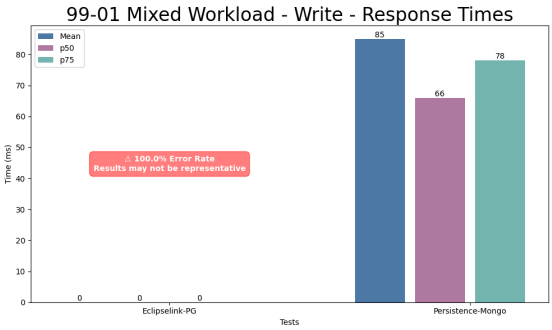
99/01 mixed workload

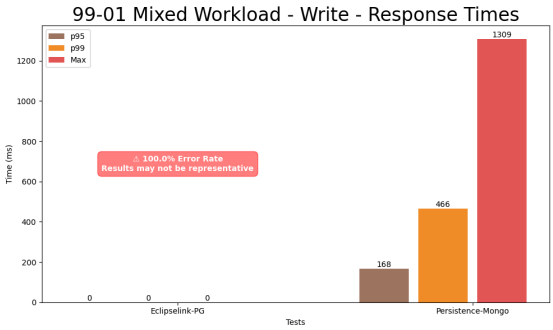
Read Performance





Write Performance





Analysis

Sequential Workload Performance

In sequential workload tests, both implementations demonstrated stable performance with 100% success rates. The new MongoDB-based implementation showed:

- **Dataset Creation**:
  - Catalog creation was approximately 46% faster
  - Namespace creation performance was equivalent
  - Table creation was approximately 21% faster
  - View creation was approximately 37% faster
- **Read/Write Operations**:
  - For the 50/50 workload, read operations were approximately 13% slower, but write operations were about 23% faster
  - For the 99/01 workload, read operations were approximately 13% slower, but write operations were about 46% faster
  - Note that those results may not be fully reproducible given the low total number of requests (<300)

Concurrent Workload Performance

The concurrent workload tests revealed significant differences in scalability and stability:

- **Dataset Creation**:

    - The Eclipselink implementation had significant failures with concurrent catalog creation (28% success rate) and complete failure for namespace/table/view creation
    - The MongoDB implementation maintained 100% success rate for all operations
- **Read/Write Operations**:
    - For the 50/50 workload, the Eclipselink implementation failed entirely for both read and write operations
    - For the 99/01 workload, the Eclipselink implementation failed entirely
    - The MongoDB implementation completed all read and write operations successfully

The response times cannot be compared between the two versions given the total failure rate of the Eclipselink implementation.

The Eclipselink failure rate can be explained by a combination of top-level namespace creation failures (HTTP 500) leading to ancestors missing for the rest of the entities (namespaces, tables, views). See issue #1123. This high failure rate was reproduced at every run.

Conclusion

The benchmarks demonstrate that the new Persistence implementation offers:

1. Comparable or better performance for sequential operations
2. Significantly better reliability under concurrent load
3. Consistent read and write performance even under high-concurrency scenarios

These results suggest that the new implementation provides a robust foundation for scaling Polaris, particularly for workloads dominated by high concurrency.

Appendix: Raw Results

The complete Gatling reports, with query and error breakdown, are available here: https://drive.google.com/file/d/142XQF3kSe5TezXZbwZ45REpX2LzRz_7c/view?usp=sharing

Sequential Benchmark Results

```
Test case,Implementation,Query,Number of requests,Number of successful requests,Number of failed requests,avg response time,min response time,p50
```

```
response time,max response time
Dataset creation,Eclipselink-PG,Create Catalog,500,500,0,63,31,61,256
Dataset creation,Eclipselink-PG,Create Namespace,8191,8191,0,20,11,19,107
Dataset creation,Eclipselink-PG,Create Table,8192,8192,0,66,54,65,450
Dataset creation,Eclipselink-PG,Create View,8192,8192,0,79,68,79,184
Dataset creation,Persistence-Mongo,Create Catalog,500,500,0,34,26,33,180
Dataset creation,Persistence-Mongo,Create Namespace,8191,8191,0,20,17,20,141
Dataset creation,Persistence-Mongo,Create Table,8192,8192,0,52,46,52,440
Dataset creation,Persistence-Mongo,Create View,8192,8192,0,50,45,50,154
50/50 RW workload,Eclipselink-PG,Read,154,154,0,16,5,18,40
50/50 RW workload,Eclipselink-PG,Write,146,146,0,44,18,47,77
50/50 RW workload,Persistence-Mongo,Write,147,147,0,54,20,59,87
50/50 RW workload,Persistence-Mongo,Read,153,153,0,18,8,13,62
99/01 RW workload,Eclipselink-PG,Read,295,295,0,16,4,18,29
99/01 RW workload,Eclipselink-PG,Write,5,5,0,44,18,51,52
99/01 RW workload,Persistence-Mongo,Read,296,296,0,18,8,14,45
99/01 RW workload,Persistence-Mongo,Write,4,4,0,64,42,72,74
```

Concurrent Benchmark Results

```
Test case,Implementation,Query,Number of requests,Number of successful
requests,Number of failed requests,avg response time,min response time,p50
response time,max response time
Dataset creation,Eclipselink-PG,Create Catalog,500,140,360,828,325,799,1587
Dataset creation,Eclipselink-PG,Create Namespace,65535,0,65535,0,0,0,0
Dataset creation,Eclipselink-PG,Create Table,65536,0,65536,0,0,0,0
Dataset creation,Eclipselink-PG,Create View,65536,0,65536,0,0,0,0
Dataset creation,Persistence-Mongo,Create
Catalog,500,500,0,3663,326,3799,4457
Dataset creation,Persistence-Mongo,Create
Namespace,65535,65535,0,79,16,78,249
Dataset creation,Persistence-Mongo,Create
Table,65536,65536,0,631,106,625,3365
Dataset creation,Persistence-Mongo,Create View,65536,65536,0,626,97,620,1234
50/50 RW workload,Eclipselink-PG,Read,14903,0,14903,0,0,0,0
50/50 RW workload,Eclipselink-PG,Write,15158,0,15158,0,0,0,0
50/50 RW workload,Persistence-Mongo,Read,15036,15036,0,234,7,22,7971
50/50 RW workload,Persistence-Mongo,Write,15067,15067,0,685,16,80,14182
99/01 RW workload,Eclipselink-PG,Read,29954,0,29954,0,0,0,0
99/01 RW workload,Eclipselink-PG,Write,295,0,295,0,0,0,0
99/01 RW workload,Persistence-Mongo,Read,29585,29585,0,18,6,13,755
99/01 RW workload,Persistence-Mongo,Write,303,303,0,85,33,66,1309
```

Appendix: Benchmark code

The Gatling benchmarks are available at https://github.com/pingtimeout/polaris/tree/39582664789237c669e953a23980ccd05570a501/benchmarks