Ingesting over a million rows per second on a single instance

Time-series processing using QuestDB

Javier Ramirez, Developer Advocate at QuestDB
@supercoco9
An open source time-series database for fast ingest and SQL queries

questdb.io

Releases

6.3
current branch
2 weeks ago

Contributors

169 contributors
+ 92 contributors

Languages

- Java 95.0%
- C++ 2.7%
- C 2.7%
- Assembly 1.7%
- CMake 0.1%
- Shell 0.1%
We would like to be known for:

- Performance
  - Better performance with smaller machines
- Developer Experience
- Proudly Open Source (Apache 2.0)
Not all big (or fast) data problems are the same
Do you have a time-series problem? (1/2)

- Most of your queries are scoped to a time range
- You mostly insert data. You rarely update or delete individual rows
- It is likely you write data more frequently than you read data
- Since data keeps growing, you will very likely end up with much bigger data than your typical operational database would be happy with
- You often need to resample your data for aggregations/analytics
- You often need to align timestamps from multiple data series
Do you have a time-series problem? (2/2)

- You typically access recent/fresh data rather than older data
- But still want to keep older data around for occasional analytics
- Your data origin might experience bursts or lag, but keeping the correct order of events is critical for you
- But you typically request your reads to show data captured recently
- Both ingestion and querying speed are critical for your business
Some time-series demo queries

https://demo.questdb.io/
Ingesting over 1 million time series per second on a single instance
I am dead inside.
All benchmarks are lies (but they give us a ballpark)

Ingesting over 1.4 million rows per second (using 5 CPU threads)
https://questdb.io/blog/2021/05/10/questdb-release-6-0-tsbs-benchmark/

While running queries scanning over 4 billion rows per second (16 CPU threads)
https://questdb.io/blog/2022/05/26/query-benchmark-questdb-versus-clickhouse-timescale/
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>.github/workflows</td>
<td>Create go.yml for GitHub workflow</td>
<td>Last year</td>
</tr>
<tr>
<td>cmd</td>
<td>Questdb benchmark support</td>
<td>Last year</td>
</tr>
<tr>
<td>docs</td>
<td>Questdb benchmark support</td>
<td>Last year</td>
</tr>
<tr>
<td>helm</td>
<td>TSBS Docker and helm chart</td>
<td>2 years ago</td>
</tr>
<tr>
<td>internal</td>
<td>Questdb benchmark support</td>
<td>Last year</td>
</tr>
<tr>
<td>load</td>
<td>Enable persisting ingestion/query benchmark results in a common fo...</td>
<td>Last year</td>
</tr>
<tr>
<td>pkg</td>
<td>Questdb benchmark support</td>
<td>Last year</td>
</tr>
<tr>
<td>scripts</td>
<td>Questdb benchmark support</td>
<td>Last year</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Questdb benchmark support</td>
<td>Last year</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>Add multinode to master</td>
<td>Last year</td>
</tr>
<tr>
<td>Dockerfile</td>
<td>TSBS Docker and helm chart</td>
<td>2 years ago</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Update copyright year to 2021</td>
<td>2 years ago</td>
</tr>
</tbody>
</table>

**About**

Time Series Benchmark Suite, a tool for comparing and evaluating databases for time series data.

- benchmarking
- cassandra
- mongodb
- influxdb
- time-series
- timescaledb

**Readme**

MIT license

- 1k stars
- 46 watching
- 233 forks

**Releases**

No releases published

**Packages**

No packages published
Technical decisions and trade offs we made to get here
We can make many assumptions about the shape of the data and usage patterns
Written FROM SCRATCH for performant time-series

- Using JAVA unsafe mode, with zero GC and sharing memory with C++
- Writing our own IO functions, with native memory networking and zero GC
- Own implementation of String and other common classes, to avoid overhead
- Own implementation of Logger, for speed and to avoid interpolations
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mode</th>
<th>Cnt</th>
<th>Score</th>
<th>Error</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogBenchmark.testLogOneIntBlocking</td>
<td>avgt</td>
<td>2</td>
<td>265.391</td>
<td></td>
<td>ns/op</td>
</tr>
<tr>
<td>LogBenchmark.testLogOneInt</td>
<td>avgt</td>
<td>2</td>
<td>82.985</td>
<td></td>
<td>ns/op</td>
</tr>
<tr>
<td>LogBenchmark.testLogOneIntDisabled</td>
<td>avgt</td>
<td>2</td>
<td>0.661</td>
<td></td>
<td>ns/op</td>
</tr>
<tr>
<td>Log4jBenchmark.testLogOneInt</td>
<td>avgt</td>
<td>2</td>
<td>877.266</td>
<td></td>
<td>ns/op</td>
</tr>
<tr>
<td>Log4jBenchmark.testLogOneIntDisabled</td>
<td>avgt</td>
<td>2</td>
<td>1.368</td>
<td></td>
<td>ns/op</td>
</tr>
</tbody>
</table>

Down to the nanosecond
QUESTDB'S APPROACH TO PERFORMANCE

I WILL FIND YOU...
AND I WILL QUICKEN YOU
Quick recap

- Time-series problems can be hard
- QuestDB only does time-series
- Ingestion is done via official clients (or ILP over socket), queries are done via SQL
- QuestDB’s storage model makes ingestion very fast, and indices unnecessary
- We measure-implement-repeat continuously to improve performance
- All benchmark are lies, but if you like them take a look at

https://questdb.io/blog/tags/engineering/
THANKS!

For more info, [https://questdb.io](https://questdb.io) and [https://demo.questdb.io](https://demo.questdb.io)

We 💖 contributions and ⭐ stars [github.com/questdb/questdb](https://github.com/questdb/questdb)

Javier Ramirez, Developer Advocate at QuestDB
@supercoco9