Data Historians in the Data Management Landscape

Brice Chardin, Jean-Marc Lacombe, Jean-Marc Petit

The original publication is available at www.springerlink.com:
https://doi.org/10.1007/978-3-642-36727-4_9

Data Historians in the Data Management Landscape

Brice Chardin[1,2], Jean-Marc Lacombe[1], and Jean-Marc Petit[2]

[1] EDF R&D, France
[2] Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France

Abstract. At EDF, a leading energy company, process data produced in power stations are archived both to comply with legal archiving requirements and to perform various analysis applications. Such data consist of timestamped measurements, retrieved for the most part from process data acquisition systems. After archival, past and current values are used for various applications, including device monitoring, maintenance assistance, decision support, statistics publication, etc.
Large amounts of data are generated in these power stations, and aggregated in soft real-time – without operational deadlines – at the plant level by local servers. For this long-term data archiving, EDF relies on data historians – like InfoPlus.21, PI or Wonderware Historian – for years. This is also true for other energy companies worldwide and, in general, industry based on automated processes.
In this paper, we aim at answering a simple, yet not so easy, question: how can data historians be placed in the data management landscape, from classical RDBMSs to NoSQL systems? To answer this question, we first give an overview of data historians, then discuss benchmarking these particular systems. Although many benchmarks are defined for conventional database management systems, none of them are appropriate for data historians. To establish a first objective basis for comparison, we therefore propose a simple benchmark inspired by EDF use cases, and give experimental results for data historians and DBMSs.

1 Introduction

In industrial automation, data generated by automatons – sensors and actuators – are generally used with critical real-time constraints to operate the plant. Beside this operational usage, these data streams – mainly measurements from sensors – may be mined to extract useful information for failures anticipation, plant optimization, etc.

At EDF, a worldwide leading energy company, process data produced in power stations are indeed archived for various analysis applications and to comply with legal archiving requirements. These data consist of timestamped measurements, along with meta-data on data quality, retrieved for the most part from process data acquisition systems.

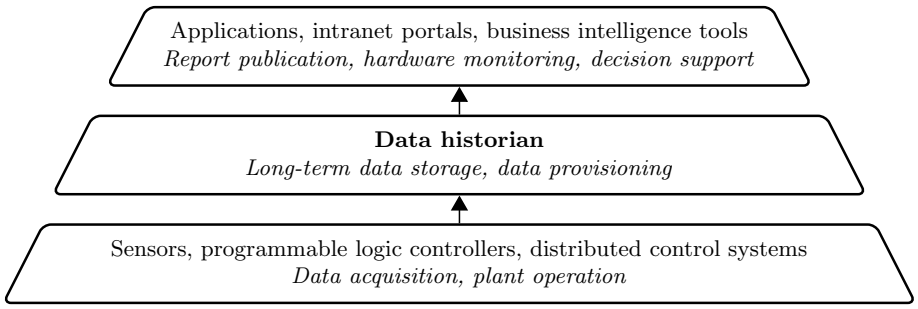

Fig. 1. Data historians in the production information system

These archived data – past, but also current values – are used for various applications, including devices monitoring, maintenance assistance, decision support, statistics publication, compliance with environmental regulation, etc. Data mining may also be performed, essentially with signal processing techniques: cross-correlation, filtering, dimension reduction, spectrum analysis, prediction, etc.

Power stations generate large amounts of data for thousands of measurement time series, with sampling intervals ranging from 40ms to a few seconds. This data is aggregated in soft real-time – without operational deadlines – at the plant level by local servers. For this long-term data archiving, EDF relies on data historians for years. Figure 1 gives an overview of power plants information systems at EDF, with data historians as fundamental intermediaries to access production data.

In this paper, we aim at answering a simple, yet not so easy, question: how can data historians be placed in the data management landscape, from classical relational database management systems (RDBMS) to NoSQL systems? From a practical point of view at EDF, answering such a question may have a profound impact on the choice of its data management systems. To answer this question, we first give an overview of data historians and analyze the similarities with three types of systems: RDBMS, data stream management systems (DSMS) and NoSQL systems. We then discuss benchmarking in this context. Although many benchmarks are defined for conventional database management systems, none of them are appropriate for data historians. To establish a first objective basis for comparison, we therefore propose a simple benchmark inspired by EDF use cases, and give experimental results for a data historian (InfoPlus.21), a RDBMS (MySQL) and a NoSQL system (Berkeley DB) – DSMS are not relevant for this benchmark (i.e. no continuous queries).

The purpose of this paper is not to define a new benchmark such as TPC benchmarks, but to introduce a new application lacking adapted comparison tools. All the more so data historians are proprietary systems whose performances are not documented. To the best of our knowledge, we are not aware of similar work.

Paper organization An overview of data historian technologies is given in section 2. In section 3, an analysis of the differences between data historians and other data management systems is proposed. In section 4, we focus on performance comparison and define a benchmark to evaluate differences between these technologies. Results for this benchmark with a data historian, a RDBMS and a NoSQL DBMS are presented in section 5. Section 6 concludes and draws perspectives on this ongoing work.

2 Overview of data historians

In Supervisory Control And Data Acquisition (SCADA) systems, data acquisition begins with Programmable Logic Controllers (PLC) or Remote Terminal Units (RTU) which retrieve measurements from metering devices and equipment status reports. These data elements – called tags or points – represent a single input or output value monitored or controlled by the system. Tags usually appear as value-timestamp pairs.

After generation, data are eventually sent to other automatons, or monitoring servers to let human operators make supervisory decisions. Coincidentally, data may also be fed to a data historian to allow trending and other analytical auditing.

Data historians – like InfoPlus.21 [3] by AspenTech, PI [8] by OSIsoft or Wonderware Historian [5] by Invensys – are proprietary software designed to archive and query industrial automation time series data. They store time series following a hierarchical data model which reflect the operating environment. This data model should be consistent with the plant organization to ease browsing and group similar time series by subsystem.

Data historians receive data generated, for the most part, by industrial process control – Distributed Control Systems (DCS) or SCADA systems. For these purposes, they provide some business-oriented features which are not typically found within other data management systems: they support industrial communication protocols and interfaces – like OPC [7], Modbus or device manufacturers proprietary protocols – to acquire data and communicate with other DCS or SCADA software. They also receive data from other systems, occasionally provided by external entities, like production requirements or pricing informations from the Transmission System Operator, as well as meteorological forecasts. Additionally, manual insertions may occur to store measurements made by human operators.

Data historians provide fast insertion rates, with capacities reaching tens of thousand of tags processed per second. These performances are allowed by specific buffer designs, which keep recent values in volatile memory, to later write data on disk sorted by increasing timestamps. Acquired data that do not fall in the correct time window are written on reserved areas, with reduced performances, or even discarded.

To store large amounts of data with minimum disk usage and acceptable approximation errors, data historians often rely on efficient data compression

engines, lossy or lossless. Each tag is then associated with rules conditioning new values archival – for example: storage at each modification, with a sampling interval, or with constant or linear approximation deviation thresholds.

Regarding information retrieval, data historians are fundamental intermediary in the technical information systems, providing data for plant operating applications – like device monitoring or system maintenance – and business intelligence – like decision support, statistics publication or economic monitoring. These applications might benefit from data historians time series specific features, especially interpolation and re-sampling, or retrieve values pre-computed from raw data. Values not measured directly, auxiliary power consumption or fuel cost for example, key performance indicators, diagnostics or informations on availability may be computed and archived by data historians.

Visualization features are dispensed by standard clients supplied with data historians. They ease exploitation of archived data by displaying plots, tables, statistics or other synoptics. These clients allow efficient time series trending by retrieving only representative inflection points for the considered time range.

Data historians also provide a SQL interface, with proprietary extensions for their specific features, and offer some continuous queries capabilities, to trigger alarms for instance.

Roughly speaking, data historians can be characterized by:

- a simple schema structure, based on tags,
- a SQL interface,
- a NoSQL interface for insertions, but also to retrieve data from time series, eventually with filtering, resampling or aggregate calculations,
- a design for high volume append-only data,
- built-in specialized applications for industrial data,
- no support for transactions,
- a centralized architecture.

3 Data historians and other data management systems

3.1 Data historians and RDBMS

The hierarchical data model might be convenient to represent data according to the plant organization, but the relational model might be preferred to easily integrate other data. Moreover, data historians mostly acquire time series: other data may not be supported; they can hardly be used for relational databases. Besides, even if data historians support SQL queries, they might have limitations with their query optimizers and their compliance with the entire SQL standard. For these reasons, some data historians can be associated with a RDBMS to store relational data.

Additionally, data historians do not support transactions and might not guarantee data durability for most recent measurements, even if they often provide several levels of buffers across the network to prevent data loss during server or network failures.

3.2 Data historians and NoSQL systems

Data historians provide a dedicated non-SQL interface for insertion and retrieval. Insertions are functionally comparable to SQL insert statements, with improved performances as these routines avoid parsing and type conversions. The retrieval interface however differ significantly from SQL. Extraction queries can be defined with filtering conditions (typically using value thresholds or status verification), resampling intervals and aggregate calculations over time periods. While filtering conditions are straightforward to translate in SQL, aggregate calculations grouped by time periods (e.g. timestamp÷period) might not be handled efficiently by query optimizers. Interpolated values (with various interpolation algorithms) can be tedious to define, both in SQL and with usual NoSQL interfaces, especially when combining multiple time series with different sampling periods.

Nevertheless, ordered key-value data stores provide closely related NoSQL access methods, like Berkeley DB cursor operations [6]. These cursors can be set to a specified key value, and incremented by key order – to retrieve consecutive values of a time series in this context. However, data historian interface is specialized, and thus combine several usual algorithms and processing techniques besides raw data retrieval.

NoSQL systems can typically be distributed over multiple servers. For data historians, this horizontal scalability is separated between replication and distribution. Load balancing for data retrieval is provided by replication, where multiple servers hold the same data and are individually able to serve extraction queries. However, this architecture does not decrease insertion workloads: data distribution is achieved declaratively, by associating a tag with a specific server – which might then be replicated. Therefore, data historians provide only limited load-balancing and horizontal scalability in comparison with most NoSQL systems. However, data retrieval relies on an efficient NoSQL interface for range queries. Typically, key-value stores using distributed hash tables are not suitable, which makes scalability a complex issue.

3.3 Data historians and DSMS

Data stream management systems provide continuous queries capabilities as an extension of SQL [1] or appear as an extension on top of a classical RDBMS such as Oracle. Such systems typically process data over a relatively short time-window to execute continuous queries.

As far as insertions are concerned, data historians have similar mechanisms as they associate their write buffer with a time window, rejecting or inserting with lower performances data falling out of range. However, in our context, continuous queries are handled by specific monitoring and process control systems, with real-time constraints due to their critical aspect; while long-term data archiving is provided by data historians.

Yet, a new generation of DSMS allows long-term analysis of historical data by warehousing data streams. These stream warehouse systems still focus on

continuous queries, which is not the purpose of data historians. As for data transfer and archiving, "a stream warehouse ... receives a wide range of data feeds from disparate, far-flung, and uncontrolled sources" [4], which is not true in the context of industrial automation.

3.4 Synthesis

Data historians are products designed and sold for a specific industrial use. Other data management systems might have a wider range of applications, at a possibly lower cost, but do not include most of the business-oriented features included in data historians. These systems typically can neither acquire data from process control systems with industrial communication protocols, nor use lossy compression, interpolation or re-sampling on time series.

To sum up, the match between data historians and other data management systems is clearly imperfect:

- no data distribution,
- no transactions,
- only raw sensor data (no images, no blobs, etc.).

Despite these differences, using a RDBMS or a NoSQL system for industrial data seems feasible with some functional restrictions, even if not yet adopted by the market. As data historian manufacturers advertise high insertion speeds, we ought to investigate the capacity of other data management systems to sustain industrial automation workloads before considering them for production purposes.

Benchmarking these systems would help evaluating performance differences, otherwise unavailable. Still, this comparison turns out to be not so easy, the functionalities, the interfaces, the underlying data model being quite different.

As a matter of fact, we focus on simple data-centric operations (queries) over a generic database schema. To initiate this comparison, we propose a micro-benchmark and run it against a data historian, an ordered key-value store and a RDBMS, which have been optimized for this context.

4 Micro-benchmark

Although many benchmarks are defined for relational database management systems, like TPC-C or TPC-H [9, 10], to the best of our knowledge, none of them are designed for data historians. The idea of comparing these systems with an existing benchmark – designed for RDBMS – seems natural. However, in the context of industrial data at EDF, it seemed impractical to use one of the Transaction Processing Performance Council benchmarks for the following reasons:

- Data historians are not necessarily ACID-compliant, and generally do not support transactions.

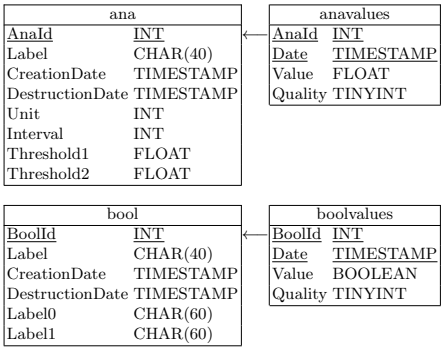

Fig. 2. Logical relational schema

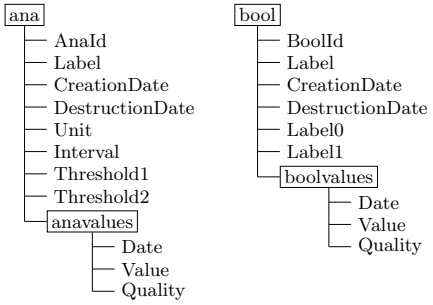

Fig. 3. Hierarchical schema for data historian

- Insertion is a fundamental operation for data historians. This type of query is executed in real-time, which prevent using benchmarks that batch insertions, like TPC-H.
- Data historians are designed to handle time series data. It is mandatory that the benchmark focuses on this type of data for results to be relevant.

Benchmarks for data stream management systems, like Linear Road [2] can also be considered; but data historians do not comprehensively handle continuous queries. Data historians – and RDBMS for that matter – use a different design by storing every data for future data mining operations. In DSMS benchmarks, even historical queries use a first level of aggregation on raw data, which is not representative of data historian utilizations at EDF.

To compare data historians and RDBMS performances, we defined a benchmark inspired by the scenario of nuclear power plants data historization. In this context, data generated by sensors distributed on the plant site are aggregated by a daemon communicating with the data historian. For insertions, the benchmark simulates this daemon and pseudo-randomly generate data to be inserted.

This data is then accessible for remote users, which can send queries to update, retrieve or analyze this data. After the insertion phase, this benchmark proposes a simple yet representative set of such queries.

4.1 Database schema

This benchmark deals with data according to a minimal database schema, centered upon times series data and simplified from EDF nuclear power plants schema. For each variable type – analog or boolean – a description table is defined (*ana* and *bool*). Measurements are stored in separate tables (*anavalues* and *boolvalues*). Figure 2 shows the logical relational schema for this benchmark.

Each time series is associated with an identifier (AnaId or BoolId), a short textual description – or name – (Label), a creation date (CreationDate) and a destruction date (DestructionDate). For analog values, the description table

ana also contains the unit of measurement (Unit), which is usually described in a separate table discarded for this benchmark, a theoretical sampling interval (Interval) and two thresholds indicating if the measured value is critically low (Threshold1) or critically high (Threshold2). For boolean values, the description table *bool* contains two short descriptions associated with values 0 (Label0) and 1 (Label1).

Times series are stored in tables *anavalues* and *boolvalues*, which contains the time series identifier (AnaId or BoolId), the timestamp with millisecond precision (Date), the value (Value) and a small array of eight bits for meta-data – data quality – (Quality).

For this benchmark to be compatible with hierarchical data models used by data historians, the relational model defined previously can not be mandatory. In figure 3, we propose an equivalent hierarchical schema, representing the same data and allowing functionally equivalent queries to be executed.

4.2 Query workload

By defining twelve queries, representative of EDF practices, this benchmark aims at giving an overview of data historians or RDBMS prevalence. Parameters generated at run time are in brackets. These parameters are exactly the same between each benchmark execution, to obtain identical data and queries. Queries are executed one by one in a fixed order; interactions are currently not evaluated with this benchmark to keep its definition simple and alleviate performances analysis. As some queries tend to have similar definitions, we do not express every SQL statement in this paper.

Insertion Data insertion is a fundamental operation for data historians. To optimize these queries, the interface and language are not imposed (ie. these queries can be translated from SQL to any language or API call, whichever maximizes performances).

Q0.1 Analog values insertions

```
INSERT INTO anavalues VALUES
  ([ID],[DATE],[VAL],[QUALITY])
```

Q0.2 Boolean values insertions

Updates Data updates, retrieval and analysis are usually performed by end-users; performance constraints are more flexible compared with insertions.

Q1.1 Update an analog value. The Quality attribute is updated to reflect a manual modification of the data.

```
UPDATE anavalues
SET Value = [VAL], Quality = (Quality | 128)
WHERE AnaId = [ID] AND Date = [DATE]
```

Q1.2 Update a boolean value. The Quality attribute is updated to reflect a manual modification of the data.

Data retrieval and analysis This benchmark defines nine such queries to evaluate the performances of each system, and identify specific optimizations for some types of queries. Queries without parameters (Q11.1 and Q11.2) are executed only once to refrain from using query caches – storing results in order not to re-evaluate the query. NoSQL equivalent queries should provide the same results. We provide two exemples, for Q2.1 and Q9, using a cursor-based interface, which can be positioned (`position`) and incremented (`readnext`).

Raw data extraction

Q2.1 Extract raw data for an analog time series between two Dates, sorted with increasing Date values.

```
SELECT * FROM anavalues
WHERE AnaId = [ID] AND Date BETWEEN [START] AND [END]
ORDER BY Date ASC
```

Algorithm 1: Q2.1 NoSQL query

input: id, start, end

```
1 position((id, start));
2 key, value ← readnext();
3 while key < (id, end) do
4 |  key, value ← readnext();
```

Q2.2 Extract raw data for a boolean time series between two Dates, sorted with increasing Date values.

Aggregate queries

Q3.1 Extract data quantity for an analog time series between two Dates.

```
SELECT count(*) FROM anavalues
WHERE AnaId = [ID]
  AND Date BETWEEN [START] AND [END]
```

Q3.2 Extract data quantity for a boolean time series between two Dates.

Q4 Extract the sum of an analog time series between two Dates.

Q5 Extract the average of an analog time series between two Dates.

Q6 Extract the minimum and maximum values of an analog time series between two Dates.

Filtering on value

Q7 Extract analog values above the threshold indicated in its description (ana.Threshold2).

```
SELECT Date, Value FROM ana, anavalues
WHERE ana.AnaId = anavalues.AnaId
```

```
  AND ana.AnaId = [ID]
  AND Date BETWEEN [START] AND [END]
  AND Value > ana.Threshold2
```

Q8 Extract analog values above a given threshold.

```
SELECT Date, Value FROM anavalues
WHERE AnaId = [ID]
  AND Date BETWEEN [START] AND [END]
  AND Value > [THRESHOLD]
```

Aggregate with value filtering on multiple time series

Q9 Identify the time series whose values most often do not fall between its high and low thresholds.

```
SELECT Label, count(*) as count FROM ana, anavalues
WHERE ana.AnaId = anavalues.AnaId
  AND Date BETWEEN [START] AND [END]
  AND (Value > Threshold2 OR Value < Threshold1)
GROUP BY ana.AnaId, Label ORDER BY count DESC LIMIT 1
```

Algorithm 2: Q9 NoSQL query

input: start, end

```
1  foreach id in ana.AnaId do
2      count[id] ← 0;
3      threshold1 ← ana[id].Threshold1;
4      threshold2 ← ana[id].Threshold2;
5      position((id, start));
6      key, value ← readnext();
7      while key < (id, end) do
8          if value.Value < threshold1 or value.Value > threshold2 then
9              count[id]++;
10         key, value ← readnext();
11 result_id ← i: ∀ id, count[id] ≤ count[i];
12 return(ana[result_id].Label, count[result_id]);
```

Sampling period verification on multiple time series

Q10 Identify the time series whose sampling period do not, by the greatest margin, comply with its description

```
SELECT values.AnaId, count(*) as count FROM ana,
(
  SELECT D1.AnaId, D1.Date,
         min(D2.Date-D1.Date) as Interval
```

```
  FROM anavalues D1, anavalues D2
  WHERE D2.Date > D1.Date
    AND D1.AnaId = D2.AnaId
    AND D1.Date BETWEEN [START] AND [END]
  GROUP BY D1.AnaId, D1.Date
) as values
WHERE values.AnaId = ana.AnaId
  AND values.Interval > ana.Interval
GROUP BY values.AnaId ORDER BY count DESC LIMIT 1
```

Current values extraction

Q11.1 Extract most recent values for each analog time series.

```
SELECT AnaId, Value FROM anavalues
WHERE (AnaId, Date) IN
(
  SELECT AnaId, max(Date) FROM anavalues
  GROUP BY AnaId
)
ORDER BY AnaId
```

Q11.2 Extract most recent values for each boolean time series.

5 Experiments

For the time being, this benchmark has been run against the data historian InfoPlus.21, the RDBMS MySQL, and the NoSQL DBMS Berkeley DB.

Data historians are proprietary softwares with distinctive designs and thus, performances. Given the EDF requirements, we chose InfoPlus.21, one of the most widespread data historians.

We selected the open source RDBMS MySQL due to its ease of use and for being perennial with a large user community, compulsory for industrial use. In our context, tuples are relatively small (e.g. 17 bytes for anavalues), and most columns are typically accessed. Additionally, query selectivity is low – ie. most tuples match the criteria – within the considered key range. These properties narrow down the benefits of using a column-oriented DBMS.

Lastly, we chose the ordered key-value store Berkeley DB, an open source library for embedded databases, for our experiments. This class of NoSQL systems adapts well to our typical usage based on range queries.

MySQL physical tuning The following results have been gathered with the InnoDB storage engine. The MyISAM storage engine has also been tested, but performances did not scale well with the amount of data, except for insertions. Results with MyISAM are not detailed in this paper.

By default, InnoDB uses a clustered index on the primary key – here (AnaId, Date) and (BoolId, Date). Given the queries of this benchmark, and, altogether, typical queries on historical data at EDF, these indexes appear to be efficient for most of these. We did not define any additional index in order not to slow down insertions.

InnoDB is a transactional storage engine, which limits its ability to buffer insertions. As a result, we disabled this functionality by setting the following options:

```
innodb_flush_log_at_trx_commit=0
innodb_support_xa=0
innodb_doublewrite=0
```

To avoid parsing neither queries nor data, the benchmark uses MySQL C API prepared statements for insertions. Additionally, as MySQL allocates only one thread per connection, multi-threading is achieved by opening multiple parallel accesses (4 has been experimentally determined to maximize performances).

With their SQL definitions, queries Q9 and Q10 are not processed efficiently by MySQL – for instance, MySQL does not divide Q9 into multiple smaller range queries (one for each tag). This issue is solved by using stored procedures.

Berkeley DB physical tuning Berkeley DB transactional capabilities are also minimized to improve performances. `DB_TXN_NOSYNC` is set to disable synchronous log flushing on transaction commit. This means that transactions exhibit the ACI (atomicity, consistency, and isolation) properties, but not D (durability).

Write cursors (one per tag) are configured to optimize for bulk operations: each successive operation attempts to continue on the same database page as the previous operation.

The database is partitioned, with one partition per tag. Without partitions, insertions are about 60% slower.

For all systems, test servers are composed of a Xeon Quad Core E5405 2.0GHz processor, 3GB RAM and three 73GB 10K Hard Disk Drives with a RAID 5 Controller. For our experiments, only one processor core is activated due to a lack of optimization of our data historian for multi-threaded insertions.

Inserted data amounts to 500,000,000 tuples for each data type – analog and boolean – which sums to 11.5 GB without compression (and timestamps stored on 8 bytes). These tuples are divided between 200 time series (100 for each data type), individually designated by their identifier (AnaId or BoolId). 1,000,000 updates for each data type are then queried against the database; followed by up to 1000 SFW queries – 100 for Q9 and Q10, 1 for Q11.1 and Q11.2 – with different parameters. Date parameters for queries Q2 to Q8 are generated to access 100,000 tuples on average. Q9 and Q10 involve all analog time series, therefore each execution access 10,000,000 tuples on average.

Table 1 reports detailed results for each system. For instance, line 1 means thet executing Q0.1 500M times took 8 003.4 seconds for InfoPlus.21, 24 671.7 seconds for MySQL and 2 849.6 seconds for Berkeley DB. Figure 4 gives an overview of performance differences.

Table 1. Query execution times

Query type		Execution time (in s)		
	(amount)	InfoPlus.21	MySQL	Berkeley DB
Q0.1	(×500M)	8 003.4	24 671.7	2 849.6
Q0.2	(×500M)	7 085.8	24 086.0	3 115.8
Q1.1	(×1M)	16 762.8	12 239.5	9 031.5
Q1.2	(×1M)	16 071.3	13 088.2	9 348.5
Q2.1	(×1000)	267.6	410.4	693.0
Q2.2	(×1000)	215.1	284.5	655.4
Q3.1	(×1000)	252.5	186.6	531.4
Q3.2	(×1000)	216.7	181.8	533.2
Q4	(×1000)	263.0	192.6	536.8
Q5	(×1000)	236.7	185.7	514.0
Q6	(×1000)	235.6	191.9	513.1
Q7	(×1000)	234.0	234.2	507.7
Q8	(×1000)	231.2	277.7	506.5
Q9	(×100)	1 640.6	1 710.0	4 877.7
Q10	(×100)	1 688.8	7 660.7	4 977.5
Q11.1	(×1)	9.5×10^{-3}	1.15	2.75
Q11.2	(×1)	2.8×10^{-4}	1.13	4.81

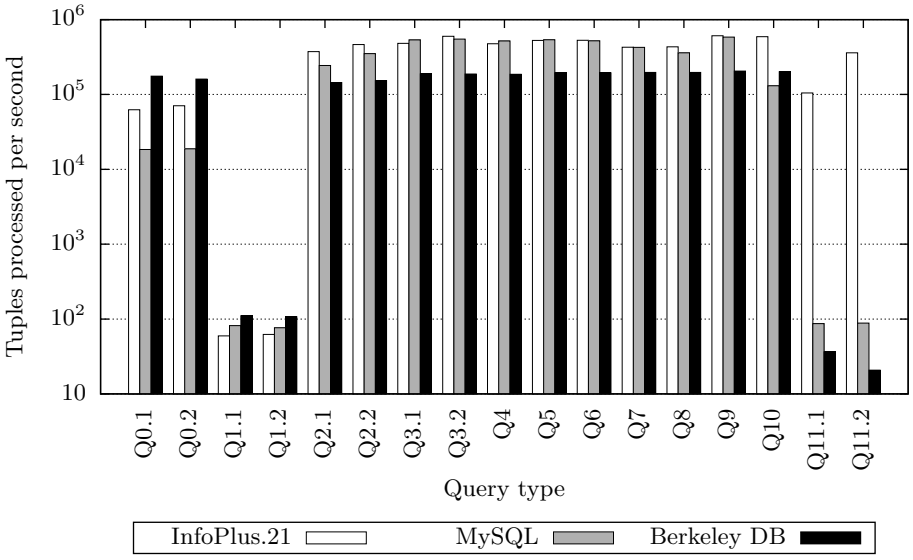

Fig. 4. Processing capacity

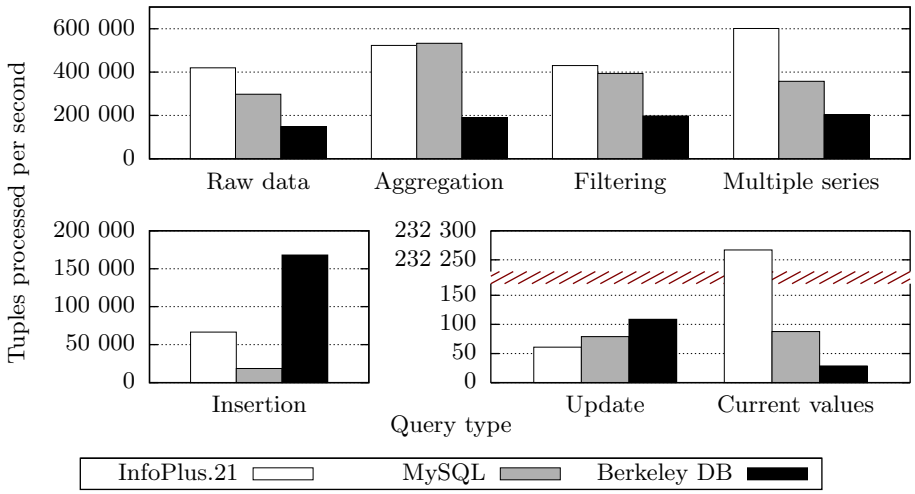

Fig. 5. Processing capacity by category

Different queries from the same category reporting similar performances ratios – ie. Q3, Q4, Q5 and Q6 for aggregate queries, and Q7 and Q8 for value filtering – are merged in figure 5 to summarize these results.

As advertised, data historians handle insertions efficiently compared to RDBMS: InfoPlus.21 reaches 66,500 insertions per second (ips), which is about 3.2× faster than InnoDB and its 20,500 ips.

Yet, Berkeley DB reaches 168,000 ips, that is, 2.5× faster that InfoPlus.21. However, it was used as an embedded library, without inter-process communication, which might significantly improve performances compared with MySQL or InfoPlus.21.

Current values extractions (Q11.1 and Q11.2) is the second anticipated strength of data historians, given their particular design with current values staying in main memory. This operation is performed several orders of magnitude faster than with MySQL (×1 850) or Berkeley DB (×6 140).

Additionally, InfoPlus.21 is faster for queries returning large results (Q2, Q7 and Q8). Since the SQL interface of MySQL involve some parsing overhead due to type conversions, we believe this overhead is important as we observed the same behavior with InfoPlus.21 SQL interface.

As for Q9 and Q10, InfoPlus.21 is faster than other systems. Physical data layouts possibly explain this behavior: InnoDB and Berkeley DB order their data according to the primary key (AnaId, Date), while data historians sort data by Date. In contrast with other queries, Q9 and Q10 investigates every time series, which are gathered in our data historian, but consist in several clusters with MySQL or Berkeley DB.

Apart from these queries, MySQL is slightly faster than our data historian on single time series (Q3, Q4, Q5 and Q6).

Overall performances for all systems, although notably different, are of the same order of magnitude, and do not ban RDBMS nor NoSQL systems from archiving industrial process data. Still, before considering any system for production purposes, additional studies with more realistic workloads are mandatory to attest their usability.

6 Conclusion

In this paper, we first highlighted data historization as a concurrent market segment with significant industrial needs. We then compared performances between a data historian (InfoPlus.21), a RDBMS (MySQL) and a NoSQL system (Berkeley DB) using a benchmark derived from a significant use case within EDF.

In light of our first experimental results, data historians could still be challenged when abstracting some business-oriented features. Lossy data compression, as well as efficient interpolation and resampling might involve important changes to the core of a DBMS, but industrial communication protocol support and various business-oriented clients supplied with data historians could be provided with independent specific developments. Disregarding business-oriented features, it makes sense to consider conventional DBMS for such industrial applications. Yet, in this context, specific optimizations for time series data insertions would bring value to relational data management systems, as this operation is critical for data historization.

To date, no benchmark is set as a standard to compare data historians together, nor analyze conventional DBMSs performances with regard to industrial automation data management.

References

1. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, 2003.
2. A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear Road: A Stream Data Management Benchmark. In *VLDB '04: Proceedings of the thirtieth international conference on Very large data bases*, pages 480–491, 2004.
3. Aspen Technology. Database Developer's Manual, 2007.
4. L. Golab and T. Johnson. Consistency in a Stream Warehouse. In *CIDR '11: Proceedings of the fifth Biennial Conference on Innovative Data Systems Research*, pages 114–122, 2011.
5. Invensys Systems. Wonderware Historian 9.0 High-Performance Historian Database and Information Server, 2007.
6. M. A. Olson, K. Bostic, and M. I. Seltzer. Berkeley DB. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, pages 183–191, 1999.
7. OPC Foundation. Data Access Custom Interface Standard, 2003.

8. OSIsoft. PI Server System Management Guide, 2009.
9. Transaction Processing Performance Council. *TPC Benchmark C Standard Specification*, 2007.
10. Transaction Processing Performance Council. *TPC Benchmark H Standard Specification*, 2008.