

Benchmarking Database Systems for the Requirements of Sensor Readings

Ciprian Pungilă, Teodor-Florin Fortiș and Ovidiu Aritioni

Research Institute e-Austria, Bvd. C.Coposu, 4, Room 045B, 300223, Timișoara, Romania

Abstract

Improving energy efficiency in order to reduce CO₂ emissions is a permanent challenge in the European space. Smart metering could help for improving energy efficiency by offering information about the way in which the energy is used. Smart metering will be based on large volumes of sensor data, since energy monitoring will bring together sensor data from various critical areas. The main purpose of this paper was to present the selection mechanism for a scalable storage solution, based on the requirements of the DEHEMS (Digital Environment Home Energy Management System) project. With regular sensor readings coming at every 6 seconds, there is an impressive amount of data collected even for the minimal target of about 250 households, 10 sensors per user. With these huge data streams that are non-stationary time-series data, collected at discrete intervals, the DEHEMS project has to offer a solution for storing and retrieving sensor data in a responsive way. We have tested both collection speed and aggregation speed for reasonable data streams of sensor data. The tests were performed on various database models, with their associated representations, including relational databases, key-value stores, column stores, self-tuning databases, as well as time-series enabled database systems. These experiments confirmed that column stores and key-value stores perform better than relational databases, while time-series databases outperform all the others.

Keywords

Benchmarking, Energy monitoring, Sensor data, Smart metering, Time-series data.

1. Introduction

Improving energy efficiency in order to reduce CO₂ emissions is a permanent challenge in the EU-27 space. Different projects and initiatives were developed around this target in order to improve energy consumption at different levels; databases on residential and on tertiary sector electricity consumption were developed and recommendation based on these developments were issued (such as the REMODECE and EL-TERTIARY projects).

As households account for about 30% of EU CO₂ emissions, with a significant part of household energy used for heating, the DEHEMS project will help in improving energy efficiency by supporting households to reduce their energy usage through better analysis and management of their energy consumption.

Different studies in energy conservation [1,2] show that behavior change in households is central to improving energy usage. According to [3], with automated room temperature control, there are potential savings of around 20-30 kWh/m².

Digital Environment Home Energy Management System (DEHEMS) developments will be based on

smart metering energy 'performance' models, which will monitor not only the levels of energy being used by a household, but will also look at the way in which the energy is used. In order to achieve these 'performance' models, energy monitoring will bring together sensor data from critical areas like appliance performance, or household heating. As sensor readings are expected to occur from various sources, it is expected that generated data collection is essentially a very large database that support storing and generating of statistical measures on sensor data. With a target of only 250 households, the generated amount of data is already extremely large. In order to select a scalable storage solution, it was in our intention to test the capabilities and limits of the different database storage engines for the requirements of large-scale implementation of the DEHEMS project.

The remainder of this paper is organized as follows: Section 2 offer the necessary background information, as well as the state-of-the-art analysis related with the objectives of the paper; Section 3 presents the specific requirements for data storage, as well as some methodological issues related with tests performed on various database systems; Section 4 includes setting details, and synthetic views of benchmarking results; Section 5 will conclude with the results of the tests performed, and offer an overview of future research.

2. Motivation and Related Work

The DEHEMS project intends to extend the state-of-the-art in energy monitoring with an expressed target to move to an 'energy performance model', by monitoring the levels of energy usage and, at the same time, looking at the way in which this energy is used. The project is supported by the UE under Framework Program 7, and it brings together companies and research centers from UK, Belgium, Bulgaria and Romania.

Sensor reading data will come from various sources: Energy meter systems and supporting sensors, or directly from household appliances. These data will have to be stored and analyzed against business intelligence in order to provide statistical and practical information to the householder in a 'green dashboard', accessible by using different means of communication. This information will include: Real-time carbon footprint assessments; real-time household energy efficiency ratings; recommendations for energy efficiency improvements, based on real-time household energy efficiency ratings; recommendations for different usage profiles that could match householder's usage needs in an efficient manner.

In the context of the DEHEMS project, data collections are essentially very large databases, which are able to store and generate statistical data based on sensor data, and to produce actions, recommendations and options for informing all or certain parts of the system. Energy information will be submitted to these databases without relying on any home storage support of these data. However, the DEHEMS project does not intend to innovate in the area of data storage and retrieval, but to seek for the best suitable database systems meeting the specific requirements of the project: The generated data set has to offer a unique view into household energy usage; also it has to enable further opportunities to disseminate the data for usage in other studies.

Similar initiatives exist in the area of energy management. The ECHONET Consortium (Energy Conservation and Homecare Network) implemented a home energy management system, equipped with a residential gateway [4]. The development of the IrisNet project was conducted towards '*a worldwide sensor web, in which users can query, as a single unit, vast quantities of data from thousands or even millions of widely distributed, heterogeneous sensors*' [5]. For the REMODECE project (Residential Monitoring to Decrease Energy Use and Carbon Emissions in Europe), the overall objective was '*to contribute to an increased understanding of the energy consumption in the EU-27 households for the different types of equipments*' [6].

The development of the IoT (Internet of Things) with household appliances will open new opportunities for developing IP-enabled solutions, more flexible and user-centric than previous approaches. With an increased number of online agents and with extended data acquisition possibilities, the data burden will significantly increase. Thus, our research could also apply to future IoT applications, as there will be an increased need to store and query for very large amounts of data (such as environmental readings, abstract events and others).

2.1 Time Series and Sensor Data

In [7] it is specified that context-aware/sensitive applications are data-centric. Data management is intensively used, ranging from data acquisition, data processing and data storage of environmental information. Based on the application model used, different approaches for implementing the data management can be considered. In the case of event-based data model, the attention is focused on event definition, detection and delivery. Sensor middleware applications, including TinyDB, DSWare, Mires and Impala offer support for an event-based data model.

The database programming model is used by several systems, such as TinyDB, Cougar or SensorWare. While TinyDB is using a technique of applying distributed queries, in the case of Cougar and SensorWare, a different technique, CACQ - Continuously Adaptive Queries over Streams, is used [8]. In the case of CACQ, the query process will first build a summary of measurements for an individual sensor, and then this summary is transmitted over the network to other sensors [9].

Three different approaches for implementing data storage were identified in [7,10]: External, local and data-centric storage. In the case of external storage, data is stored in a database station outside the sensor network, while in the case of local storage, data is stored at the place where it was generated. However, data-centric storage, a compromise between the other two approaches, seems to be the preferred approach.

Time series sensor-generated data can be obtained as a result of stochastic processes, as specified in [11]. Stochastic process techniques can be used to simulate the sensor behavior. It represents a challenge to understand the semantic of data and to develop stochastic model according to these semantic. Historical time-series sensor-generated data analysis can be used to predict future values [12].

Three different models for time-series generated data can be considered [11]: Auto-regressive, integrated, and

moving average models. The auto-regressive model is a kind of random process used to model and to predict various types of environmental process or phenomenon. The moving average model has as a central goal to reduce future variations of time-series generated signals. The 'Averaging Methods' and 'Exponential Smoothing Methods' are 'smoothing' techniques used to capture these variations.

For storing sensor-generated data various database systems can be considered, ranging from relational databases, to very large databases with a simple star schema, or time-series databases. In the case of relational databases, there are major difficulties caused by the index structure used. In [13] it is proposed the Skyline Index, which can be used to partially solve the problem of relational databases adaptation to a large amount of data. The algorithm can be used to construct indexes of time-series data, by applying the Skyline Bounding Regions technology. This technology approximates a group of time-series with tighter bounding regions, limited by the top and bottom skylines of their graphical representation.

3. Data Storage Requirements

As the DEHEMS project targets to serve a large number of household sensors, the requirements on the emerging software system are quite large: There are high expectations for the communication, data storage and querying layers of the overall architecture. These layers are highly interrelated and there exist important cooperation issues that must be addressed.

The DEHEMS system has to fulfill two major goals: Tracking of energy consumption, if possible for each device, for small periods of time; and enabling users and researchers to observe the emerging consumption patterns, by providing useful statistics, or employing complex data-mining techniques. There is a common characteristic for these requirements: Responsiveness, in order to track energy consumption on-line/real-time responsiveness is required, though consumption

analysis could be performed offline, based on previously collected data. Also, there is a clear difference between the two requirements in volume: While sensor readings are expected on a 24/7 basis from each household, consumption analysis could be performed only on a limited set of information coming from a selected subset of users at a time.

In the overall architecture of the application, each household uses several consumption meters (*sensors*) that are regularly sending their readings to some intermediate devices (*gateways*), one device per household. Collected data is submitted to a central system (*aggregator*), one for each large region.

The role of a single sensor is to notify spontaneous consumption, and report it to the gateway, at a regular rate, based on time interval supported by specific hardware devices. The specific role of the gateway is to collect individual readings, queue those readings coming from different sensors and forward these readings to an aggregator, at a regular rate, similar with the reading rate. The aggregator itself is composed of two different layers: The communication layer, and the storage and querying layer.

As one can see from Table 1, when we reach a target of only 100 000 users, with an average of 10 sensors per user, there will be around 166 667 requests per second within the system. Based on these figures, one can identify the following three issues: The ability to communicate at very high data exchange rates, the ability to support data registration (inserts) at a very high data rate and the ability to hold extremely large amounts of data on a long-term basis.

In order to address the first of these issues, one has to use some modern, EDA-based systems (Event Driven Architecture), associated with the implementation of some load-balancing techniques. As for the last two issues, one cannot consider any solution that is not based on distributed systems. Several approaches are possible:

Table 1: Estimation table based on potential number of users of the DEHEMS architecture

Users	Sensors per user	Read/min.	Rec./sec.	Rec./hour	Rec./day	Rec./month	Rec./year
1	1	10	0.16	600	14 400	432 000	5.256 mil
250	1	10	41.6	150 000	3.6 mil	108 mil	1.314 bil
1000	1	10	166.6	600 000	14.4 mil	432 mil	5.256 bil
5000	1	10	833.3	3 mil	72 mil	2.16 bil	26.28 bil
100 000	1	10	16 666.6	60 mil	1.44 bil	43.2 bil	525.6 bil
1 mil.	1	10	166 666.6	600 mil	14.4 bil	432 bil	5.256 tril
250	10	10	416	1.5 mil	36 mil	1.08 bil	13.14 bil
1000	10	10	1666.6	6 mil	144 mil	4.32 bil	52.56 bil
5000	10	10	8333.3	30 mil	720 mil	22.32 bil	262.8 bil
100 000	10	10	166 666.6	600 mil	14.4 bil	432 bil	5 256 bil
1 mil.	10	10	1.666 mil	6 bil	144 bil	4.32 tril	52.56 tril

Either use the specificity of sensor data, which can be organized as time-series data, or use a simple, column-based database system.

As the DEHEMS project has a special interest for existing database systems to be able to fit with project requirements, with a specific orientation towards open-source or community products, our current benchmarks were devised in order to meet the specific requirements of the project. These tests are not trying to validate and compare the different techniques implemented in various database systems, but to find and propose a set of suitable systems that are able to answer as far as possible to the three major issues, as identified above.

3.1 Data Collection Methodology

While the ability to communicate at very high data rates can be addressed by using some EDA-based systems, special attention must be paid to the ability of inserting incoming data at very high speeds, and the ability to hold extremely large amounts of data on a long-term basis. Thus, the initial requirements of the tests performed are related with both storing the data, and aggregating data under specific conditions.

With a set of target database systems ranging from time-series systems, to column-based or relational systems, the data model was chosen such that it is supported by most of these systems. The following items were identified as necessary for holding sensor readings information: a) *Client identification*: Household

identification; b) *sensor identification*: The actual sensor, inside a household; c) *reading timestamps*, required in order to build various statistics; d) *actual reading data*, coming from the sensor node.

The benchmark consists of two distinct stages: The storing stage and the aggregation stage. During the storing stage [Figure 1], a significant amount of (randomly generated) records are produced, and passed to the target database system. In order to ease the process, some transformations between the 'neutral' record and effective data structure, as required by the target database system must be performed.

For the insertion algorithm, two different approaches were considered. Regular, large-scale tests based on around 10 million inserts, this corresponding to randomly generated sensor data coming from random «household, sensor» identifiers. Data points were created at each 1000 inserts, in order to evaluate the total number of insert operations performed so far. The time elapsed for each batch of operations, as well as insert speed were evaluated. Second, small-scale tests were based on around 4 million insert operations. For this second test, data points were generated at each 1000 operations.

During the aggregation stage [Figure 2], we are interested in computing the response time for simple queries, including: The minimum or maximum value for readings coming from a specific (household, sensor) combination, for a specified time frame; the mean value for readings coming from a specific (household, sensor) combination, for a specified time frame.

However, because of the limitation of column-oriented databases and key-value stores, which were not designed to allow aggregations, the benchmark related to the aggregation has been transformed into a scanning

```

ensure: database is empty
timestamp ← 0
buffered ← recordsPerSecond
for i from 0 to maximumRecords do
  client ← random(0, maxClients)
  sensor ← random(0, maxSensors)
  reading ← random() {the actual value is not important}
  insertRecord(client, sensor, timestamp, reading)
  timestamp ← timestamp+1
  if mod(i, 10000) = 0 then
    {notify the wall clock time elapsed from the last report}
  end if
  if elapsed more than a second since last buffer fill then
    buffered ← recordsPerSecond
  end if
  if buffered = 0 then
    {wait until the second elapses}
    buffered ← recordsPerSecond
  end if
end for
    
```

Figure 1: Insertion algorithm.

```

ensure: database is populated
i ← 0
for client from 0 to maxClients, sensors from 0 to maxSensors do
  begin ScanSortedByTimestamp(client, sensor)
  while can_scan() do
    timestamp, reading ← retrieve_scan()
    {ensure that the timestamp is greater or equal to the previous one}
    i ← i+1
    if mod(i,10000) = 0 then
      {notify the wall clock time elapsed from the last report}
    end if
  end while
end for
    
```

Figure 2: Scanning algorithm.

benchmark: Instead of querying the database system for the minimum or average value for the given criteria, we are asking for the entire set of data records matching the given criteria and being ordered by their timestamps. This change was performed in order to allow data-mining techniques to be used easily on generated data.

4. Benchmarking Results

The target set of database systems included the following: IBM Informix – community edition, Hypertable, PostgreSQL, MySQL, Oracle BerkeleyDB, SQLite3.

- Oracle BerkeleyDB (<http://www.oracle.com/technology/products/berkeley-db/db/index.html>) is an open source, embeddable database engine. It is offered as a library that links directly into applications, enabling applications with predictable access patterns. Since queries required for the DEHEMS project require access to sorted data, the B-Tree implementation offered by BerkeleyDB can be considered as being an interesting approach for solving the data storage problem efficiently.
- Hypertable (<http://hypertable.org/about.html>) is 'a high performance distributed data storage system designed to support applications requiring maximum performance, scalability and reliability'. It was designed 'to manage the storage and processing of information on a large cluster of commodity servers, providing resilience to machine and component failures.'
- IBM Informix® Database™ expands its functionality by the TimeSeries DataBlade module (<http://www-01.ibm.com/software/data/informix/>). The module adds sophisticated support for the management of time-series data and temporal data.
- MySQL, an open-source database system (<http://www.mysql.com/>) is 'a very fast, multi-threaded, multi-user and robust SQL database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software.'
- MonetDB (<http://monetdb.cwi.nl/projects/monetdb/MonetDB/index.html>) is an auto-tunable system, based on the BAT (Binary Association Tables) algebra kernel and functional modules and MIL (MonetDB Interface Language). MonetDB offers a high-performance database system, column-store based and extendable, supporting multiple query languages.
- PostgreSQL (<http://www.postgresql.org/about/>) is a powerful open-source ORDBS, highly scalable in the number of concurrent users, and in the quantity of data it can manage. It offers generous limits regarding database size, Table size, row size or indexes per table.
- SQLite (<http://www.sqlite.org/about.html>) is an in-process library implementing a self-contained, transactional SQL database engine. It is an embedded,

open-source, database engine, storing databases directly in a disk file.

Based on the data storage requirements presented in Table 1 earlier, we have concluded that it was necessary to simulate the functionality of a real-world environment for the tests performed, which is why we have organized the tests in two categories.

The first category of tests, referred as *large-scale tests* from this point onwards, involved inserting 10 million records into the database using different insertion rates, in accordance to the data acquisition rates from Table 1. The amount of records to be inserted per second was 83,340, 166,670 and 416,000 records per second, which corresponds to 50,000, 100,000 and 250,000 users, and 10 sensors per user.

The second category of tests, referred as *small-scale tests* from this point onwards, involved inserting 4 million records into the database, again using different insertion rates, these being: 1670,8340 and 83,340 records per second, corresponding to 1000, 5000 and 50,000 users, with 10 sensors per user.

4.1 Results for Large-scale Tests

Insertion algorithm for the large-scale tests was based on around 10 million inserts, corresponding to randomly generated sensor data coming from random «household, sensor» identifiers, and with incremental timestamps. The algorithm, as depicted in Figure 1, was modeled to deal with 50,000, 100,000 and 250,000 users, respectively. In the case of the scanning algorithm, «household, sensor» identifiers were iterated up to the limits specified in the algorithm from Figure 2; readings being sorted by their timestamps.

The overall results for large-scale tests are synthesized in Table 2. Comparative performance of database systems when performing the large-scale tests is illustrated in Figure 3.

4.2 Results for Small-scale Tests

Insertion algorithm for the small-scale tests was based on around 4 million inserts, corresponding to randomly generated sensor data coming from random «household, sensor» identifiers, and with incremental timestamps. The algorithm, as depicted in Figure 1, was modeled to deal with 1000, 5000 and 50,000 users, respectively.

The overall results for small-scale tests are synthesized in Table 3. Comparative performance of database systems when performing the large-scale tests is illustrated in Figure 4.

Table 2: Overall results for large-scale tests

Database system	Average inserts (thousands operations)			Deviation – inserts – (thousands operations)		
	50 000	100 000	250 000	50 000	100 000	250 000
BerkeleyDB	120.44	120.22	120.13	44.64	44.58	45.07
Hypertable	260.79	264.80	262.32	52.12	55.27	54.46
Informix	73.65	10.99	73.51	3.64	33.32	3.64
MonetDB	10.71	10.64	10.78	0.577	0.613	0.571
MySQL	4.67	4.29	4.49	7.30	7.02	7.11
PostgreSQL	44.90	45.66	45.89	26.19	26.27	26.47
SQLite3	18.61	18.43	18.40	8.75	8.71	8.65
	Average scans (thousands operations)			Deviation –scans – (thousands operations)		
	50 000	100 000	250 000	50 000	100 000	250 000
BerkeleyDB	1760	1760	1.730	24.78	23.80	19.47
Hypertable	22.37	22.58	22.56	1.60	1.67	1.66
Informix	136.79	117.30	133.16	0.776	0.526	0.743
MonetDB	135.89	135.84	135.99	6.35	5.91	5.94
MySQL	273.94	279.58	277.19	6.17	5.20	6.40
PostgreSQL	62.22	61.43	60.72	17.06	17.29	17.37
SQLite3	281.12	280.78	280.03	1.92	2.67	2.28

Table 3: Overall results for small-scale tests

Database system	Average inserts (thousands operations)			Deviation – inserts – (thousands operations)		
	1000	5000	50 000	1000	5000	50 000
BerkeleyDB	158.81	160.98	153.40	41.92	44.67	46.03
Hypertable	267.69	277.32	269.29	35.48	37.37	38.16
Informix	7.15	47.65	76.62	20.55	36.11	4.19
MonetDB	10.81	10.82	10.71	0.466	0.757	0.622
MySQL	16.57	15.25	14.15	6.75	7.33	6.88
PostgreSQL	908.86	925.90	930.50	721.92	735.04	738.16
SQLite3	23.36	24.39	24.75	13.12	12.78	12.80
	Average scans (thousands operations)			Deviation – scans – (thousands operations)		
	1000	5000	50 000	1000	5000	50 000
BerkeleyDB	1700	1680	1650	34.29	34.42	32.56
Hypertable	7.41	7.41	7.68	0.673	0.683	0.667
Informix	105.30	105.76	108.72	0.827	0.851	0.479
MonetDB	96.71	96.43	97.28	5.22	5.49	5.28
MySQL	168.05	173.44	168.77	15.33	11.79	16.01
PostgreSQL	66.06	64.81	61.77	4.62	4.86	5.92
SQLite3	290.44	290.41	291.59	3.97	5.08	3.15

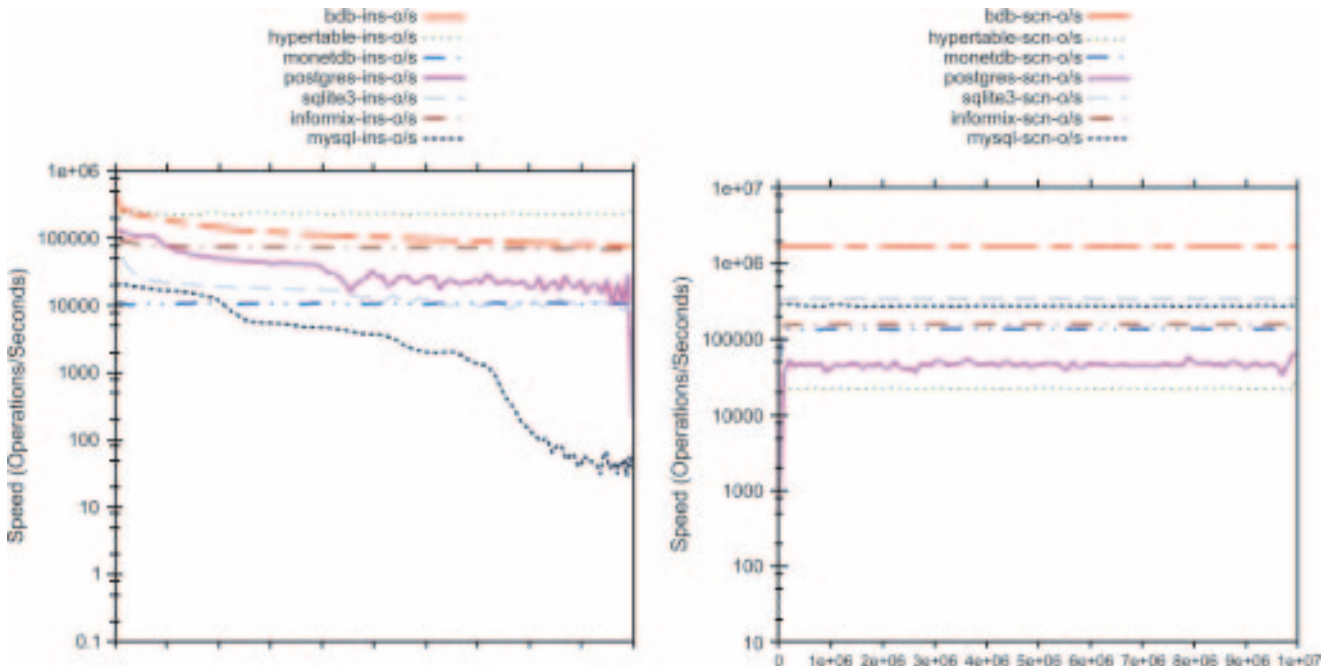


Figure 3: Results for large-scale tests in logarithmic scale: inserts (left), scans (right).

5. Conclusion and Future Work

The nature and flow of insert operations heavily impacts the storage performance of various database systems: When most of the queries performed are for collecting all items per specified interval, a key based on (timestamp/client identifier/sensor identifier) must be used; on the other hand, when most of the queries performed are for collecting all items per specified client, a key based on (client identifier/sensor identifier/timestamp) must be used. Thus, we might have to have two different schemas for storing the same information.

While our tests were performed for randomly generated data, sensor readings could offer data streams with ‘almost’ sorted information. This characteristic can be achieved by using a key based on (timestamp/client identifier/sensor identifier). Some database systems, like BerkeleyDB, obtained a significantly increased performance due to this characteristic. When facing clustered data, with a key based on (client identifier/sensor identifier/timestamp) combination, the performance decreases logarithmically in the case of these database systems.

Hypertable offer the largest number of insert operations per second (average), while scan operations are at

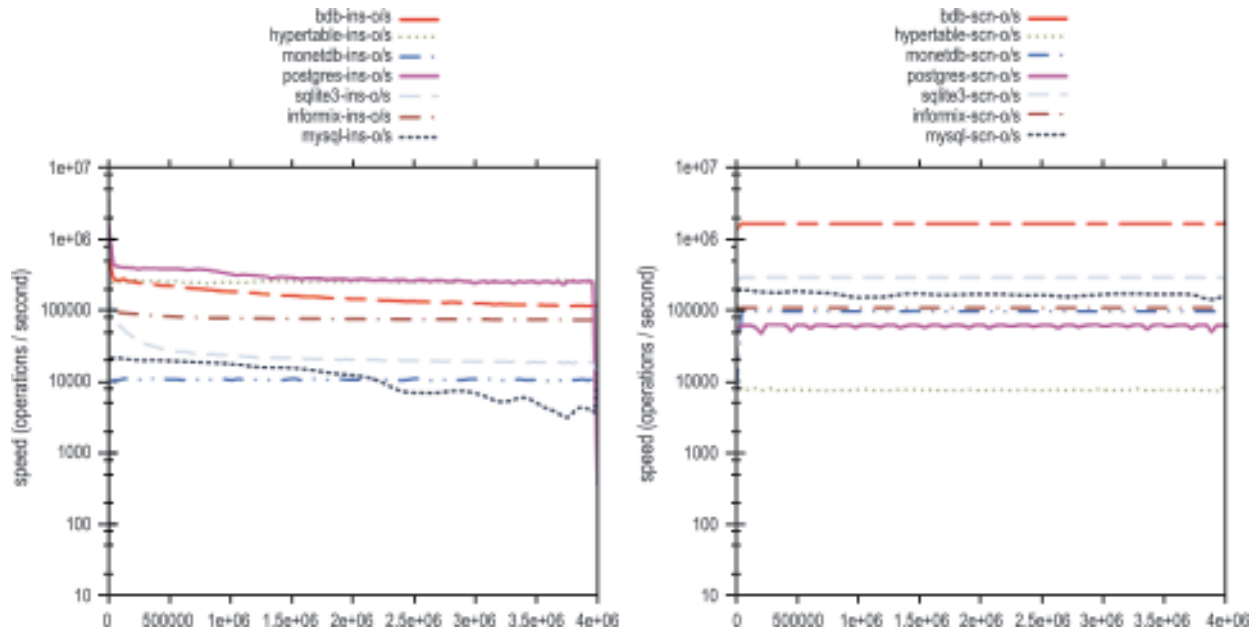


Figure 4: Results for small-scale tests in logarithmic scale: inserts (left), scans (right).

a reasonable rate. However, considering that scan operations are to be performed at lower rate than insert operations, Hypertable offer good performance and could be considered as one of the options for the DEHEMS project. Even if Informix, a time-series database system, has lower insert performances as compared with Hypertable, the overall performance combined with its time-series orientation recommend this product as being an interesting candidate for the DEHEMS project, too. Finally, BerkeleyDB offer an interesting compromise between a higher data acquisition rate and a good data scanning capability.

Our future developments in the context of the DEHEMS project are related with modeling an architecture for data storage and aggregation. Currently, a prototype for simulating the functionality of the system is being developed, together with an engine for data pattern analysis, engine that will be used for predicting user and device behavior, and for helping users in making decisions about their energy consumption habits.

A central goal is related with optimization of data insertion process. In order to implement some optimization mechanisms, it could be useful to detect patterns in sensor-generated data, improving operation for predictable data, and performing full insert operations for unpredictable data. A secondary goal related with this issue is to detect the change-points in time/data series and to store these change-points in the database. By using this technique one can reduce the amount of data stored and improve the overall performance of insert operations. Another important goal is related with a middleware for sensor data acquisition adapted

to the specificities of received information. The problem of detecting new sensor/devices used to measure the current cost is a challenge both for the DEHEMS project and for the scientific community. The middleware sensor data acquisition must use a reflexive architecture, to be an auto-adaptive software. The sensors must be able to connect or disconnect from our system in an easy way. These features will build a new vision of our persistence architecture.

Acknowledgments

This work was carried out with partial funding from the EU through the STREP project DEHEMS (FP7 reference number 224609) and from the Romanian DEHEMS-RO project RO_PNII_20 EU/11.06.09. The views expressed in this paper do not necessarily reflect those of the DEHEMS consortium members. Further information on DEHEMS can be obtained from <http://www.dehems.eu>. Special thanks go to our colleague, Ciprian-Dorin Crăciun, who helped in setting up and conducting these tests.

References

1. T. Jackson. Motivating sustainable consumption, a review of evidence on consumer behaviour and behavioural change. Technical report, Sustainable Development Research Network (SDRN), 2005.
2. M. Letcher, Z. Redgrove, S. Roberts, B. Longstaff, and A. Inverarity. Mobilising individual behavioural change through community initiatives: Lessons for climate change. Technical report, Centre for sustainable energy, 2007.
3. M. Kleeman, U. Birnbaum, R. Heckler, G. Kolb, P. Markewitz, and K. Leubner. Systematisierung der potenziale und optionen für den gebäudebereich. Technical report, Forschungszentrum Jülich, Programmgruppe Systemforschung und Technologische Entwicklung, 2001.
4. N. Kushiro, S. Suzuki, M. Nakata, H. Takahara, and M. Inoue.

- Integrated residential gateway controller for home energy management system. *IEEE Transactions on Consumer Electronics*, 49:629-36, 2003.
5. P.B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a world-wide sensor web. *IEEE Pervasive Computing*, 2(4), 2003.
6. Anibal de Almeida, Paula Fonseca, Barbara Schlomann, Nicolai Feilberg, and Carlos Ferreira. Residential monitoring to decrease energy use and carbon emissions in Europe. In *Proceedings of "International Energy Efficiency in Domestic Appliances and Lighting Conference 2006"*, 2006.
7. S. Ratnasami, D. Estrin, R. Govindran, B. Karp, and S. Shenker. Data-centric storage in sensor-nets. In *1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 78-87, 2002.
8. S. Madden, M. Shah, J.M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *In SIGMOD*, pages 49-60, 2002.
9. M.M. Gaber (Editor), and Joao Gama (Editor). *Learning from Data Streams: Processing Techniques in Sensor Networks*. Springer, 2007.
10. S. Tilak, N.B. Abu-Ghazaleh, and W. Heinzelman. Collaborative storage management in sensor networks. *International Journal of Ad Hoc and Ubiquitous Computing*, 1:47-58, 2005.
11. B.C.E. Pelham, and J. Gwilym M. Time series analysis: Forecasting and control. 1994.
12. C. Liu, K. Wu, and M. Tsao. Energy efficient information collection with the Arima model in wireless sensor networks. *Global Telecommunications Conference*, (4), December 2005.
13. Q. Li, I.F. Vega Lopez, and B. Moon. Skyline index for time series data. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):669-84, 2004.

AUTHORS



Ciprian Pungilă is a Junior Researcher at the e-Austria Research Institute in Timișoara, Romania, and a teaching assistant and PhD student at the Department of Computer Science at West University of Timișoara. He received his MSc degree in Distributed Systems and Artificial Intelligence from the same university and a Computer Engineering degree from the 'Politehnica' University of Timișoara. His research interests include Software Engineering, Static and Dynamic Analysis and Intrusion Detection Systems.

E-mail: cpungila@info.uvt.ro



Teodor-Florin Fortiș is an Associate Professor in the Department of Computer Science at West University of Timișoara, Romania, Department of Computer Science. Also, he is a Senior Researcher at the e-Austria Research Institute in Timișoara, Romania. He received his PhD in Computer Science from West University of

Timișoara, Romania in 2001.

His research interests include Formal Languages, Web and Workflow Technologies, Service-Oriented Computing.

E-mail: fortis@info.uvt.ro



Ovidiu Aritoni is a junior researcher at the Research Institute e-Austria in Timișoara, Romania, and a teaching assistant and PhD student at the Department of Computer Science at West University of Timișoara. He received his MS. degree in Distributed Systems and Artificial Intelligence from the same university. His research interests include Software Engineering and Ambient Intelligence.

E-mail: oaritoni@info.uvt.ro

DOI: 10.4103/0256-4602.55279; Paper No TR 204_09; Copyright © 2009 by the IETE