



Lars Quentin

## Evaluation of Time-Series Databases

Elasticsearch and InfluxDB

## Table of Contents

- **1** Getting up to Speed: Introduction
- 2 The Need for Speed: From Lucene to Grafana
- 3 Looking in the Rear-View-Mirror: Related Work
- 4 Under the Hood: Our Methodology
- 5 The Podium: Results and Conclusion

## Why do we care about Time Series Metrics Data?

- Usage Overview
- Find Bottlenecks
- Help with Workload Balancing
- Demand Analysis and Forecasting
- Optimize Energy Efficiency

But why do we care about performance?





Introd	uction
000	

Overview

Related Work

## But actually...



Figure: Emmy's 1422 nodes located in Göttingen

## The Need for Speed: From Lucene to Grafana

#### We Evaluate Two Time-Series Databases:

- ► Elasticsearch, a distributed search engine.
- InfluxDB, a time-series database.

In order to understand why, one has to look at their shared history.

Introduction	Overview ⊙●○○○○	Related Work	Methodology 000000	<b>Results</b> ○
		-		

#### Lucene

- Java-based Search Engine Library
- Developed in 1999 for Apache Nutch
- Fuzzy Full-Text Search

#### Elasticsearch

- Distributed Search Engine
- Based on Lucene
- Developed in 2010
- Used at Wikipedia, Netflix, Stackoverflow, LinkedIn

## **TUCENE**

Figure: Lucene Logo



Figure: Elasticsearch Logo

## Money, Money, Money

- Elasticsearch rose to popularity amongst the DevOps community.
- Thus, it grew beyond the scope of a hobby project and needed funding.
- And a database alone is not enough for business applications.
- Thus, the ELK stack was born.

Results

 Introduction
 Overview
 Related Work
 Methodology

 000
 000
 000000
 000000

ELK(B) stack

# beats logstash elasticsearch kibana

Data Collection Data Aggregation & Processing Indexing & storage

Analysis & visualization

Figure: ELK Stack

Results

Introduction

**Overview** 

Related Work

## Grafana

- System Monitoring Solution
- Forked from Kibana
- Specialized for time-series data
- Supports multiple data sources
  - No Elasticsearch vendor lock-in
  - Allows for more specialized database technologies



## InfluxDB

- Time-Series Database
- Built for technology applications
- Highly specialized for time-series data
- Also used at the GWDG as a data source for Grafana



Figure: InfluxDB Logo

## Looking in the Rear-View-Mirror: Related Work

- Only a single exhaustive performance comparison of Elasticsearch and InfluxDB.
- Conducted by InfluxData, the company behind InfluxDB.
- Publically available on GitHub.
- In this section, we will deep dive into their methodology and findings.

μ	master +	Go to file Add file *	<> Code +	About
8	alespour fte: MongoDi	3 schema and lot use-case (#2 ini on Nov 2	8, 2022 🕤 947	Code for comparison write ups InfluxDB and other solutions
h	bulk_data_gen	fix: MongoDB schema and iot use-case (#2	3 months ago	Readme
h	bulk_load	Move old output to stderr log and set new $\cdot_{\rm ex}$	2 years ago	☆ 288 stars
	bulk_query	Enable option for json output for query be	2 years ago	60 watching
	bulk_query_gen	fix: MongoDB schema and iot use-case (#2	3 months ago	Y 102 forks
8	cmd	fix: MongoDB schema and iot use-case (#2	3 months ago	
h	mongo_serialization	point may contain multiple fields just like w	5 years ago	Releases
h	timescale_serializait	Binary serialization for Timescale using go	5 years ago	No releases published
b.	util/report	Added test for storing result report in influ	4 years ago	
h	void_server	void server returns "no content" header	7 years ago	Packages
C	DEMO.txt	Rename ES aggregation template switch a	7 years ago	No packages published
D	LICENSE	Initial commit	7 years ago	
C	README.md	Update README.md	2 years ago	Used by 4
D	bonitoo.toml	Add files via upload	4 years ago	Brobsk / high_cardinality
C	go.mod	feat: add MongoDB 5.x timeseries support	3 months ago	@chrono / high_cardinality
D	go.sum	feat: add MongoDB 5.x timeseries support	3 months ago	@m3dbx / promremotebench
C	mongo.flatbuffers.fbs	point may contain multiple fields just like w	5 years ago	
D	timescale.proto	Binary serialization for Timescale using go	5 years ago	Contributors 20
	README md			17 🗢 🕘 🚥 🙈 🥸

## Overview

#### Measured across 3 vectors

- 1 Data ingest performance
- 2 On-disk storage requirements
- 3 Mean query response time

#### Split into 5 disconnected steps

- 1 Data Generation
- 2 Data Loading
- 3 Query Generation
- 4 Query Execution
- 5 Query Validation

#### 1. Data Generation

- Random and Deterministic (pinned PRNG seed)
- Shared generation logic
- Generated beforehand
- Modelled realistically
  - DevOps related metrics, same structure as telegraf
    - cpu, diskio, kernel, mem, redis...
  - clamped random walk
    - important for optimizations such as delta compression

#### 2. Data Loading

#### KISS

- Batched into bulk queries (default 5000 documents)
- parallelized (default 5 workers)
- sent as fast as possible

#### 2. Data Loading

#### KISS

Batched into bulk queries (default 5000 documents)

- parallelized (default 5 workers)
- sent as fast as possible

#### 3. Query Generation

- Random and Deterministic (pinned PRNG seed)
- Shared generation logic
- Generated beforehand

#### 4. Query Execution

#### KISS

Sends parallelized range queries

#### 4. Query Execution

KISS

Sends parallelized range queries

#### 5. Query Validation

Done via manual verification

Ensuring that both aggregation results are approximately the same

#### According to the White paper

- InfluxDB outperformed Elasticsearch by **3.8x** when it came to data ingestion
- InfluxDB outperformed Elasticsearch by up to 7.7x when measuring query performance
- InfluxDB outperformed Elasticsearch by delivering **9x** better compression

#### According to the White paper

- InfluxDB outperformed Elasticsearch by 3.8x when it came to data ingestion
- InfluxDB outperformed Elasticsearch by up to 7.7x when measuring query performance
- InfluxDB outperformed Elasticsearch by delivering **9x** better compression

#### Problems

- Bad incentive structure
- Done with Influx version 1
- Not oriented for HPC workloads and topologies
- Data was ingested in bulk

- Extending InfluxData paper's methodology
- Everything not mentioned is the same.
- Adapted to our use case
  - This is a huge feat; Emmy is big
  - ▶ We run the recommended production configuration
  - We mainly focus on write, not query read
    - Since this is the bulk of the work
- Split into distinct phases as well.
  - KISS! KISS! KISS!

Results

## 1. Data Generation

- Also random and deterministic, pinned PRNG seeds
- Only generating hardware / kernel measurements, no application metrics
- We use clamped 1D perlin noise
- One file per ingest worker!
  - Less error prone!
  - KISS!

## 2. Data Ingestion

- We don't use bulk ingestion
  - instead, data of one node per request
- Sending as fast as possible
- Flushing at the end
- Faster is better

## 3. Check Index Compression

- We do not trust their analytics
- Multi-Step process
  - 1 Get size of data directory
  - 2 Fill the data
  - 3 Flush and Compress
    - InfluxDB: Tree Compaction
    - Elasticsearch: Force Merge API
  - 4 After that, we measure again

Smaller ∆ is better

## 4. Design Queries

#### Methodology

- 1 Get a real world Grafana dashboard
- 2 Extract the queries through the networking tab
- Port them to the Query Languages
- 4 Make them parameterized



Figure: Extracting through Networking Tab

## 5. Benchmark Queries

- We test querying while ingesting data
- Linear step increment of index size
  - correllate the response time
- Faster is better

Introduction	Overview	Related Work	Methodology	Results
000	000000	000000	000000	•

The Podium: Results and Conclusion

## Stay tuned ;-)