**Collecting Metrics**

**Key Metrics to Capture**

**General Metrics**

* CPU usage
* Memory consumption
* Disk I/O
* Network throughput
* System load

**MongoDB-Specific Metrics**

* Connections (current, available, total created)
* Opcounters (insert, query, update, delete, getmore, command)
* Lock percentage
* Queue sizes (read, write)
* Page faults
* Btree miss ratio
* Replication lag
* Oplog window size
* Index usage
* Query performance
* Storage engine metrics (e.g., WiredTiger cache usage)
* Journaling metrics
* Background flush metrics
* Cursor metrics

**Sharding-Specific Metrics**

* Chunk sizes
* Balancer activity
* Shard key performance
* Query response times per shard
* Migration status
* Orphan documents
* Chunk migrations
* Balancer rounds

**Spark-Specific Metrics**

* Execution time
* Task completion rate
* Data transfer metrics
* Garbage collection time
* Memory usage
* Disk I/O
* Shuffle read/write
* Executor metrics (CPU, memory, disk I/O)
* Stage metrics (input size, output size, shuffle read/write)
* Job metrics (number of stages, tasks, failed tasks)
* Scheduler delay
* Task serialization time
* Result serialization time
* JVM metrics (heap usage, non-heap usage, GC time)

**Read/Write Operation Metrics**

* Throughput
* Latency
* Error rates during live application interactions and reporting tool queries
* Index usage
* Query performance
* Aggregation pipeline performance
* MapReduce performance

**Step-by-Step Guide to Collect Metrics**

1. **General Metrics**:
   * Use **mongostat** to monitor CPU, memory, and disk I/O.
   * Example command:

mongostat --host <mongos\_host> --port <mongos\_port>

1. **MongoDB-Specific Metrics**:
   * Use **mongostat** and **mongotop** to track connections, opcounters, lock percentage, etc.
   * Example commands:
   * mongostat --host <mongos\_host> --port <mongos\_port>

mongotop --host <mongos\_host> --port <mongos\_port>

* + Use **db.serverStatus()** to get detailed metrics:

db.serverStatus()

1. **Sharding-Specific Metrics**:
   * Use Ops Manager/Cloud Manager dashboards to track chunk sizes, balancer activity, and shard key performance.
   * Example command for balancer status:

sh.status()

* + Use **db.printShardingStatus()** for detailed sharding status:

db.printShardingStatus()

1. **Spark-Specific Metrics**:
   * Configure Spark to log metrics to Prometheus.
   * Integrate Prometheus with Grafana for visualization.
   * Example Spark configuration:
   * spark.metrics.conf.\*.sink.graphite.class=org.apache.spark.metrics.sink.GraphiteSink
   * spark.metrics.conf.\*.sink.graphite.host=<graphite\_host>

spark.metrics.conf.\*.sink.graphite.port=<graphite\_port>

* + Use Spark UI for real-time monitoring:

http://<spark\_master\_host>:4040

1. **Read/Write Operation Metrics**:
   * Use **mongotop** to track read/write operations.
   * Example command:

mongotop --host <mongos\_host> --port <mongos\_port>

* + Use **explain()** to analyze query performance:

db.collection.explain().find({...})

**Analyzing Metrics**

**Interpretation Guidelines**

1. **CPU, Memory, and I/O Metrics**:
   * High CPU usage may indicate inefficient queries or indexing issues.
   * High memory consumption may suggest the need for more RAM or optimization of data structures.
   * High disk I/O may point to slow storage or the need for better indexing.
   * High network throughput may indicate network bottlenecks.
   * High system load may suggest resource contention.
2. **MongoDB-Specific Metrics**:
   * High connection counts may indicate connection leaks or inefficient connection management.
   * High opcounters may suggest heavy load or inefficient queries.
   * High lock percentage may indicate contention issues.
   * High queue sizes may suggest the system is overwhelmed.
   * High page faults may indicate insufficient memory.
   * High Btree miss ratio may suggest inefficient indexing.
   * High replication lag may indicate issues with the replica set.
   * Small oplog window size may suggest insufficient disk space.
   * Low index usage may indicate the need for better indexing strategies.
   * Poor query performance may suggest the need for query optimization.
   * High WiredTiger cache usage may indicate the need for more memory.
   * High journaling metrics may suggest frequent write operations.
   * High background flush metrics may indicate heavy write load.
   * High cursor metrics may suggest inefficient query patterns.
3. **Sharding-Specific Metrics**:
   * Large chunk sizes may indicate the need for better shard key selection.
   * Frequent balancer activity may suggest uneven data distribution.
   * High query response times may point to inefficient shard key or indexing issues.
   * Migration status and orphan documents may indicate issues with data migration.
   * High chunk migrations may suggest frequent data movement.
   * High balancer rounds may indicate continuous balancing activity.
4. **Spark Job Performance**:
   * Long execution times may indicate inefficient data processing.
   * Low task completion rates may suggest resource contention.
   * High data transfer metrics may point to network bottlenecks.
   * High garbage collection time may indicate memory management issues.
   * High memory usage may suggest the need for more RAM.
   * High disk I/O may point to slow storage.
   * High shuffle read/write may indicate inefficient data shuffling.
   * High executor metrics may suggest resource contention.
   * High stage metrics may indicate inefficient data processing.
   * High job metrics may suggest job failures or retries.
   * High scheduler delay may indicate resource contention.
   * High task serialization time may suggest inefficient data serialization.
   * High result serialization time may suggest inefficient result serialization.
   * High JVM metrics may indicate memory management issues.

**Benchmarking Standards**

**Defining Benchmarks**

1. **Typical Tenant Workloads**:
   * Establish benchmarks based on average tenant workloads and data volume.
   * Example benchmarks:
     + Read throughput: 10,000 ops/sec
     + Write throughput: 5,000 ops/sec
     + Query latency: < 50 ms
     + Replication lag: < 10 seconds
     + Oplog window size: > 24 hours
     + Index usage: > 90%
     + Query performance: < 100 ms
     + Aggregation pipeline performance: < 500 ms
     + MapReduce performance: < 1000 ms
2. **Benchmark Tables and Charts**:
   * Create tables and charts to visualize benchmark expectations for different operations.
   * Example table:

| **Operation** | **Benchmark Value** |
| --- | --- |
| Read Ops | 10,000 ops/sec |
| Write Ops | 5,000 ops/sec |
| Query Latency | < 50 ms |
| Replication Lag | < 10 seconds |
| Oplog Window Size | > 24 hours |
| Index Usage | > 90% |
| Query Performance | < 100 ms |
| Aggregation Pipeline | < 500 ms |
| MapReduce Performance | < 1000 ms |

# Performance Measurement Template for MongoDB and Spark Integration

## 1. Introduction

Purpose of Document: Provide guidelines and procedures for measuring and analyzing the performance of MongoDB with Spark integration.  
System Overview: Description of the MongoDB cluster and Spark setup, including hardware and software specifications.

## 2. Test Environment Setup

Hardware Specifications: List of all hardware used, including servers, network configurations, and storage details.  
Software Versions: Versions of MongoDB, Spark, and any other relevant software.  
Configuration Details: Configuration settings for MongoDB and Spark.

## 3. Metrics to be Collected

### A. Spark Metrics

Task and Stage Metrics  
 - Task Completion Rates  
 - Stage Failures  
 - Executor Deserialization Time  
Memory Management  
 - Memory Used vs. Total Available  
 - Garbage Collection Time  
Data Shuffling Metrics  
 - Shuffle Read/Write Bytes  
 - Shuffle Fetch Wait Time  
Job Scheduling and Execution  
 - Scheduler Delay  
 - Result Serialization Time

### B. MongoDB Metrics

General Cluster Health Metrics  
 - Server Status Metrics  
 - Replica Set Metrics  
 - Resource Utilization  
Sharding-Specific Metrics  
 - Chunk Distribution and Size  
 - Balancer Activity  
 - Shard Key Efficiency  
 - Jumbo Chunks  
Database Operations Metrics  
 - Operation Throughput  
 - Operation Latency  
 - Query Executor Stats  
 - Slow Queries  
Storage Engine Metrics  
 - WiredTiger Metrics  
 - Checkpoint Metrics  
 - I/O Wait  
Journaling Metrics  
 - Journal Write Latency  
 - Journal Compression Ratios  
Replication Metrics  
 - Replication Lag  
 - Replication Throughput  
 - Oplog Window  
Network Metrics  
 - Network Traffic  
 - Network Errors and Retransmits  
Concurrency Metrics  
 - Locks and Latches  
 - Queue Lengths

## 4. Data Collection Procedure

Collection Tools: Specify tools and methods for data collection, e.g., MongoDB Ops Manager, Spark Web UI, command-line utilities.  
Frequency and Duration: Define how often and for how long data should be collected during testing.  
Data Storage: Describe where and how the collected data will be stored and managed.

**Step-by-Step Guide to Collect Metrics**

1. **General Metrics**:
   * Use **mongostat** to monitor CPU, memory, and disk I/O.
   * Example command:

mongostat --host <mongos\_host> --port <mongos\_port>

1. **Sharding-Specific Metrics**:
   * Use Ops Manager/Cloud Manager dashboards to track chunk sizes, balancer activity, and shard key performance.
   * Example command for balancer status:

sh.status()

1. **Spark-Specific Metrics**:
   * Configure Spark to log metrics to Prometheus.
   * Integrate Prometheus with Grafana for visualization.
2. **Read/Write Operation Metrics**:
   * Use **mongotop** to track read/write operations.
   * Example command:

mongotop --host <mongos\_host> --port <mongos\_port>

**Analyzing Metrics**

**Interpretation Guidelines**

1. **CPU, Memory, and I/O Metrics**:
   * High CPU usage may indicate inefficient queries or indexing issues.
   * High memory consumption may suggest the need for more RAM or optimization of data structures.
   * High disk I/O may point to slow storage or the need for better indexing.
2. **Sharding-Specific Metrics**:
   * Large chunk sizes may indicate the need for better shard key selection.
   * Frequent balancer activity may suggest uneven data distribution.
   * High query response times may point to inefficient shard key or indexing issues.
3. **Spark Job Performance**:
   * Long execution times may indicate inefficient data processing.
   * Low task completion rates may suggest resource contention.
   * High data transfer metrics may point to network bottlenecks.