Athens Institute for Education and Research
ATINER

ATINER's Conference Paper Series
COM2016-1955

# Storing Sensor Data in Different Database Architectures

**Till Haenisch**
**Professor**
**DHBW Heidenheim**
**Germany**

**Manfred Roessle**
**Professor**
**University of Applied Sciences Aalen**
**Germany**

**Rene Kuebler**
**Research Fellow**
**University of Applied Sciences Aalen**
**Germany**

ATINER CONFERENCE PAPER SERIES No: COM2016-1955

# An Introduction to

# ATINER's Conference Paper Series

ATINER started to publish this conference papers series in 2012. It includes only the papers submitted for publication after they were presented at one of the conferences organized by our Institute every year. This paper has been peer reviewed by at least two academic members of ATINER.

Dr. Gregory T. Papanikos
President
Athens Institute for Education and Research

This paper should be cited as follows:

**Haenisch, T., Roessle, M. and Kuebler, R.** (2016). **"Storing Sensor Data in Different Database Architectures",** Athens: ATINER'S Conference Paper Series, No: **COM2016-1955.**

Athens Institute for Education and Research
8 Valaoritou Street, Kolonaki, 10671 Athens, Greece
Tel: + 30 210 3634210 Fax: + 30 210 3634209 Email: info@atiner.gr URL: www.atiner.gr
URL Conference Papers Series: www.atiner.gr/papers.htm

ISSN: 2241-2891
31/08/2016

# Storing Sensor Data in Different Database Architectures

**Till Haenisch**

**Manfred Roessle**

**Rene Kuebler**

## Abstract

The Internet of Things comes along with a huge number of “things” producing data. All collected mass data must be stored in real time for actual data processing and future analysis. Almost every database on the market has problems handling time series data. In the past that was only a problem for a small group of users, but today and in the future that will change: one important manifestation of the Internet of Things are sensor networks, possibly large numbers of sensors generating data in more or less fixed time intervals. Internet of Things applications have to handle large amounts of time series data efficiently. There are many different approaches for storing this kind of data like relational databases, NoSQL databases, in-memory systems files and so on. This paper benchmarks typical software platforms used in the Internet of Things scenarios, especially regarding their ingestion rates, and reveals interesting results.

**Keywords:** Internet of Things, Database systems, Performance evaluation, Wireless sensor networks.

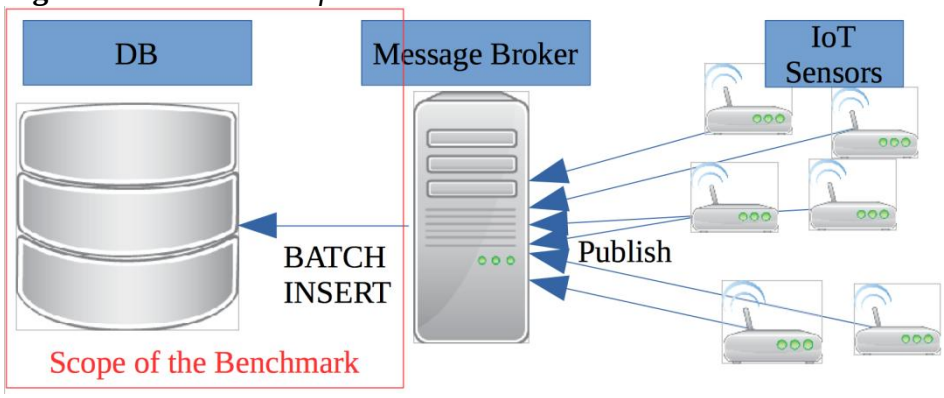
## Introduction

Almost every database on the market has problems handling time series data, see for example [5]. In the past that was only a problem for a small group of users, but today and in the future that will change: one important manifestation of the Internet of Things (IoT) are sensor networks, possibly large numbers of sensors generating data in more or less fixed time intervals. Consider for example buildings like those of a university with about 300 rooms. If theses rooms are equipped with sensors for temperature, humidity, light intensity, sound level and motion there will be at least approximately 2000 sensors in the building (sensors in the rooms plus a few in the corridors etc.). If each sensor produces one measurement per second this equals to 120,000 measurements per minute. That means 7.2 million measurements per hour or 173 million per day and some 63 billion measurements per year. Assuming 60 bytes per measurement this equals to roughly 4 Terabyte of data per year just for one building.

Another example is our DataCast project. We are working on a realtime quality prediction system for die-cast aluminium. There are 32 sensors for temperature, cavity pressure, die fill control etc. installed. The sampling rate is 5 kHz, which means every sensor provides 5000 measurements per second. We use 80 Byte per tuple, 8 Byte each for the partnumber and timestamp and 2 Byte per sensor-value. That is 2 MB of data per second. The process time for one shot is about 30 seconds over all. The duration of the measurement phase within the casting time is about 10 seconds, which means 4 MB per part. Producing 8 hours per day we have to deal with 4 GB data per day and casting machine. For a year we have to store about 1 TB for each casting machine.

The typical architecture of those IoT-sensor networks is shown in the Figure 1 below.

**Figure 1.** *Architecture of an IoT Sensor Network*




The sensors or the sensor networks are connected to a message broker. The message broker consists of two different layers. One layer collects the data published by the sensors. Because not every sensor is working with the same sampling rate it is necessary to provide a kind of “staging layer” to cache the sensor data while being prepared to be written onto the database. The staging

layer is also used to build batches of records for specialized “insert many”-database commands. These commands will strongly influence the writing performance of the database systems.

For the ongoing research we neglect the obvious influence of the message broker and focus instead on the question how to store that large amount of data efficiently on inexpensive hardware while allowing flexible queries for reporting on different time scales.

Traditionally the queries are predefined, especially the time domains (e.g. mean temperature per hour) so the data can easily be compressed by aggregations like storing only statistical parameters of the measured values like means, standard deviations and extreme values for different time scales like minutes, hours, days and months. But the idea with big data is to store all data without predetermining the possible analysis strategies. Because of this it is necessary to store all measurements for future unknown requirements.

At the moment there is much ado about NoSQL and InMemory databases and proponents of these systems, especially vendors, postulate that conventional relational databases are useless in these scenarios. This paper tries to underpin these discussions with some calculations and experimental results. A typical IoT scenario like the ones mentioned above is simulated and data is stored in various systems, e.g. MySQL, MongoDB, operating system files, VoltDB, Apache Kafka and SAP HANA.

The write performance for the scenario is compared and analyzed.

## Method

To compare the relative write performance ("*ingestion rate*") of the different systems, a simple load model was implemented and tested with current versions of the database platforms. The simulated load consists of records with three elements, a timestamp, a sensor id and the actual value, if necessary a unique id was added. This resulted in a size per record of approximately 32 bytes. A large number of these records were inserted by {1, 2, 4, 8} processes. This corresponds to an architecture where the sensor values are collected and distributed via a middleware, e.g. a message broker like ZeroMQ, and are written to the database by a single consumer.

In the scenarios outlined above, we had data rates between a few thousand records per minute and some 100.000 records per second. Therefore we measured the time to write 1.000.000 records to the database to get a rough estimate of the performance limits.

We used a representative sample of the current market with one system of each relevant architecture: a standard SQL database (MySQL), a widely used NoSQL database (MongoDB), an in-memory row-store (VoltDB) and an in-memory column-store (SAP HANA). In addition to these more

or less conventional systems we used a persistent messaging system which can be used for event processing (Apache Kafka) and plain operating system files as a baseline.

All systems were used out of the box without special tuning. Of course it should be possible to get better results by fine tuning of the systems, but our experience database systems are often used without special tuning efforts in practice. Besides that, we only want to get a rough estimate of the performance that can be expected.

We tested the batch sizes of {1, 10, 50, 100, 500, 1000, 10000} with all systems that were capable of using batch inserts, to see a possible potential for optimization.

To have a common base, we used Java (1.8) as the programming language of all tests. For all systems we utilized the official Java drivers, that where available, in addition JDBC. The full source code used in this benchmark is available on GitHub https://github.com/TillHaenisch/TimeseriesBench.

The software versions used were HANA SPS-10, VoltDB 5.6 community edition, Apache Kafka 2.11, MongoDB 3.2 and MySQL 5.6.

All database servers have been installed on the Amazon Web Services EC2 r3.2 xlarge with 60 GB Ram and 8 CPU cores and SSD ebs volumes. The standard operating system was Ubuntu Server 14.04 LTS (HVM), except SAP HANA. SAP HANA was automatically configured by SAP Cloud Appliance Library on SUSE Enterprise Linux. Transparent HugePages have been disabled for all database servers.

The benchmark program itself was run on an AWS EC2 c4.2 xlarge instance. The benchmark server and the test servers were configured for the same aws availability zone and placement group.

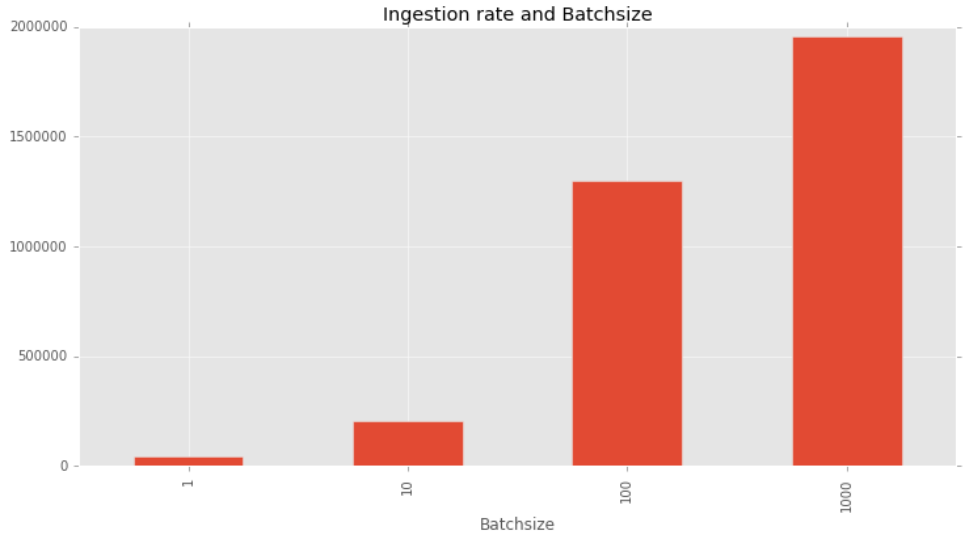
The measurements were repeated a few times to ensure that the result is not a single outlier, but we did not perform statistical analysis of the results, because it is not our intent to give precise benchmark results. The idea of this experiment is to provide some facts about the order of magnitude of realistic performance of the different architectures.

## Results

Table 1 shows the measured throughput in 1000 records/second, so for example SAP HANA was able to write some 500.000 records per second. With all systems we tried larger numbers of records up to 10 millions to see, if there is a performance penalty for small data sizes or if there are caching effects. SAP HANA showed no difference for these larger sizes, the throughput of VoltDB increased by roughly a factor of two up to some 350.000 records per second with increasing numbers of records. The other systems showed no large differences. For MySQL, SAP HANA, Kafka and MongoDB it is possible to control the size of batches, which means the number of records which are inserted with one command. The MongoDB tests showed that a batch size bigger than 100 did not result in much better and sometimes even worse insert rates. Kafka scaled best with big batch sizes of 10000 and more. The throughput with MySQL was also increased heavily when larger batches were

used: VoltDB was the only system that could handle single inserts with high insert rates. Figure 2 below shows the Ingestion rate versus the batchsize.

**Figure 2.** *MySQL: Ingestion Rate versus Batchsize*




Kafka is faster than the database systems and writing to an operating system file is much faster than any other platform.

**Table 1.** *Performance of the Different Systems for Ingesting 100.000 Records in 10.000 Batches (when possible) Best Value of Threads*

| | Throughput in 1000 records/sec |
|---|---|
| SAP HANA SPS10 | 506 |
| Volt DB 6.2 CE | 345 (single inserts) |
| Mongo DB 3.2 | 156 |
| MySQL 5.6 | 743 |
| Kafka 2.11 | 1522 |
| file | 6107 |

## Discussion

Our experiment shows, that a write performance of some 100.000 records per second is achievable on commodity hardware with all of the technologies. There are no differences on the order of magnitude between the database systems. Some databases perform better with the type and grouping of records

we used here, some may behave better in somewhat different scenarios. Our results conform to published results like Kafka being able to process two million writes per second on three machines [1], MongoDB writing some 6.000 1kbyte records (10 times the size of our records) per second [2], MySQL handling 150.000 records/second on one node [3] or VoltDB handling some 260.000 records per second [4].

The important point is, that linear write performance is not a critical factor to consider when choosing a technology for this kind of application, except, when there are massive data rates. In that case probably neither of the database systems evaluated are suitable. Using an event processing system like Apache Kafka is faster than every database and writing to an operating system file is about 25 times faster than Kafka.

## Conclusions

The ingestion rate of the tested Database Management Systems was high enough for moderate data rates. Batch inserts were necessary to achieve higher ingestion rates. A two tiered System with a Message Broker to collect the Sensordata and to write Batches into the DBMS is able to handle massive Data rates.

If massive ingestion rates are needed an OS file is a very performant alternative if the evaluation is not required in realtime.

In this paper we evaluated the write-performance of different Internet of Things scenarios. Benchmarking the read-performance is much more difficult. The reason for this is, that the workload for writing sensor data is easy to predict: In most cases it is either a continuous stream of messages or messages are aggregated and written in bulk. Both has been measured in our research.

The read-performance depends heavily on the access pattern of the actual application. But there are many possible applications like for example big data algorithms which tend to read the whole data set over a large amount of data maybe in many iterations, or jobs, where data is sampled and a more random-access like pattern is relevant.

In some applications the number of concurrent readers and writers could differ substantially. In some applications there might be only one writer and one reader while in different applications there might be hundreds of thousands of writers and many readers. Especially handling concurrency becomes very difficult here. This is the reason why we didn't discuss read performance, this is considered an important topic for our future research.

## References

[1] J. Kreps, “Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines), http://bit.ly/2bBp2PI.

[2] L. H. Jeong. Nosql benchmarking. http://bit.ly/2bBxfXm.

[3] A. Kumar, “Ingesting over 1,000,000 rows a second into MySQL in AWS Cloud,” http://bit.ly/2bFMY58.
[4] Diedrich, H., “877,000 TPS with Erlang and VoltDB,” http://bit.ly/2ce3Sfi.
[5] Stonebraker, M.; Brown, P.; Donghui Zhang; Becla, J., “SciDB: A Database Management System for Applications with Complex Analytics,” J. Name Stand. Abbrev., Computing in Science & Engineering, vol.15, no.3, pp.54,62, May-June 2013.