It's All Relative: A Performance Comparison of Relational and Non-relational Databases for IP Metadata Lookup

Sabrina Reis
UC Santa Cruz
Santa Cruz, CA USA

1. Introduction

Network researchers frequently require IP address metadata to conduct internet measurements, identify online security risks, assess network reliability, or monitor real-time network conditions. To get this metadata, researchers often rely on cloud-hosted network metadata APIs. One API commonly used by researchers is Whois, which takes an IP address from users and reads from various internet registry databases to provide autonomous system information [1]. Another example used by researchers, engineers, and users alike is the Comcast Outage Map, which takes a residential address and reads from a status database to provide information on service status [2]. These tools were primarily designed for small-scale research or informational purposes rather than heavy concurrent use. This design choice can be problematic when the number of users for these portals spikes, such as when a network event like an outage drives people to seek out more information. Consequently, for these IP metadata lookup tools to be maximally helpful, performance must be a key consideration so that they can provide information in a variety of workload conditions. However, as far as I am aware, the performance of these databases has not been profiled. Without performance testing, it is unclear how resilient these lookup tools can be during periods of unusually high activity.

To address this knowledge gap, I create prototype databases on Google Cloud Platform (GCP) using real IP metadata aggregated from IPInfo and RouteViews. To ensure thorough coverage in the scope of my analysis, I present both a relational database using MySQL and a non-relational database using MongoDB. To simulate the use case of an IP metadata lookup API, I test the databases with a read-only workload corresponding to the Yahoo! Cloud Serving Benchmark (YCSB) workload C. To understand how the databases perform in a variety of workload conditions, I subject each database to realistic soak, constant load, stress, and spike tests. I also develop a scaling model for each database to analyze the cost-performance tradeoffs of horizontal scaling. The results from my experiments provide comprehensive insight into the performance of both relational and non-relational databases in a read-only context for retrieving big data. Source code and additional project information can be found at https://github.com/sabrina-reis/cloud_computing_final.

2. Background and State-of-the-Art

After decades of research, the database management systems community has reached several general conclusions regarding relational and non-relational database performance. As recently summarized in the survey by Samarta et al., prior work has found that relational databases deliver better performance on structured data, especially when queries are complex, but exhibit poor scalability and flexibility on unstructured data [3]. Inversely, non-relational databases are better suited for unstructured data, for which they can provide better performance and support [3].

Some work has disputed the idea that relational databases always perform better on structured data. The most robust and relevant challenge to this idea appears in Eyada et al., which evaluated the performance of MySQL and MongoDB cloud databases for IoT data using read- and write-heavy workloads [4]. The scope of the authors' work was markedly broader than this project; they not only conduct many performance tests with industry-scale workloads on a variety of cloud architectures, but also present a prediction model to estimate the response time for queries on different database configurations and cloud architectures. The authors conclude that MongoDB generally provides better performance than MySQL. However, my workload is read-only, necessitating additional work to profile relational and non-relational databases in this specific context. Additionally, Eyada et al. compare more advanced configurations of MongoDB and MySQL that promise better performance. For cloud users who are not database systems researchers, ease of use is a key priority. Consequently, my performance evaluation of off-the-shelf database models on a popular cloud platform provides practical insight for a broad audience of users.

3. Approach/Hypothesis

3.1. Databases

I deploy my databases on GCP because prominent network measurement databases like Censys use GCP to host their data, making it the state-of-the-art [5]. To best simulate an IP metadata database, I draw on my pre-existing database of IP metadata, which aggregates data from popular measurement platforms RouteViews and IPInfo [6] [7]. In terms

of database implementations, I use MySQL for the relational test and MongoDB for the non-relational test since these database formats are well-supported by GCP. The original IP database contained 1.4 billion lines representing all public IP addresses, but I limit my test databases to 50 million lines due to time, storage, and compute limitations. The test databases contain the fields shown in Table 1.

TABLE 1.
FIELDS OF INFORMATION INCLUDED IN IP TEST DATABASES

Field	Type
IP	String
Network	String
Network City	String
Network Region	String
Network Country Code	String
Network Postal Code	String
IP Binary Representation	Integer
Autonomous System Number	Integer
Autonomous System Organization	String

3.2. Cloud Architecture

The test databases exist on VM instances that support either MongoDB and MySQL. The specifications of these VMs, outlined in 2, align with GCP recommendations and default settings in the setup menus. VM instances are provisioned and de-provisioned by managed instance groups (MIGs), as shown in 1. When average CPU utilization exceeds 70%, the MIG triggers the auto-scaling procedure. Because GCP does not support more fine-grained analysis of database stress, I am limited to using CPU for auto-scaling. Each MIG allows up to 8 VM instances when auto-scaling is enabled.

Prior to testing, I extract a sample of 100,000 IPs from the test database and store the sample IPs in a text file. During testing, K6 test scripts randomly select a number between 0 and 1, multiply this number by 100,000, and then access the IP at the line number of the result. K6 then sends this IP as a `SELECT` query to the appropriate load balancer, which distributes this query to a VM instance. To balance workloads between VMs, the load balancers use TCP, which is currently the most relevant load balancing technique offered on GCP for both databases. While GCP offers a database-aware load balancer for MySQL, it does not provide one for MongoDB. Due to time constraints, I could not create a custom load balancer for MongoDB to correspond with the MySQL load balancer, so I use TCP for both to maintain consistency. After the load balancer distributes the query to a VM, the database driver on the VM instance executes the query and returns the result to k6. Throughout the testing process, the Grafana VM sends metrics collected by Cloud Monitoring to the Grafana portal for observation. The specific versions of all software used in the testing process can be found in Table 3.

TABLE 2.
SPECIFICATIONS OF VMS

VM Type	MongoDB	MySQL
Machine Type	e2-small	e2-small
OS	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS
Disk Type	Standard persistent disk	Standard persistent disk
Storage Size	50 GB	50 GB

TABLE 3.
SOFTWARE VERSIONS

Software	Version
MySQL	8.0.44-0
MongoDB	12.3.0
Grafana	12.3.0
K6	1.4.2
xk6-SQL	1.0.5
xk6-mongo	0.0.5
Go	1.25.1

3.3. Hypothesis

In testing the performance of relational and non-relational IP metadata in a variety of read-only workload conditions, I aimed to answer the following questions:

- Under which workload conditions do relational databases deliver better performance for efficient IP metadata lookup?
- Under which workload conditions do non-relational databases deliver better performance for efficient IP metadata lookup?
- How can we characterize the cost-performance tradeoff for achieving specific latency and throughput targets under varying workload conditions?

Based on my experience performing query-heavy workloads on MySQL data for network research, I hypothesized that SQL would deliver better general performance but was unsure about how MySQL and MongoDB would compare in specific conditions like a sudden spike in users. Regarding the cost-performance tradeoff, I hypothesized that horizontal scaling would improve latency and throughput based on my lectures. I predicted that these improvements would come at

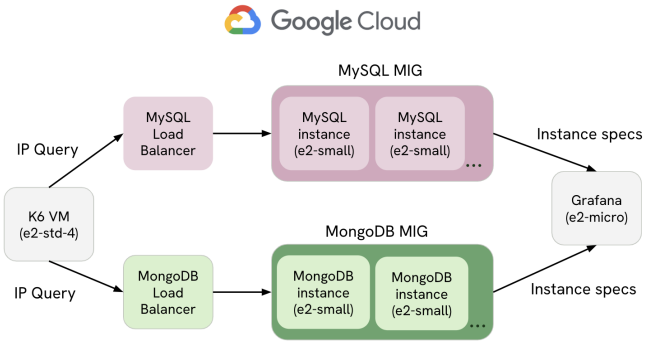

Figure 1. A depiction of the project workflow and architecture on Google Cloud Platform.

Test	Users	Ramp Up	Test	Cool Down
Endurance	50	2 min	4 hr	2 min
Constant	50	2 min	30 min	2 min
Spike	300	0 min	2 min	1 min
Stress	100	10 min	30 min	5 min

TABLE 4. CONFIGURATIONS FOR EACH LOAD TEST. EACH RAMP UP PERIOD HAD THE SAME NUMBER OF USERS AS THE TEST PERIOD. EACH COOL DOWN PERIOD HAD A TARGET OF 0 USERS.

a minimal cost due to the relatively low price of the e2-small machines used by the VMs.

3.4. Load Testing

To profile the performance of the databases, I conducted endurance, constant, spike, and stress tests. Each test simulated a workload with 100% reads corresponding to the YCSB workload C to match the intended use case of an IP metadata lookup API. The configurations for every test are shown in 4. I set the number of target users and duration of each stage using the load test guidelines developed by Grafana [8] [9] [10] [11] [**?**]. With each test, I collected latency and throughput data from K6, as well as average CPU utilization and average disk I/O latenc with Grafana and Cloud Monitoring. To ensure that tests were running as expected, I also monitored bytes read per second in Grafana.

To determine a realistic number of concurrent users, I used site traffic data from the Censored Planet Observatory, a cloud-hosted website lookup API developed by my advisor that tracks the results of censorship scans [12]. Based on Censored Planet usage, a similar research-oriented informational portal can expect 20-25 users per hour. According to SEMRUSH site traffic data, longstanding and authoritative lookup portals like WhoIs receive much heavier site traffic, with approximately 1200 concurrent users per hour [13]. My research-oriented use case is likely closer to Censored Planet than WhoIs, so I conducted load testing with 25 users, but heavier loads are tested for the scaling model. Additionally, if a database struggles with a lower workload similar to Censored Planet, we can likely assume that the performance issues will be exacerbated at a higher load that more closely resembles WhoIs usage.

3.5. Scaling Model

To assess the performance-cost tradeoff of horizontal scaling, I performed tests to gather information on how latency and throughput varied according to the number of instances. To empirically establish a baseline for the number of users that one instance could support, I performed 12 tests with increasing numbers of users. By monitoring CPU utilization and average disk I/O latency over several tests, I determined that one database instance started to approach saturation around 120 concurrent users, as latency began to climb above 500ms and CPU utilization increases over 70%. Figures 2 and 3 display these trends. The near saturation point of 120 virtual users held for both MongoDB and MySQL. With this insight, I then tested 2, 4, and 8 instances of MySQL and MongoDB VMs with 120 concurrent users and gathered metrics on latency, throughput, and CPU utilization using K6 and Grafana. I fit a linear regression to the overall throughout data to determine if throughput scales linearly, sub-linearly, or super-linearly. To provide more context for these constant load tests, I also tested 2, 4, and 8 instances with heavier workloads of concurrent users.

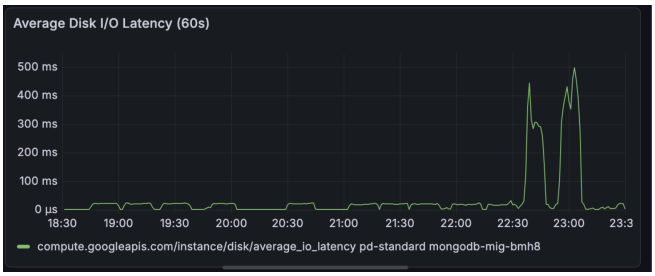

Figure 2. MongoDB disk I/O latency during tests conducted with monotonically increasing virtual user counts to establish a near-saturation point for the scaling model.

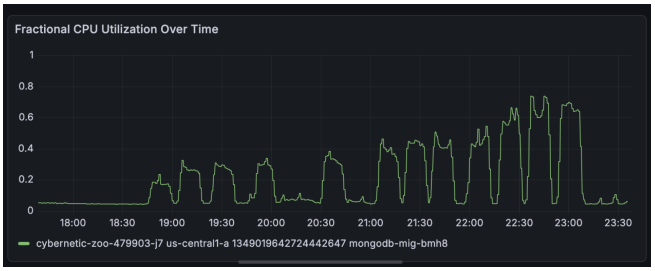

Figure 3. Fractional CPU utilization during tests conducted with monotonically increasing virtual user counts to establish a near-saturation point for the scaling model.

4. Resources Needed

Most required software can be found in Table 3. Additional custom software includes a Python script to randomly select 100,000 IPs from the test database. Required cloud resources include:

- Google Cloud account and compute credits,
- Google Cloud bucket containing IP metadata for import to MySQL and MongoDB VM templates,
- MongoDB and MySQL virtual machines (hardware described in Table 2),
- Managed Instance Groups for MongoDB and MySQL virtual machines,
- VM for running K6 and K6 test scripts,
- Load balancers for distributing K6 queries to MySQL and MongoDB MIGs,
- Cloud Monitoring service account to collect VM metrics, and
- VM for transmitting Cloud Monitoring metrics to Grafana.

5. Timeline

5.1. Architecture Setup

November 30th-December 2nd, 10 hours
I began the testing process by truncating my original MySQL database to 50 million lines and exporting the truncated data. To transfer my IP metadata to the cloud, I uploaded the original database to a cloud storage bucket. I then created VMs for running MySQL and MongoDB, imported the data from the storage bucket, converted the data to the appropriate database format, and saved the VMs as an instance template. After, I created managed instance groups for MySQL and MongoDB to enable VM autoscaling. Finally, I created the load balancers for distributing traffic across VM instances. Lastly, I created a VM for running K6 and installed and configured K6.

5.2. Measurement Setup

December 3rd-December 4th, 3 hours To collect metrics from my system, I set up a VM instance, installed and configured Grafana, and enabled a monitoring viewer in Cloud Monitoring. Afterward, I created a Grafana account and added Google Cloud Monitoring as a data source for Grafana. I then explored many of the metrics to determine which were viable before choosing bytes read per second, CPU utilization, and average disk I/O latency. With Grafana working, I began referencing Grafana resources on performance engineering to develop k6 test scripts.

5.3. Testing

December 4th-December 8th, 20 hours With the architecture in place, I began testing. The process was difficult at first, as I had never used Grafana or K6 before. I originally provisioned the k6 VM to be e2-micro and had to re-provision the VM multiple times for k6 to have enough compute power to generate sufficient VUs for performance modeling, stress testing, and spike testing. I also had to re-run tests for many reasons. For instance, I realized after a day that k6 did not collect P(99) latency by default and needed to redo several tests to collect that information.

5.4. Analysis and Reporting, 15 hours

December 8th-December 11th After I finished the test results and gathered the screenshots, I began analyzing and writing up the experiment design and conclusions for the report and presentation. In total, I estimate that I spent approximately 50 hours on this project.

6. Discussion

6.1. Workload Testing

6.1.1. Endurance Testing. Based on the throughput and latency metrics reported in Table 5, we can see that MySQL exhibits better performance in terms of both latency and throughput over the four hour test period. In Figures 6 and 4, we observe that MySQL instances decreased their disk I/O latency over time, which led to an increase in the number of bytes read per second, thereby improving query throughput. Figure 5 demonstrates that with higher query throughput, CPU utilization increased without exceeding healthy levels, allowing latency to remain low. In comparison, disk I/O latency did not improve over time on MongoDB VM instances, resulting in relatively lower throughput and higher latency. CPU utilization remained lower in MySQL, but because this did not result in lower latency or higher throughput, this result suggests that disk I/O latency presented a bottleneck for performance in MongoDB.

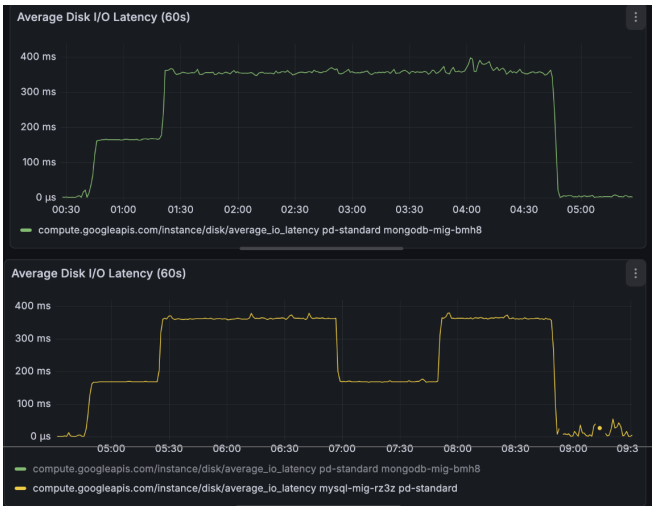

Figure 4. Average disk I/O latency over 60 seconds for MongoDB and MySQL instances during endurance testing

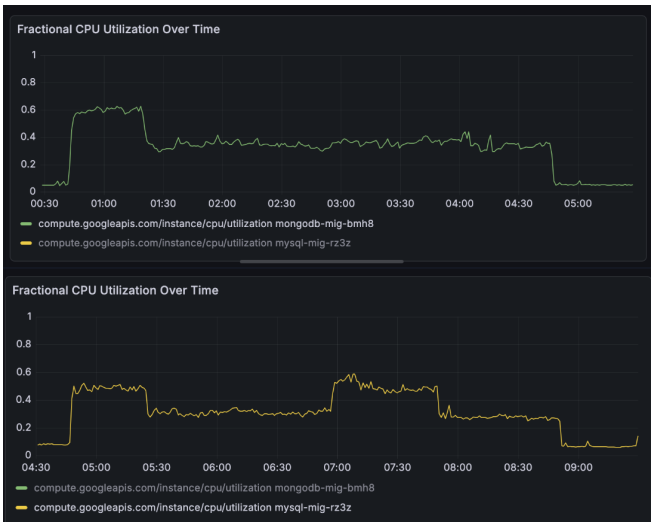

Figure 5. Fractional CPU utilization of MongoDB and MySQL instances during endurance testing.

Figure 6. Bytes read per second by MongoDB and MySQL instances during endurance testing.

TABLE 5. P(99) LATENCY, P(95) LATENCY, AVERAGE LATENCY, AND THROUGHPUT FOR ENDURANCE WORKLOAD TESTS CONDUCTED ON MONGODB AND MYSQL DATABASES.

DB Type	P(99)	P(95)	Avg. Latency	Throughput/s
MongoDB	946.71ms	788.17ms	298.35ms	83.11
MySQL	782.06ms	739.51ms	249.82ms	99.25

6.1.2. Constant Testing. As evidenced by the metrics in Table 6, MongoDB exhibited slightly higher throughput and slightly lower average query latency than MySQL while supporting a constant load of users for 30 minutes. However, MongoDB had higher P(95) and and P(99) query latency than MySQL, indicating that it struggles with straggler queries. The higher query latency in MySQL is supported by 7 showing that disk I/O latency was consistently higher in MySQL. The high query and disk I/O latency likely produced the lower fractional CPU utilization in MySQL shown in 8, though CPU utilization for both databases remained within healthy levels.

TABLE 6. P(99) LATENCY, P(95) LATENCY, AVERAGE LATENCY, AND THROUGHPUT FOR CONSTANT WORKLOAD TESTS CONDUCTED ON MONGODB AND MYSQL DATABASES.

DB Type	P(99)	P(95)	Avg. Latency	Throughput/s
MongoDB	415.24ms	366.08ms	138.37ms	158.50
MySQL	397.14ms	363.70ms	155.16ms	141.36

6.1.3. Stress Testing. Similar to the constant load tests, throughput and latency were generally better in MongoDB VM instances than in MySQL instances, as reported in 7. Once again, MongoDB had slightly higher throughput and slightly lower average latency than MySQL. MongoDB still had higher P(95) query latency than MySQL, but had lower P(99) latency, suggesting fewer issues handling straggler queries. Average disk I/O latency was similarly high for both MongoDB and MySQL, as shown in Figure 9. This finding

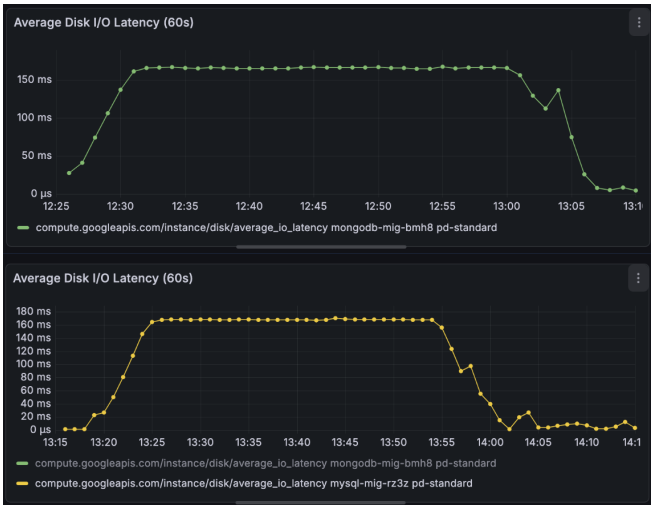

Figure 7. Average disk I/O latency over 60 seconds for MongoDB and MySQL instances during constant testing

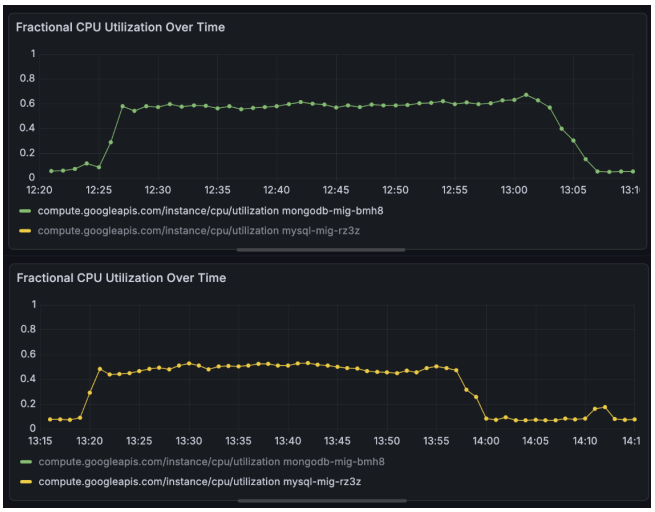

Figure 8. Fractional CPU utilization of MongoDB and MySQL instances during constant testing.

establishes that the higher latency values in MySQL were primarily caused by inefficient query processing. Finally, Figure 10 demonstrates that MongoDB VM instances had slightly higher CPU utilization than MySQL, likely due to higher throughput. From these results, we can conclude that MongoDB is generally more resilient to moderate length periods of stress.

TABLE 7. P(99) LATENCY, P(95) LATENCY, AVERAGE LATENCY, AND THROUGHPUT FOR STRESS WORKLOAD TESTS CONDUCTED ON MONGODB AND MYSQL DATABASES.

DB Type	P(99)	P(95)	Avg. Latency	Throughput/s
MongoDB	2.47s	1.54s	588.06ms	141.83
MySQL	2.70s	1.40s	618.47ms	134.86

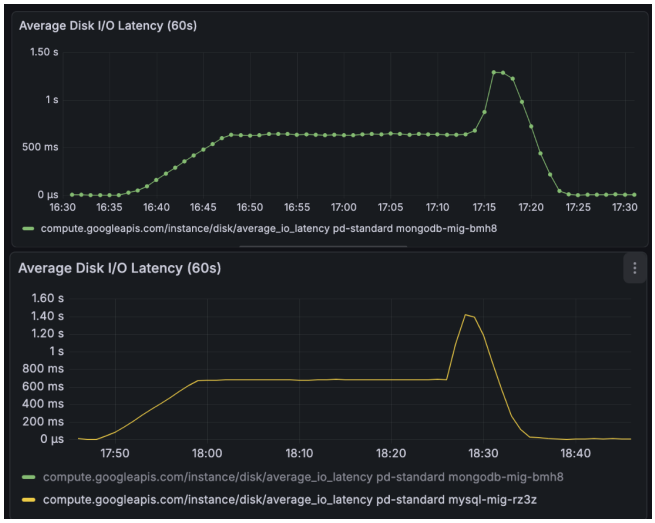

Figure 9. Average disk I/O latency over 60 seconds for MongoDB and MySQL instances during stress testing

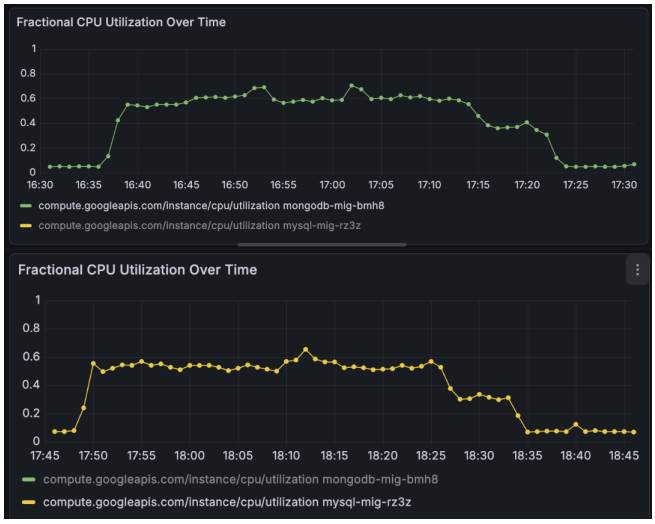

Figure 10. Fractional CPU utilization of MongoDB and MySQL instances during stress testing.

6.1.4. Spike Testing. The latency and throughput metrics from spike testing displayed in Table 8 demonstrate that MySQL was clearly more resilient to spikes, with dramatically higher throughput (1292.47/s vs. 109.92/s) and dramatically lower average latency, P(95) latency, and P(99) latency. As Figure 11 illustrates, both MongoDB and MySQL have similar maximum average disk I/O latency of approximately 300ms. However, MySQL approaches this maximum faster and maintains the maximum for longer, indicating that the much lower query latency in MySQL is largely due to more efficient query processing. The much higher throughput in MySQL leads to much higher levels of CPU utilization shown in Figure 12, including at dangerous levels. Though k6 did not report that any checks in the spike test failed, when CPU utilization approached 100%, k6 returned errors stating that there were too many concurrent users. I am unsure if k6 retried these queries or dropped them entirely, as the documentation is unclear, but a drop would be inconsistent with k6 reporting that no queries failed.

Figure 11. Average disk I/O latency over 60 seconds for MongoDB and MySQL instances during spike testing

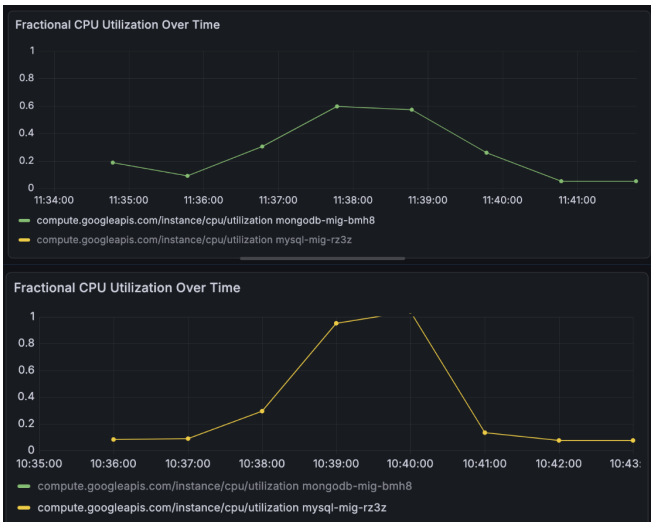

Figure 12. Fractional CPU utilization of MongoDB and MySQL instances during spike testing.

TABLE 8. P(99) LATENCY, P(95) LATENCY, AVERAGE LATENCY, AND THROUGHPUT FOR SPIKE WORKLOAD TESTS CONDUCTED ON MONGODB AND MYSQL DATABASES.

DB Type	P(99)	P(95)	Avg. Latency	Throughput/s
MongoDB	5.28s	3.5s	1.37s	109.92
MySQL	2.05s	783.99ms	89.98ms	1292.47

6.2. Scaling Model

The scaling model tests yielded unexpected results for both MongoDB and MySQL. Surprisingly, MySQL experiences poor scaling with respect to both throughput and latency. Given that the slope in the MySQL linear regression equation is -4.516, as shown in 14, throughput scaling is so poor that it is more accurately described as a negative than as sub-linear. Likewise, increasing the number of instances produced higher latency, as described in 13. MongoDB also experienced negative scaling regarding throughput, with the linear regression producing a slope of -3.855. Figure 16 illustrates how more instances results in lower per-instance throughput and either no change or worsening overall throughput. However, per-instance latency decreased slightly for 2 and 4 instances of MongoDB VMs and increased sharply for 8 instances, as shown in 15. The metrics for all scaling model tests can be found in **??**.

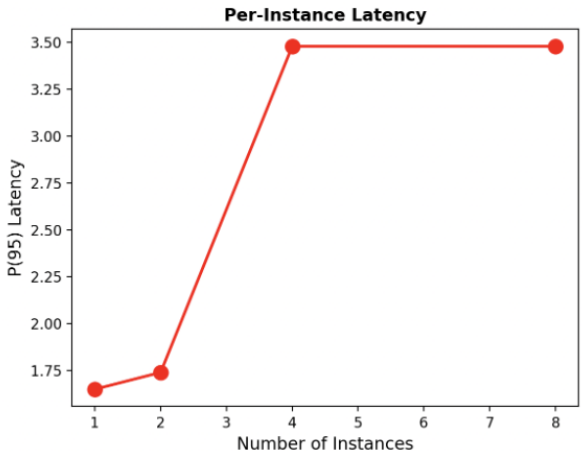

Figure 13. Per-instance latency for MySQL at a constant load of 120 VUs tested on 1, 2, 4, and 8 instances.

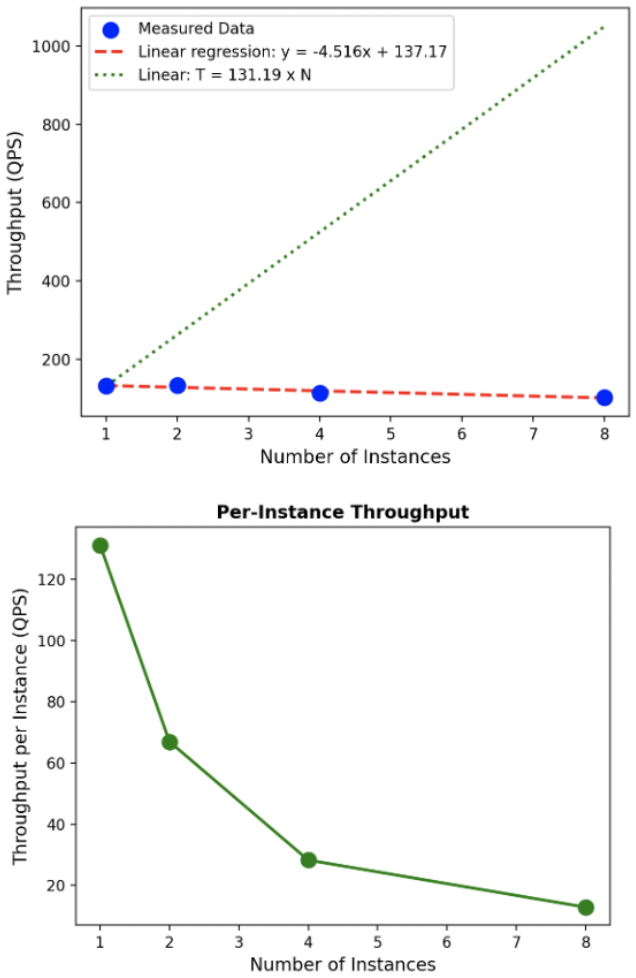

Figure 14. Overall throughput and per-instance throughput for MySQL at a constant load of 120 VUs tested on 1, 2, 4, and 8 instances.

These findings appear to suggest that the overhead involved in managing multiple instances produces worsening throughput and, generally, worse latency for both MySQL and MongoDB. If these results are accurate, then there is little to no performance improvement for the additional costs associated with using more instances. In this case, higher costs would not return any significant performance benefits. Given the surprising lack of any benefit produced by horizontal scaling, however, these results may also be the result of an inadequate load balancer or faulty task generation by k6. We discuss possible steps to investigate and resolve confounding factors in the Discussion.

7. Conclusions and Future Work

Based on the load testing experiments, I believe that MySQL would be best suited for the intended use case of an IP metadata lookup API. This API will be running for long periods of time and will likely experience a spike in users when a network event occurs, making the results of the endurance and spike tests most salient. Even in the tests where MySQL exhibited worse throughput and latency, the differences were fairly minimal. However, due to a lack of clarity in which queries k6 marks as failed, I would recommend vertically scaling MySQL VM instances to add CPUs. If the ``Too many concurrent users warnings indicated that queries were dropped by k6, increasing the number of CPUs on a VM would likely prevent the likelihood of queries being dropped by the spike. I also recommend vertical scaling since the scaling model tests seemed to suggest that horizontal scaling is relatively ineffective.

Due to concerns over the time and compute cost involved in using the original database containing metadata for 1.4 billion IPs, I limited the test dataset size to 50 million lines. To better approximate a real-world IP metadata database, future work could test and profile larger databases that approach the size of the public IP address space. Additionally, given the inconclusive results of the scaling model, future

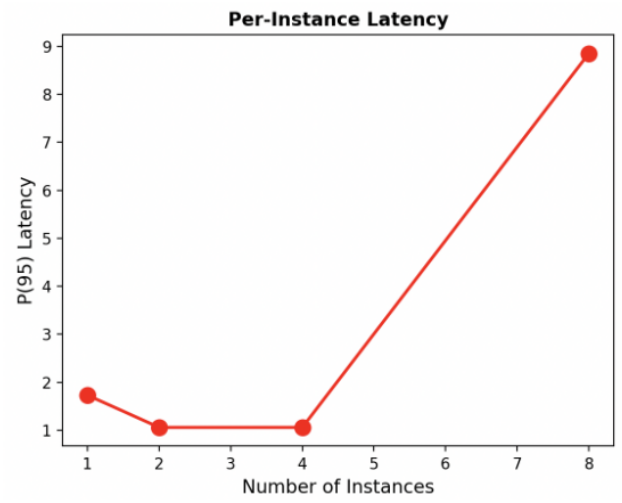

Figure 15. Per-instance latency for MongoDB at a constant load of 120 VUs tested on 1, 2, 4, and 8 instances.

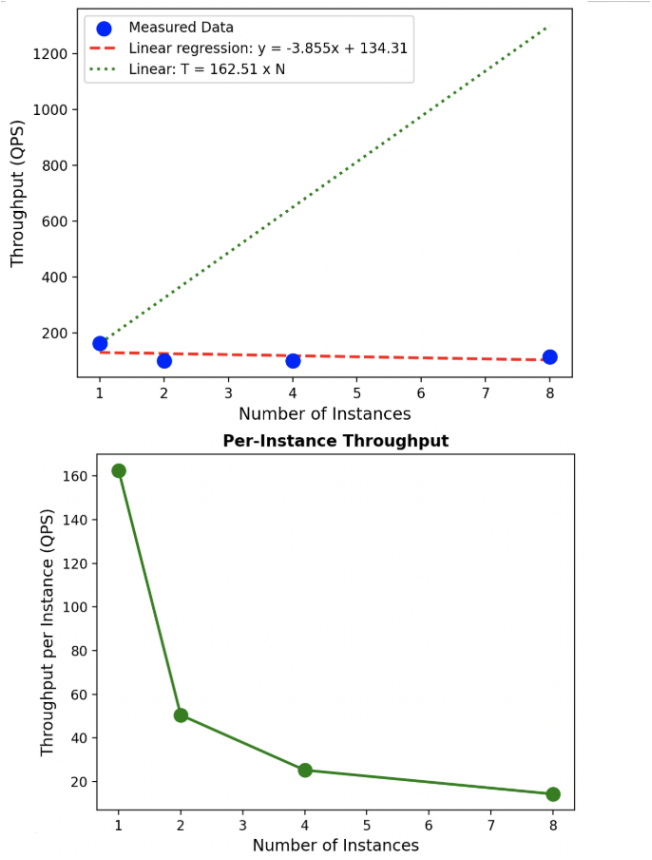

Figure 16. Overall throughput and per-instance throughput for MongoDB at a constant load of 120 VUs tested on 1, 2, 4, and 8 instances.

research could investigate and address potential limitations with k6 or the TCP-based load balancer to improve the scaling model. With more time, building a custom load balancer for MongoDB and using the provided load balancer for MySQL in GCP would likely yield more performance benefits from horizontal scaling.

In spite of these limitations, this work presents a thorough evaluation and analysis of relational and non-relational databases for IP metadata lookup. The conclusion that MySQL delivers the best long-term performance and can endure dramatic spikes in usage provides a clear path forward for my research, which will ultimately produce an IP metadata lookup tool. More broadly, other hosts of read-only database lookup APIs can use these findings to guide their database implementations and cloud workflows to deliver quick and reliable results for many users.

References

[1] Whois, "Whois domain lookup," 2025. [Online]. Available: https://www.whois.com/whois/

[2] Xfinity, "Outage map," 2025. [Online]. Available: https://www.xfinity.com/support/statusmap

[3] M. T. Samarta, A. A. Santoso Gunawan, and M. E. Syahputra, "Systematic literature review and comparative performance analysis of sql and nosql databases in big data applications," in *2024 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, 2024, pp. 218–222.

[4] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance evaluation of iot data management using mongodb versus mysql databases in different cloud environments," *IEEE Access*, vol. 8, pp. 110 656–110 668, 2020.

[5] Censys, "Gcp hosted cloud connector." [Online]. Available: \url{https://docs.censys.com/docs/asm-gcp-cloud-connector}

[6] Network Startup Resource Center, "Routeviews," 2025. [Online]. Available: https://archive.routeviews.org/

[7] IPInfo, "Ipinfo," 2025. [Online]. Available: https://ipinfo.io/

[8] Grafana, "Types of load testing," 2025. [Online]. Available: https://grafana.com/load-testing/types-of-load-testing/

[9] ——, "Soak testing," 2025. [Online]. Available: https://grafana.com/blog/2024/01/30/soak-testing/

[10] ——, "Load testing," 2025. [Online]. Available: https://grafana.com/docs/k6/latest/testing-guides/test-types/load-testing/

[11] ——, "Spike testing," 2025. [Online]. Available: https://grafana.com/docs/k6/latest/testing-guides/test-types/spike-testing/

[12] Censored Planet, "Censored planet observatory," 2025. [Online]. Available: https://dashboard.censoredplanet.org/-observatory.html

[13] SEMRUSH, "Website traffic checker," 2025. [Online]. Available: https://www.semrush.com/website/

Appendix

TABLE 9. METRICS FROM ALL SCALING MODEL TESTS USED TO ASSESS THE COST–PERFORMANCE TRADEOFF OF SCALING.

Database	**Instances**	**Users**	**Avg. Latency**	**P(95) Latency**	**P(99) Latency**	**Throughput (iter/s)**
MongoDB	1	120	707.13ms	1.73s	2.28s	162.51
MySQL	1	120	839.17ms	1.65s	1.69s	131.19
MongoDB	2	50	1.02	1.08	NA	41.73
MongoDB	2	60	1.02	1.08	NA	49.95
MongoDB	2	80	1.02	1.07	NA	67.14
MongoDB	2	100	1.01	1.06	NA	84.15
MongoDB	2	120	1.01	1.06	NA	101.14
MongoDB	2	240	1.52s	3.3s	4.36s	144.9
MySQL	2	120	822.43ms	1.74s	3.49s	133.93
MongoDB	4	120	1.01	1.06	NA	101.03
MongoDB	4	180	1.01	1.06	NA	151.56
MongoDB	4	230	1.25	1.89	NA	157.23
MongoDB	4	375	2.49s	7.31	13.48s	138.27
MySQL	4	120	974.95ms	3.48s	3.57s	113
MongoDB	8	230	1.38	2.15	NA	142.69
MongoDB	8	375	2.57s	7.47s	12.58s	133.75
MySQL	8	120	1.07s	3.48s	3.53s	102.84
MongoDB	8	120	3s	8.85s	15.82s	114.73

TABLE 10. METRICS FOR WORKLOAD TESTS CONDUCTED ON MONGODB AND MYSQL DATABASES.

DB	Test	P(99)	P(95)	Avg. Latency	Throughput/s
MongoDB	Soak	946.71ms	788.17ms	298.35ms	83.11
MySQL	Soak	782.06ms	739.51ms	249.82ms	99.25
MongoDB	Spike	5.28s	3.5s	1.37s	109.92
MySQL	Spike	2.05s	783.99ms	89.98ms	1292.47
MongoDB	Constant	415.24ms	366.08ms	138.37ms	158.50
MySQL	Constant	397.14ms	363.70ms	155.16ms	141.36
MongoDB	Stress	2.47s	1.54s	588.06ms	141.83
MySQL	Stress	2.70s	1.40s	618.47ms	134.86