MongoDB and Redis

Group - 1:

- Jigyashu Saravta (saravta2)
- Shivam Rawat (ssrawat2)
- Sundar Raghavan (sundarr2)

Presentation Flow

- Project Overview
- Background
- Methodology
  - Data Preparation
  - Database Setup
  - Query Testing
  - Backup and Restore
  - Monitoring and Benchmarking
- Results
  - Performance
  - Scalability
  - Cost Analysis
- Discussion

Project Overview

- Objective:
  - Evaluating MongoDB and Redis for Geospatial Data Processing.
- Why this project?:
  - To compare the strengths and trade-offs of document-based vs. in-memory database architectures in managing complex data and workloads.
- Tech Stack:
  - MS Azure VMs, Redis Cloud, MongoDB, Compass, Redis, Redis Insight.

Background

- Geospatial Data Overview:
  - Geospatial data represents geographical features, enabling analysis of spatial patterns, transportation networks, and urban development.
- OpenStreetMap (OSM):
  - OpenStreetMap (OSM) is a crowdsourced map dataset containing geospatial information like roads, buildings, and points of interest.
- Technologies Overview:
  - MongoDB: A document-based database with native geospatial querying and indexing.
  - Redis: An in-memory key-value store, offering fast access to geospatial data.
- Use Case for Comparison:
  - Comparing these technologies helps understand the suitability of different database architectures for large-scale geospatial data.

Methodology

- Data Preparation:
  - Preprocessing OSM data for MongoDB and Redis.
- Database Setup:
  - Configurations, security, and environment setup.
- Query Testing:
  - Executing traditional and geospatial queries and analyzing performance.
- Backup and Restore:
  - Strategies for data reliability and recovery.
- Monitoring and Benchmarking:
  - Collecting performance metrics for comparison.

Database Setup

Category	MongoDB	Redis
Installation	Installed using MongoDB's official repo on Azure VM.	Installed using Redis repo on Azure VM and Redis Cloud.
Configuration	• Enabled authentication (username/password). • Added necessary geospatial indexing. • Install mongodump, mongorestore, and osmium-tool	• Half of the work was done on Azure B2 instances and half on Redis Cloud. • Configured RedisJSON module for geospatial data.
Security	Applied IP whitelisting for access control.	Applied IP whitelisting for access control.
Challenges	None	Installing Redis modules.

Data Preparation

Overview:

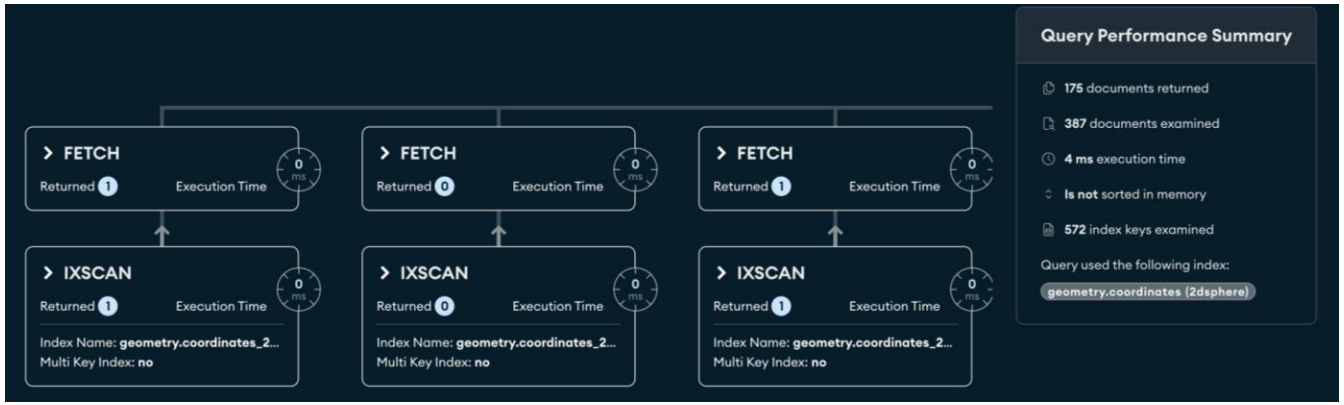
Preparing OpenStreetMap (OSM) geospatial data for compatibility with MongoDB and Redis.

Category	MongoDB	Redis
Data Format	Osm.pbf -> Geojson -> json	Osm.pbf -> Geojson -> json -> txt(redisJson commands)
Tools Used	Osmium-tool, python scripts.	Osmium-tool, python scripts
Steps	Filtered points of interest, 2d geospatial index.	Indexing for each JSON key of interest, creating a sorted set of coordinates to mimic a geo index
Challenges	Handling large OSM files during conversion.	• 0.038% of records failed to upload • Generating a txt file with individual SET commands for each record is a negative for data integrity

Query Testing

Category	MongoDB	Redis
Types of Queries	• Geospatial: finding nearby points of interest. • Match query	• Geospatial: finding nearby points of interest. • Match query
Query Execution	• Used MongoDB Compass for testing queries. • Wrote aggregation pipelines to combine geospatial and non-spatial data.	• Used RedisInsight's workbench CLI tool for testing geospatial queries.
Data Size	1.8 Million docs	1.8 Million docs
Performance Metrics	• Query execution time. • CPU and memory usage during execution.	• Query execution time. • CPU and memory usage during execution.
Challenges	Complex Javascript syntax as queries grew larger	• Limited community support meant everything had to be learnt from Redis docs • Lua scripts which could've been convenient are hard to use and integrate

Query Testing (Geospatial)




```
GEORADIUSBYMEMBER geo_data 674cb88775da06487a82b0d1 1 km

1) "674cb8e275da06487a89c87b"
2) "674cb8dc75da06487a895244"
3) "674cb8dc75da06487a8952ae"
4) "674cb8dc75da06487a89535d"
5) "674cb89075da06487a836438"
6) "674cb88775da06487a82b0d1"
7) "674cb92e75da06487a8fb4e6"
8) "674cb88775da06487a82b0d2"
9) "674cb88775da06487a82b0cf"
10) "674cb88775da06487a82b0d0"
11) "674cb8e275da06487a89c87a"
12) "674cb8e275da06487a89c875"

19:16:58 8 Dec 2024        43.118 msec
```

Query Testing (Match)

FT.SEARCH master_index "@highway:(crossing)" 19:00:14 8 Dec 2024 90.561 msec

Matched: 147597

Doc	$
item:1281139	{\"_id\":{\"$oid\":\"674cb98675da06487a963d38\"},\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":[-97.
item:1293765	{\"_id\":{\"$oid\":\"674cb98875da06487a966e8a\"},\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":[-97.1
item:1410819	{\"_id\":{\"$oid\":\"674cb9a075da06487a9837c8\"},\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":[-97.0
item:1442879	{\"_id\":{\"$oid\":\"674cb9a775da06487a98b504\"},\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":[-96.

Backup and Restore

Category	MongoDB	Redis
Backup Methods	• Used a shell script containing the mongodump command to create backups on the VM. • Fetched backup files to the local machine.	• Used a shell script containing the redis-cli --rdb command to create .rdb backups on the VM. • Fetched .rdb files to the local machine.
Restore Methods	• Restored data locally using the mongorestore command. • Validated data accuracy post-restore.	• Restored .rdb files using redis-cli commands. • Verified restored geospatial data for integrity.
Automation	Scheduled the shell script as a recurring cron job for automated backups.	Automated .rdb backups by scheduling the shell script with a cron job.
Challenges		Redis Cloud instances doesn't provide access to .rdb (dump) files.

Benchmarking

Category	MongoDB	Redis
Monitoring Tools	Used Azure Monitor for tracking CPU, memory, and I/O usage.	Tracked CPU, memory, and I/O usage with Azure Monitor.
Benchmarked Metrics	Scaling Factor 1000, 3000, 5000, 10000	
Findings	MongoDB's TPS decreased significantly as the dataset scaled up, from 3367 at a scaling factor of 1000 to 818.93 at 10,000.	Redis maintained consistently high TPS across all scaling factors, exceeding 80,000 even at the largest scale.
Challenges		None

Results

Performance

- Mongo DB:
  - Match Query: Took 1081ms using a 2D index, significantly slower than Redis.
  - Geospatial Query: Outperformed Redis at 4ms due to optimized 2D indexing in Compass.
- Redis:
  - Match Query: Achieved 90.561ms, ~12x faster than MongoDB, leveraging in-memory sorted sets.
  - Geospatial Query: Took 43.118ms using geoadd sorted sets, with slight overhead for query execution.

Results

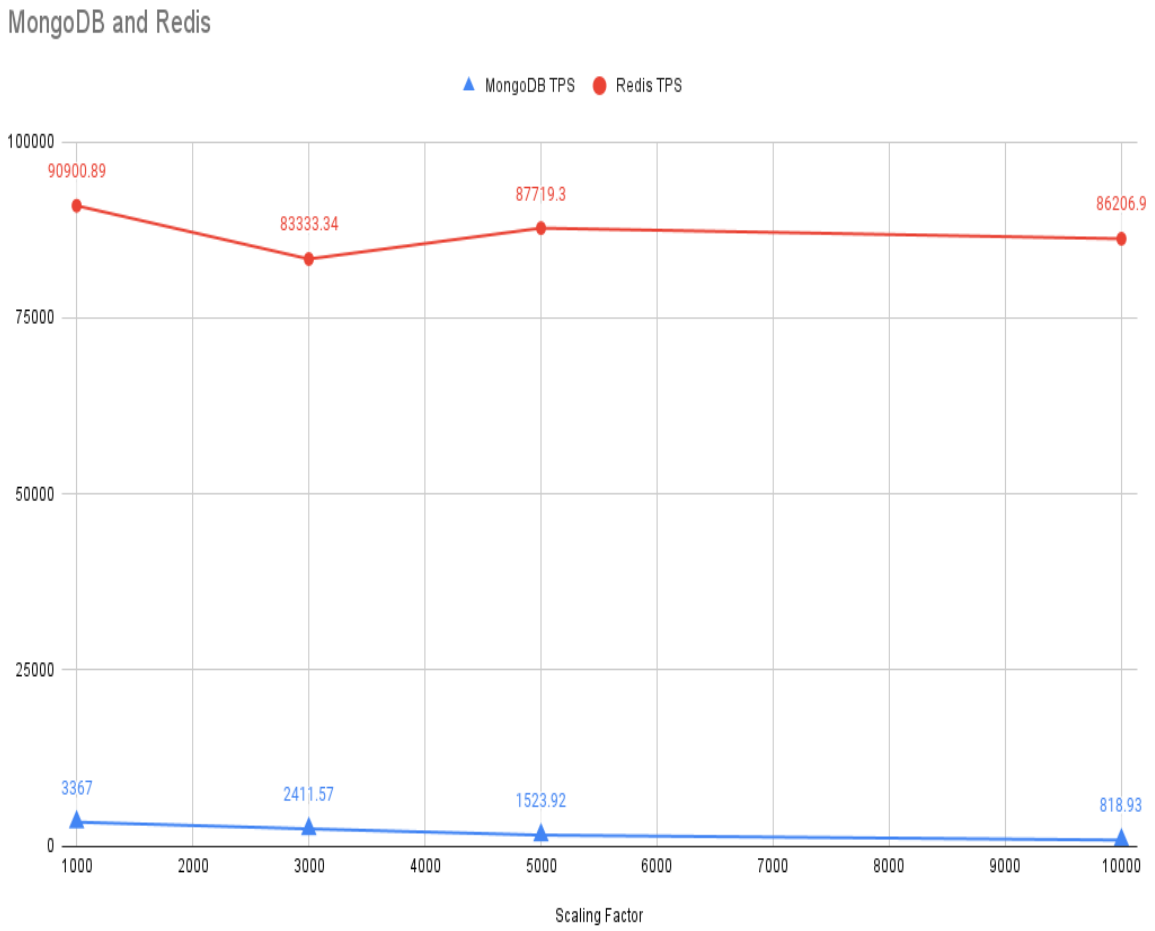
Scalability

- MongoDB:
  - Scalable and cost-effective as it uses disk space, but loading data took significantly longer compared to Redis.
  - Handles large datasets well, but slower to query as dataset size increases.
- Redis:
  - Highly scalable for smaller datasets with quick data uploads due to in-memory architecture.
  - Becomes expensive as dataset size grows, requiring more RAM to store data.

Results - Cost Analysis (Benchmarking)

Scaling Factor	MongoDB TPS	Redis TPS
1000	3367	90900.89
3000	2411.57	83333.34
5000	1523.92	87719.3
10000	818.93	86206.9

Standard B2s, vCPUs 2, RAM 4 GiB cost/hr	$ 0.042/hr			
Database	TPS	1 Day cost ($)	Transaction/ day in millions	Relative cost/performance (cost per million transactions in $)
MongoDB TPS (max for Scaling Factor 1000)	3367	1.008	290.9	0.0034
Redis TPS (max for Scaling Factor 1000)	90909.89	1.008	7854.61	0.00012



Aspect	MongoDB	Redis
Strengths	- Excellent for complex queries, especially aggregations. - Cost-effective for larger datasets due to disk-based storage. - Geospatial indexing is highly optimized for smaller datasets.	- Lightning-fast for simple lookups and match queries due to in-memory architecture. - Consistent performance across query types. - Quick data upload times.
Weaknesses	- Slower data upload and query execution for match queries compared to Redis. - Higher memory and CPU usage under load.	- Expensive to scale for large datasets as more RAM is required. - Slightly slower geospatial queries compared to MongoDB's 2D index in this setup.
Use-Case Recommendations	- Ideal for applications requiring complex data relationships, aggregations, and cost-effective scaling. - Suitable for scenarios where data latency isn't critical.	- Best suited for real-time systems requiring low-latency data access, such as caching, gaming, and real-time analytics. - Optimal for smaller datasets where in-memory advantages can be fully leveraged.

Discussion

Findings-

1. Redis was way faster in terms of writing data than MongoDB.
2. MongoDB was faster in terms of Geospatial querying.
3. Setting up Redis is way more complex than MongoDB.
4. Very less community support hence limited learning sources.