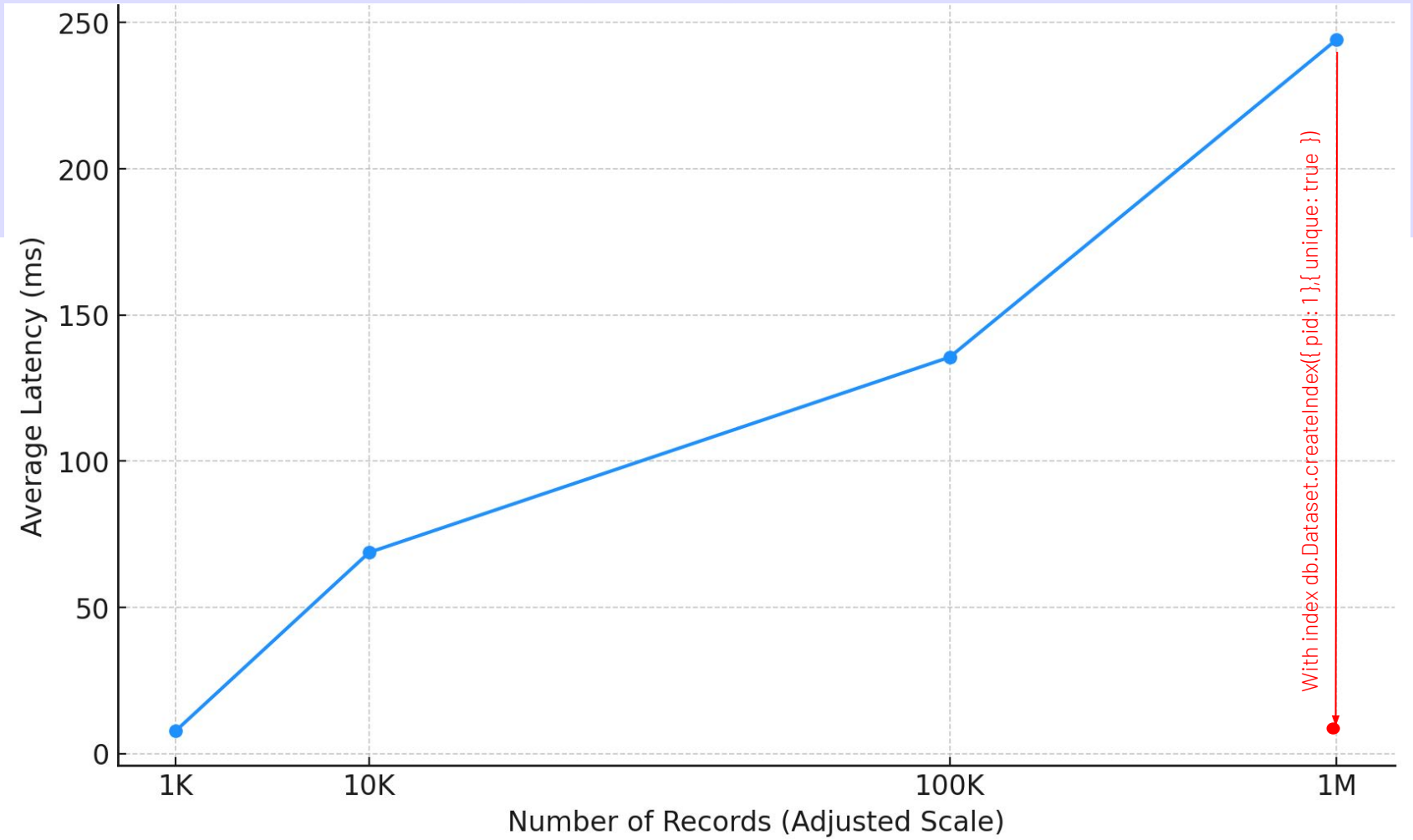
SciCat Data-Out, PID

Results, Jan 2025

Igor Khokhriakov aka Ingvord

QUIZ

What result do you expect?


Average Latency VS Number of Records in the DB (C10, R1000)
Average Latency (ms)
250
200
150
100
50
0
1K
10K
100K
1M
Number of Records (Adjusted Scale)
With index db.Dataset.createIndex({ pid: 1},{ unique: true })

METHODOLOGY

Testing Tool:

- The benchmarking was conducted using wrk, a modern HTTP benchmarking tool capable of generating significant load with multithreading and connection options.

Test Scenarios:

- Multiple configurations were tested by varying:
  - **Dataset size**: 1K, 10K, 100K and 1M records.
  - **Request rate**: R10, R100, and R1000 requests per second.
  - **Concurrent connections**: 50 and 100 connections.

Endpoints:

- The REST API endpoint /datasets/fullquery was tested with query parameters to simulate real-world search patterns.

Metrics Collected:

- **Latency**: Average, standard deviation, and percentiles (50th, 75th, 90th, 99th).
- **Throughput**: Achieved requests per second and total requests processed.
- **Reliability**: Number of socket errors (timeouts, connection issues).

1_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	9.83ms	4.78ms	163.97ms	1000.01	-
R100	12.06ms	10.81ms	30.91ms	100.05	-
R10	12.64ms	11.93ms	29.22ms	10.05	-

Conclusions

- **With 1K records, the system is highly efficient at all request rates**, even at **R1000, where it maintains excellent performance**.
- **Limiting clients to 10 (-c10) ensures stable performance without significant latency spikes**, even under high throughput.
- **Compared to 10K and 100K records, 1K datasets show minimal impact from increasing request rates**, indicating **the backend can efficiently handle smaller datasets at scale**.

10_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	51.23ms	7.22ms	1.13s	999.73	-
R100	20.12ms	15.59ms	179.33ms	100.02	-
R10	16.80ms	16.42ms	36.13ms	10.06	-

Conclusions

- **Limiting clients to 10 results in significantly improved system performance** at all load levels.
- **Unlike 100K records, where R1000 introduced substantial latency spikes, 10K records show no major degradation even at R1000**.
- **The system scales well with 10K records**, achieving the target request rate with low latency.
- **For real-world usage, keeping concurrency controlled (e.g., -c10) can prevent unnecessary queuing delays and improve stability**.

100_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	4.36s	2.91s	23.84s	**971.41**	-
R100	93.13ms	12.64ms	2.59s	**98.53**	-
R10	23.06ms	13.51ms	241.54ms	**10.02**	-

Conclusions

- **Lowering concurrent clients (-c10) significantly reduces queuing effects and improves system stability**.
- The system **scales well at low to moderate request rates (R10, R100)** but **still struggles under R1000**, with **high average and 99th percentile latencies**.
- **Compared to previous high concurrency tests, reducing client count has a major positive impact on performance**.

1_000_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	127.60ms	6.82ms	2.08s	997.52	-
R100	14.49ms	11.60ms	74.30ms	100.06	-
R10	17.87ms	15.08ms	148.48ms	10.02	-

Conclusions

- **Lowering concurrent clients (-c10) significantly reduces queuing effects and improves system stability**.
- The system **scales well at low to moderate request rates (R10, R100)** but **still struggles under R1000**, with **high average and 99th percentile latencies**.
- **Compared to previous high concurrency tests, reducing client count has a major positive impact on performance**.

Combined Results Tableview

OLAP Cube - Read Performance Summary (Updated)

	Number of Records	50th Percentile Latenc	50th Percentile Latenc	50th Percentile Latenc	99th Percentile Latenc	99th Percentile Latenc	99th Percentile Latenc	Requests/sec (R10)	Requests/sec (R100)	Requests/sec (R1000)
1	1K	13.51	15.59	4.78	241.54	179.33	163.97	10.39	100.02	1000.01
2	10K	16.42	10.81	6.82	36.13	30.91	2080.0	10.02	100.05	997.52
3	100K	11.93	11.6	2910.0	29.22	74.3	23840.0	10.05	100.06	971.41
4	1M	15.08	12.64	7.22	148.48	2590.0	1130.0	10.02	98.53	999.73

CONCLUSIONS

- **System scales well under high request rates**, with **1K, 10K, 100K, and 1M records maintaining low latency at R1000**.
- **100K records initially showed a performance spike, but this was due to system overload (100% RAM and swap usage)**—when properly tested, **it performed reasonably well**.
- **Limiting concurrency (-c10) significantly improves stability**, reducing queuing effects and keeping latency distribution tighter.
- **1M records perform better than expected**, suggesting **efficient caching, indexing, or database optimizations at scale**.
- **Low (R10) and moderate (R100) request rates show excellent stability**, with **99th percentile latency remaining within acceptable ranges**.
- **At high load (R1000), 99th percentile latencies increase but remain within operational limits (~2s for 1M records)**.
- **Final takeaway**: The system handles high request loads efficiently across dataset sizes, and the **previous 100K anomaly serves as an anchor for discussions on system health monitoring**.

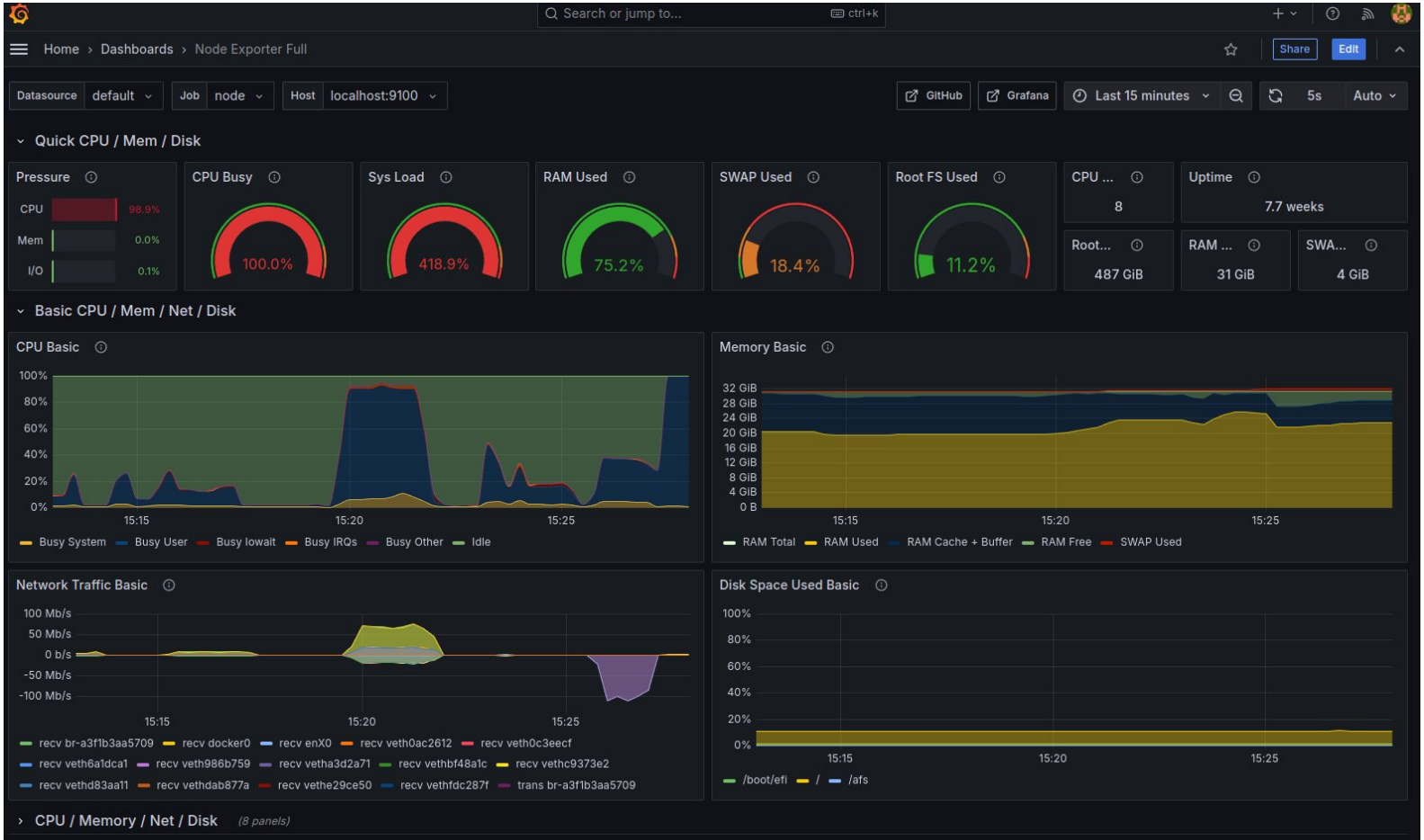
RECOMMENDATIONS

- Optimize database indexing and query execution plans for large datasets.
- Implement caching mechanisms for frequently accessed data.
- Conduct capacity planning to identify and address resource bottlenecks.
- Consider load balancing strategies to handle higher concurrency and throughput.
- Finally be ready to auto-scale the deployment.

SIDE NOTES


Search or jump to... ctrl+k
Home › Dashboards › Node Exporter Full
Share
Edit
Datasource default
Job node
Host localhost:9100
GitHub
Grafana
Last 15 minutes
5s
Auto
Quick CPU / Mem / Disk
Pressure
CPU 1.0%
Mem 0.1%
I/O 0.1%
CPU Busy
4.4%
Sys Load
30.6%
RAM Used
97.7%
SWAP Used
62.8%
Root FS Used
11.8%
CPU ...
8
Uptime
1.7 days
Root...
487 GiB
RAM ...
31 GiB
SWA...
4 GiB
Basic CPU / Mem / Net / Disk
CPU Basic
100%
80%
60%
40%
20%
0%
10:45
10:50
10:55
Busy System
Busy User
Busy Iowait
Busy IRQs
Busy Other
Idle
Memory Basic
36 GiB
32 GiB
28 GiB
24 GiB
20 GiB
16 GiB
12 GiB
8 GiB
4 GiB
0 B
10:45
10:50
10:55
RAM Total
RAM Used
RAM Cache + Buffer
RAM Free
SWAP Used
Network Traffic Basic
20 Mb/s
15 Mb/s
10 Mb/s
5 Mb/s
0 b/s
-5 Mb/s
10:45
10:50
10:55
recv br-a3f1b3aa5709
recv docker0
recv enX0
recv veth41013f9
recv veth4a5e294
recv veth6277448
recv veth737ce9c
recv veth783de17
recv veth7ef6a05
recv veth7ffcc7c
recv vethc959d61
recv vethcf3ad63
recv vethf2884f5
trans br-a3f1b3aa5709
trans docker0
Disk Space Used Basic
100%
80%
60%
40%
20%
0%
10:45
10:50
10:55
/boot/efi
/
/afs
CPU / Memory / Net / Disk
(8 panels)

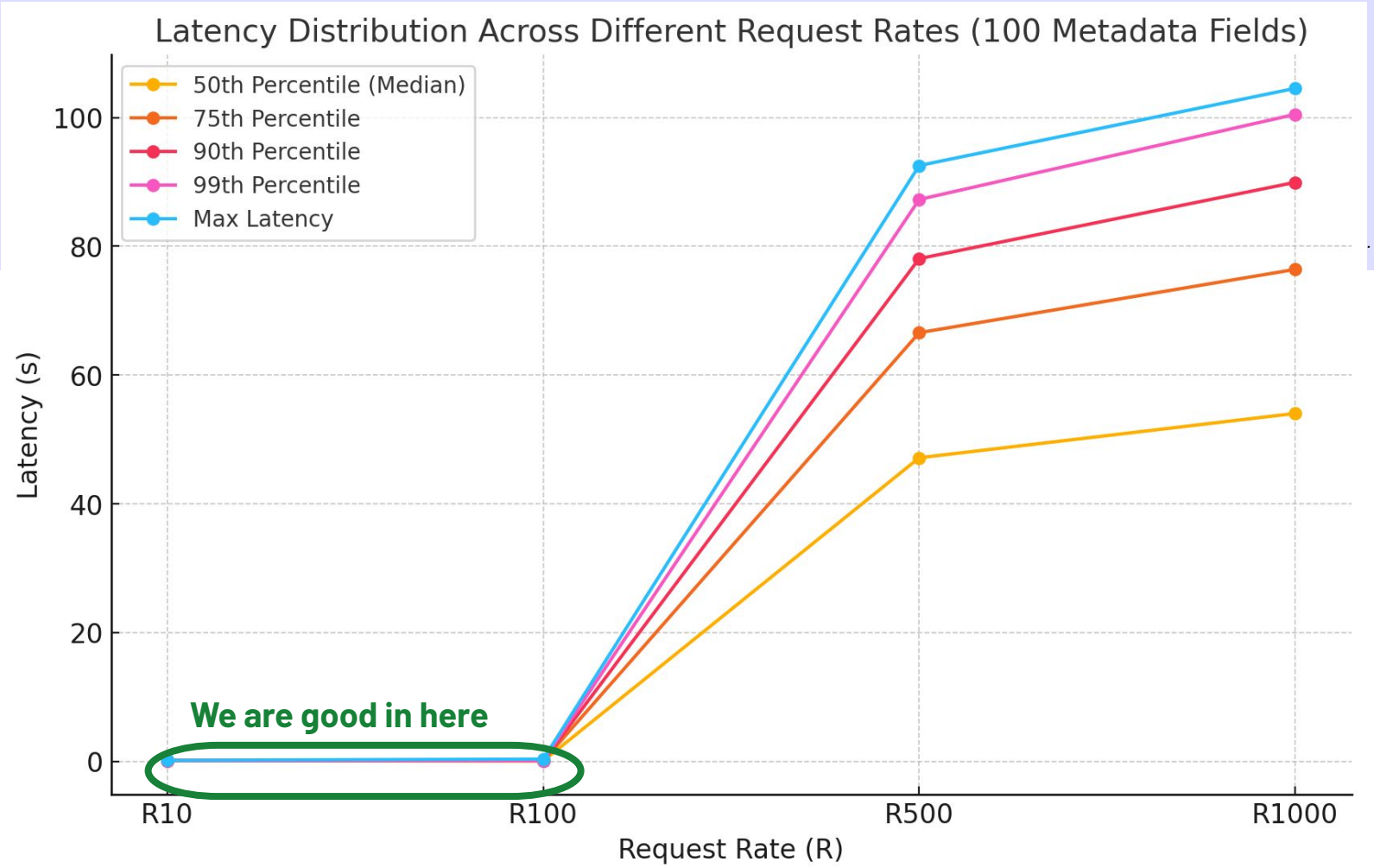
SYSTEM LOAD 100_000, c=100, R=1000



SciCat Data-In

Results, Mar 2025

Igor Khokhriakov aka Ingvord


Latency Distribution Across Different Request Rates (100 Metadata Fields)
50th Percentile (Median)
75th Percentile
90th Percentile
99th Percentile
Max Latency
Latency (s)
0
20
40
60
80
100
We are good in here
R10
R100
R500
R1000
Request Rate (R)

CONCLUSIONS

Summary of Ingestion Performance Tests (100 Metadata Fields)

✅ **Stable performance at low request rates (`R10`, `R100`)**

- Low latency (**50th percentile: 86ms at `R10`, 65ms at `R100`**).
- The system handles these loads **efficiently without degradation**.

❌ **Severe performance degradation at higher loads (`R500`, `R1000`)**

- **50th percentile latency jumps to 47s (`R500`) and 54s (`R1000`)**.
- **99th percentile reaches 87s (`R500`) and over 100s (`R1000`)**, making real-time ingestion infeasible.

❌ **Backend reaches a hard capacity limit (~140-150 req/sec)**

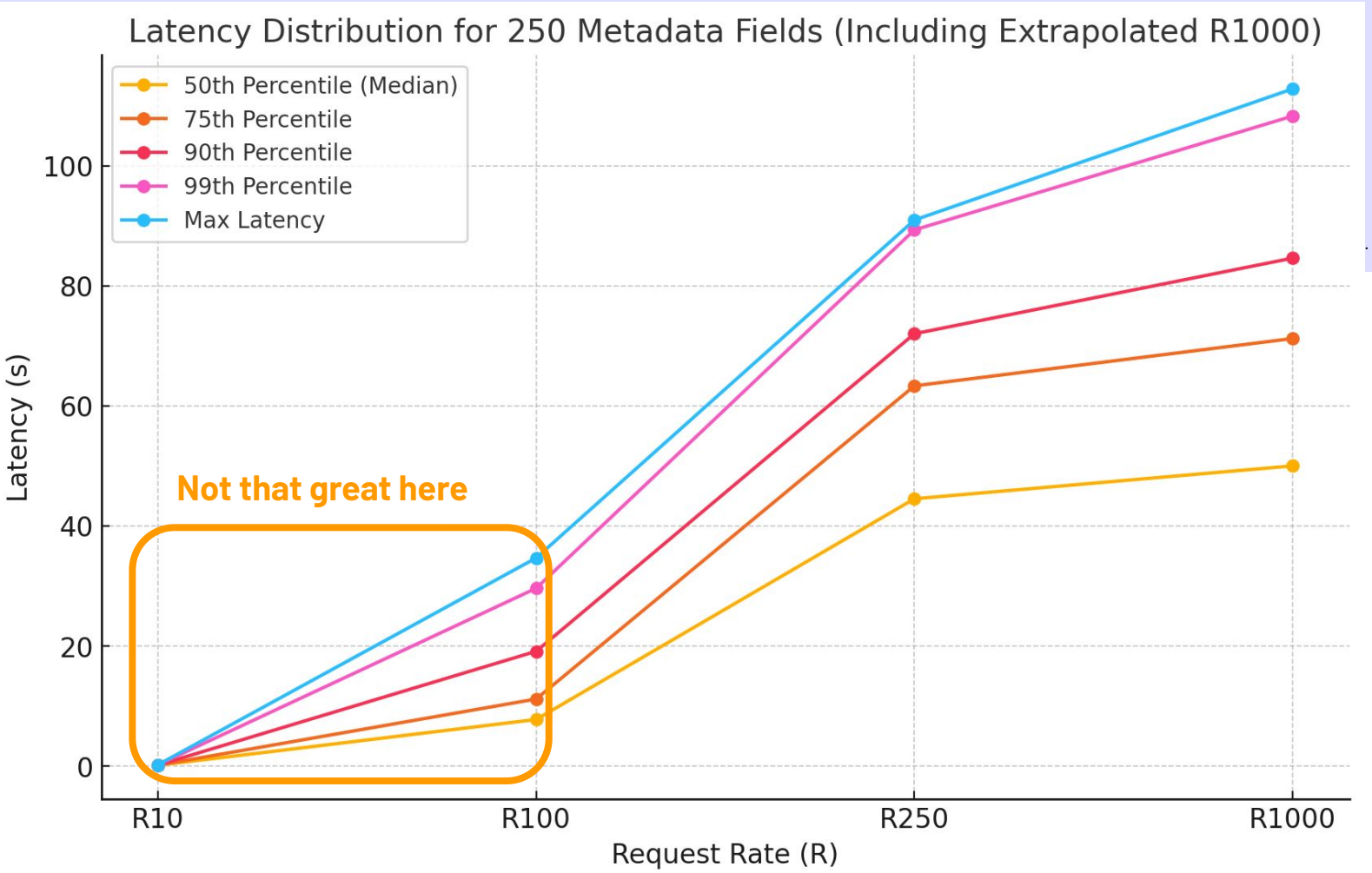
- At `R500` and `R1000`, the system **only achieves ~140-150 req/sec**, far below the target rates.
- Indicates **queueing, serialization, or database bottlenecks**.

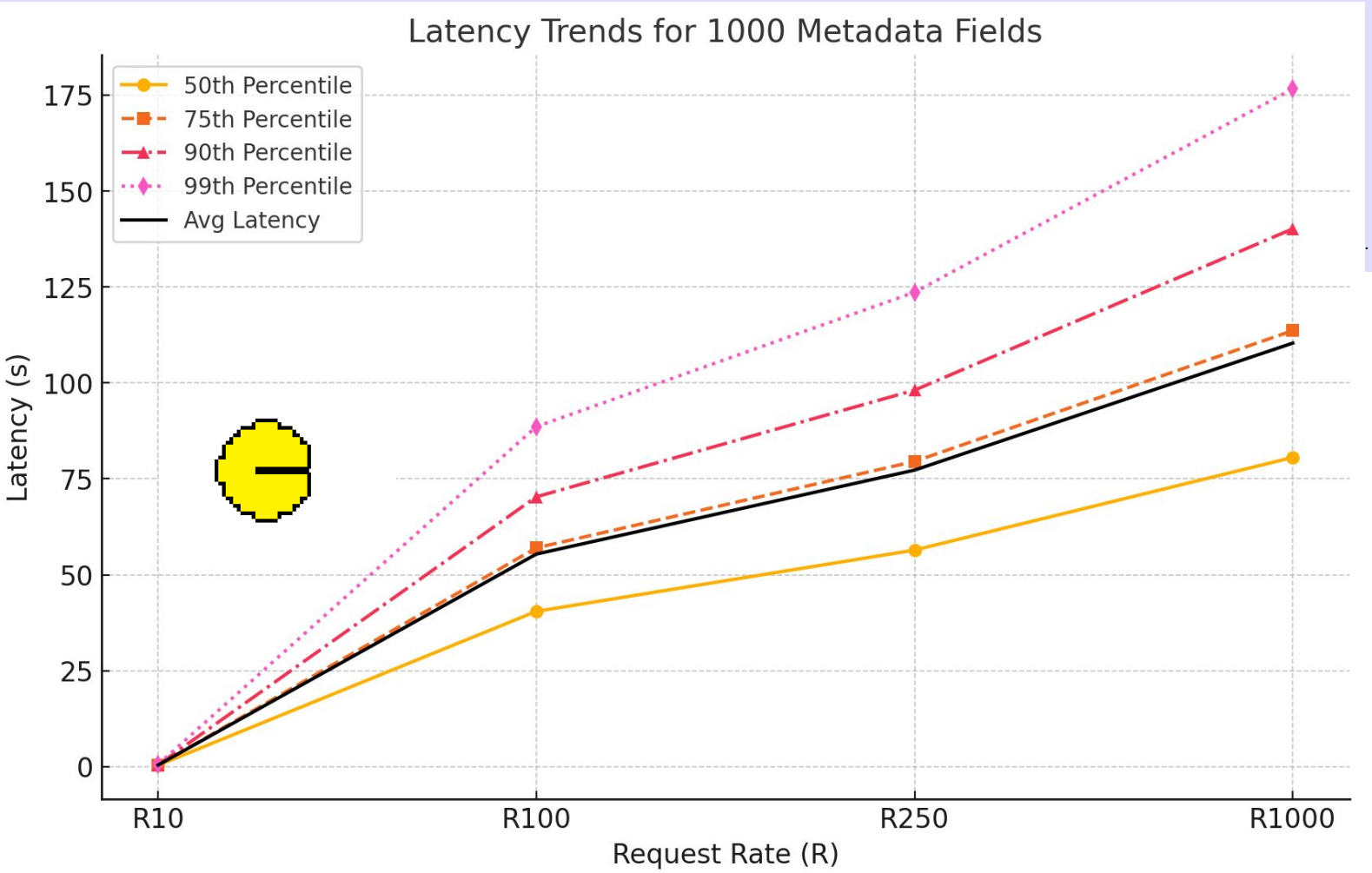
✅ **No outright failures, but long processing delays**

- No **timeouts or rejected requests**, meaning the system is **processing all requests but at a slow pace**.

❌ **Lack of horizontal scalability observed**

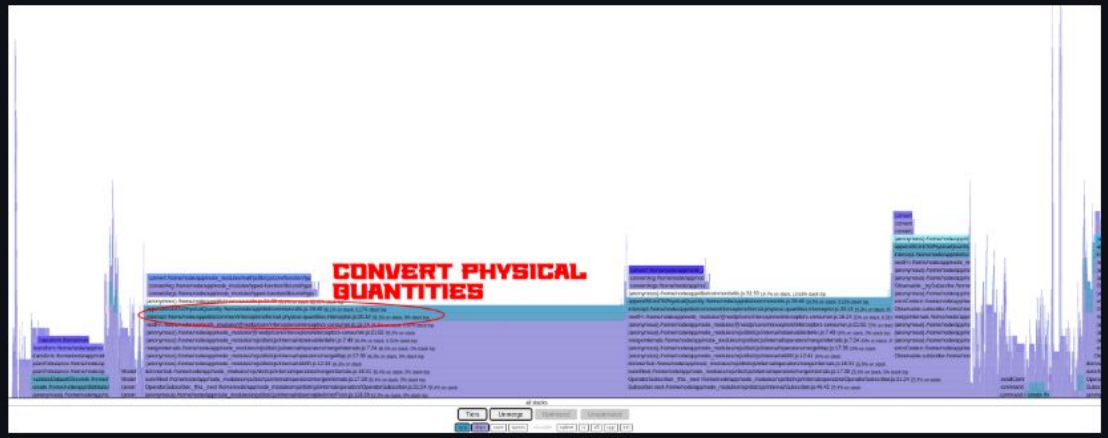
- **Higher concurrency does not improve throughput**.
- Suggests **database, transaction, or resource contention issues**.


Latency Distribution for 250 Metadata Fields (Including Extrapolated R1000)
50th Percentile (Median)
75th Percentile
90th Percentile
99th Percentile
Max Latency
Latency (s)
0
20
40
60
80
100
Not that great here
R10
R100
R250
R1000
Request Rate (R)


Latency Trends for 1000 Metadata Fields
50th Percentile
75th Percentile
90th Percentile
99th Percentile
Avg Latency
Latency (s)
175
150
125
100
75
50
25
0
R10
R100
R250
R1000
Request Rate (R)

Ingvord on Mar 23 Member Author ···

Jo, I pinned down the suspect - it is `convert-physical-quantities-interceptor`. I this in the flame grpah:



As well as in APM:

tetur adipiscing elit.

Thanks

Directed by
Igor Khokhriakov aka Ingvord