Real-Time Integration of Building Energy Data†

(Extended Abstract)

Diogo Anjos
Instituto Superior Técnico (IST)
Universidade de Lisboa
Lisbon, Portugal
diogo.silva.anjos@tecnico.ulisboa.pt

***Abstract*—Energy Management Systems (EMSs) are used to monitor energy consumption in buildings with the purpose of improving energy efficiency, by identifying savings opportunities and misuse situations. To achieve that, an EMS collects energy metering data streams from a network of energy meters in the building. Sensor data must be processed in (*near*) real-time, to support a timely decision making process. Currently, EMSs are using traditional DBMSs to process such data, introducing a persistence step that translates to an unacceptable latency on data evaluation. Moreover, sensor data monitoring queries are not elegantly supported by the SQL query language, thus hampering the ability of an EMS to process energy metering data in real-time. Data Stream Management Systems (DSMSs) are used to process data streams efficiently in several domains. Many sensor network monitoring applications have been implemented upon DSMSs resulting in significant improvements on performance and overall resource usage. This work validates the hypothesis proposed in [1] that, to process energy metering data streams in real-time, EMSs should be supported by DSMSs, instead of DBMSs. We introduce an EMS's Data Processing Architecture supported by a DSMS that supports the implementation of an EMS capable of performing real-time data processing. We validate our solution through a benchmark evaluation against a DBMS based architecture. The results clearly show that the DSMS-based EMS outperformed the state of the art approach, both in data evaluation latency and query language expressibility—demonstrating its adequacy to process energy metering data streams in real-time.**

***Index Terms*—Energy Management Systems, Real-Time Monitoring of Energy Metering Networks, Querying Sensor Data Streams, Buildings Energy Efficiency, DSMS and DBMS Benchmark.**

I. Introduction

Buildings account for 40% of energy consumption, ahead of other sectors, such as industry or transportation [2]. Therefore, small improvements on building energy consumption translate to major savings. Among other ways, energy efficiency in buildings can be achieved through Intelligent Energy Management [3]. This topic concerns the monitoring of energy consumption and the careful tracing of its usage, in order to enable building managers to identify saving opportunities. Such monitoring is performed by an Energy Management System (EMS), through the gathering of data from the building energy metering network. The gathered sensor data is organized across several dimensions such as time, area, occupation, equipment state, expected consumption, among others, and then analyzed, to determine energy usage patterns, providing required information to determine the adjustments towards improving energy usage [4].

One fundamental aspect of energy management is timeliness: faster decisions translate to less waste and larger savings. In other words, up-to-date information greatly improves the decision making process, because building managers are able to immediately diagnose and promptly respond to anomalous situations [5]. EMSs are real-time decision making applications that require (near) real-time[1] integration of huge quantities of sensor data streams, wherein each stream tuple relates to a very short period. This means that, EMSs must be capable of continuously processing massive quantities of energy related data in real-time in order to improve the decision making process of building managers towards energy efficiency. However, there is a set of circumstances preventing this from being possible.

In recent years, many advances have been made in sensor technology, making it affordable and widely available, forming what we know as the *"Internet of Things"* (IoT) [6]. Such ubiquity of sensors is leading to pervasive sets of sensor data that, by being so large and complex, are not suitable of being timely processed by the traditional data processing systems, such as DBMSs [7]. In fact, this problem is requiring a lot of attention from the community, since it belongs to a trend of issues known as the challenges of Big Data, which are imposing a paradigm shift on how data is being handled [8].

A. Motivation

The data processing architecture of an EMS is generally supported by a traditional DBMS [9][10][11], which, as already said, is not capable to process sensor data streams in real-time [7]. Their persistent data model is not suited to extract relevant information from collected data in a timely fashion. For instance, let us consider the following queries:

Q1 Identify the energy meters reporting an energy consumption 10% above the respective average over the last 8 hrs.

†This Extended Abstract was submitted as a Regular Research Paper (12-pages length) to the "32nd IEEE International Conference on Data Engineering, **ICDE 2016**, Helsinki, Finland (May 2016)", with the title: "Real-Time Monitoring of Building Energy Metering Networks"

[1]In this work the terms "Real-Time" and "(Near) Real-Time" are used interchangeable to denote the ability of a system to deliver up-to-date date information.

Q2 Identify faulty meters that are not being able of produce energy consumption measurements every 60±5 secs.

Q3 For each meter, tell me how energy consumption is varying (%) from the average over the last 5 mins.

Q4 For each meter, give me its current and expected energy consumption. Being the expected value of each meter given by the average consumption of the current hour, computed along last (sliding) month.

The evaluation of queries such as those presented above raise a number of requirements that are not easily addressed by DBMSs. For example, in query Q1 (see Fig. 1): *(i)* the last 8 hours from *now* are always changing, therefore the query must be *continuously evaluated*; *(ii)* the time it takes to evaluate the query—query evaluation latency—should be small enough to allow the processing of new data as fast as it arrives (i.e. in real-time); and *(iii)* queries must be capable of compute data aggregates over moving subsets of the data stream, such as the AVG of Q1 over the 8 hrs sliding time window.

DBMSs only run queries over persistent data. Meaning that, to be queried, sensor data streams have first to be stored in disk to then be retrieved and evaluated in main memory, which imposes an unacceptable disk I/O latency for many data streaming applications [12], including EMSs. In addition, the DBMS query language model is not the most appropriate to pose queries requiring continuous evaluation. The one-time queries provided by DBMSs are designed to query static persistent data, having to be explicitly executed by the application, instead of spontaneously react to changes in the (time-varying) dataset. Moreover, one-time queries are evaluated in a batch manner, having the entire dataset to be evaluated before the output could be produced, which is not the most adequate strategy to process (potentially unbounded) data streams. Finally, DBMSs SQL language lacks operators to effectively manage time-varying data, such as time windows.

We believe that existing EMSs are not prepared to provide useful energy management information in a timely manner, neither their software architecture or functionalities are conceived to be a truly real-time data processing system [1]. Sensor data driven applications, such EMSs, are struggling to cope with a set of emerging challenges for which they were not initially conceived—process large volumes of data streams in real-time. Which is forcing the community to rethink the data processing infrastructure of these applications.

B. Data Stream Management Systems

A set of disruptive solutions have been developed to address these Big Data challenges, the Data Stream Management Systems (DSMSs) were introduced to effectively process data streams, such as energy metering sensor data [12][13]. DSMSs do not require the data to be persistent in order to be evaluated, the arriving data is placed directly *in memory* to be processed on-the-fly by *continuous queries*. By avoiding disk access overheads, DSMSs are capable of achieve query evaluation latencies that does not compromise the ability of the application to process data in real-time. Moreover, the query language model of a DSMS is the one that best suits the requirements

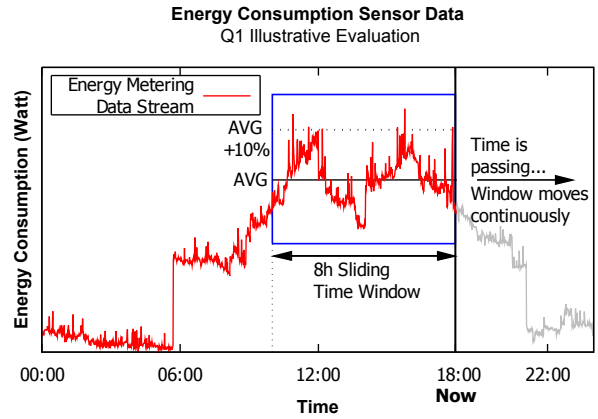

Fig. 1. Illustrative evaluation of query Q1. The 24-hours energy consumption sample shows the volatility of an energy metering data stream over time. For each meter, query has to continuously: 1) assess the energy metering data stream of *"last-8h-from-Now"*, 2) compute their average value, 3) identify if current value is 20% above average.

of the monitoring queries under consideration: *(i)* queries evaluate the arriving data streams incrementally, outputting a result for each data stream tuple that is evaluated, which is the most adequate approach to evaluate potentially unbounded data streams; and *(ii)* DSMS's query language is rich in time related operators, such as the sliding time window of Q1, which simplifies the implementation of queries on time-variant data. Across different domains, several monitoring applications that are sensor data driven, are being developed upon a DSMS (instead of a DBMS) with significant performance improvements. For instance, the monitoring of data on: stock market transactions [14], [15], network traffic [16], healthcare [17], [18] and environment [19]. Suggesting that this same approach must be followed to develop energy management applications.

C. Problem Statement and Contributions

The problem identified by this work is that: current EMSs are not capable of process energy metering data in real-time, since that, for that matter, they are supported by traditional DBMSs [9], which are known for being unsuitable to timely process sensor data streams, and the lack of expressibility to pose sensor monitoring queries [12]—seriously hampering the ability of an EMS to efficiently manage energy consumption in buildings. To solve this problem, this work aims to validate the hypothesis proposed in [1] that, an EMS supported by a DSMS would perform better on processing energy metering data, than the state of the art solutions supported by a DBMS. By better performance, we mean the ability to: *(i)* process energy metering data streams in real-time and *(ii)* provide a more suitable query language to evaluate these type of data within the requirements of this domain.

This hypothesis was validated, through the development of an EMS's Data Processing Architecture supported by a DSMS, and a benchmarking evaluation of its performance against a DBMS based solution. By pointing out how to develop an EMS capable of evaluate sensor data streams in real-time, we contribute with a solution that states how energy management applications must be developed in order to overcome the issues of Big Data, and thereby be capable of provide useful energy management information in a timely manner. Essential

to improve energy efficiency in buildings and achieve major energy savings.

II. EMS's Real-Time Data Processing Architecture

Our solution aims at creating a data processing architecture to integrate energy related data in real-time. Existing architectures, supported by DBMSs, process data in a batch manner through a *pipeline of data transformation steps*, impeding data to be processed in a timely manner [20][21][22][23][24]. This work proposes a Data Processing Architecture supported by a DSMS—instead of a DBMS—that allows EMSs to process data streams in real-time. The proposed solution adapts the *pipeline* stated before in order to process data continuously—in a streaming manner—allowing the freshness of data to be measured in seconds.

A. Architecture Overview

The solution architecture is depicted by Fig. 2 which identifies the three main architectural components of an EMS, and how they interact with each other: Data Acquisition Tier, Data Processing Tier, and Data Presentation Tier. The bulk of the complexity lies in the Data Processing Tier, for which we propose the Data Processing Architecture depicted, defined by the following main components: Adapters, Data Integration and Evaluation component, and Data Queues. We believe this is the best solution to implement an EMS capable of process data in data real-time, for the following reasons:

1) The proposed architecture is supported by a DSMS, known for being the most appropriate type of query engine to timely process sensor data streams.
2) The loose coupling of architectural components, allows to deploying them in fully distributed settings (such as cloud environments), by deploying each component in a cluster node, highly improving the systems scalability on huge workloads scenarios.

The proposed Architecture for the Data Processing Tier, as well the Data Acquisition and Presentation Tiers that compose an EMS are detailed below (see [1] for details).

Data Processing Tier. Is the core component of the solution. Conceptually, it works like a pipeline of data transformations, where data received by Data Acquisition Tier is *continuously* processed to produce the information required to feed the Data Presentation Tier, according to data stream application requirements. The data transformation flow is structured in stages using the following components:

Adapters mediate the extraction of data from several sources into the data transformation process. Adapters understand the sources data delivering model (push or pull based) and push data into remaining components of the architecture. Adapters may perform a set of data validation steps, such as identify and discard faulty tuples produced by faulty equipment that may hamper the process, and normalize into a common schema distinct data stream schemas that come from different types of sensors. The adapters role is critical to the effectiveness of all data transformation process: they bring to the pipeline only the strictly necessary data, pre-processed in the most convenient way, for the remaining data transformation process.

Data Integration is the core functionality of the data transformation process, which consists of data integration and cleaning steps. The main purpose is to combine and analyze several data streams, in order to compute a new set of data flows, adopting schemas that better fit the problem domain, and that will be used as input for domain specific queries. Note that, the integration of several streams are far from being a trivial process, raising several data quality issues. For instance, some data cleaning may be required in order to ensure data consolidation and consistency. These issues must be solved in this components, that must be able to merge data from multiple sources (e.g. sensor networks and databases), transform data under different schemes, recalculate and synthesize attributes, specify default values, calculate new attributes, etc.

Data Evaluation supports the evaluation of application queries including those that represent energy monitoring use-case scenarios. These queries have as input the previous integrated data streams, which represent available data sources for these application queries. From the evaluation of these queries will result the essential Key Performance Indicators (KPIs) used to support the decision making process. The timely computation of such KPIs depends on how suitable are the data streams produced by the Data Integration component.

Application Adapter converts the output streams into a format that can be understood by the Data Stream Application (e.g. the Real-Time Monitoring Dashboard).

Data Queues holds excess of data when the arrival rate of data stream tuples becomes higher than the processing capability of the receiver component, otherwise there would be loss of data. Queues will be placed at the entrance of the Data Processing Tier and between the most critical components (e.g. Data Integration and Data Evaluation), the ones that due their different data transformation complexity may yield data at different rates. Besides their major purpose, queues may also, if necessary, perform some additional computation, for instance to impose some priority order on the delivery of tuples or even to prevent its infinite growth through the usage of Load Shedding techniques, which carefully select tuples that may be discarded without largely affect the accuracy of produced results.

Data Acquisition Tier. Covers all data sources that may interact with Data Processing Tier. Sources may be splitted into two major types according to the nature of produced data: dynamic sensor data and persisted static data. The former is produced by the building energy metering network, used to monitor their energy consumption performance, which leads to a continuous production of sensor data streams.

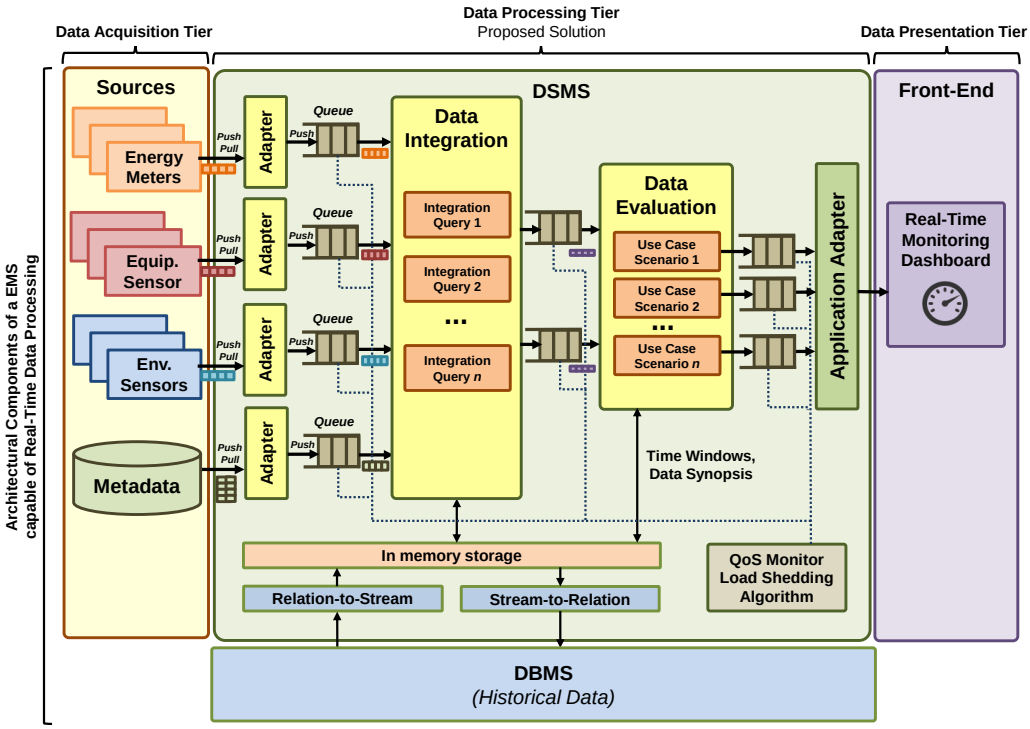

Fig. 2. Proposed architecture for the Data Processing Tier. From left to right, the data processing components of an EMS: the Data Acquisition Tier composed by the data stream sources, the Proposed Architecture for the Data Processing Tier, that is responsible for the integration and evaluation of gathered data streams, and the Data Presentation Tier where the computed results will be presented, typically in the EMS's real-time monitoring dashboard.

These data streams may be produced by three different types of sensors: energy meters, environmental, and equipment sensors. The later consists on building metadata that rarely undergoes changes (such as room areas, equipments by room, energy tariffs, etc.) and that is typically available through a database. Although less transient, metadata is highly useful when integrated with volatile data streams, contributing to improve the data stream processing.

Data Presentation Tier. Is the client of Data Processing Tier, consuming the information that is continuously produced through the evaluation of acquired data streams. From all EMSs data presentation applications, the real-time monitoring dashboard is the one that will benefit the most from the timely computation of produced results, thus its reference in the solution architecture. However, provided the proper adapter, any data stream client application could consume data from the Data Processing Tier.

B. Implementation

As discussed in [1], we conclude that the implementation of the prototype of the proposed architecture should be supported by **Esper**[2], a DSMS able to process Continuous Queries (CQs) over unbounded data streams. That is, the architectural data processing components—Data Integration and Evaluation—were implemented as a composition of CQs, to support the pipeline of data transformations in Fig. 2. Those queries are expressed through Esper declarative query language, that is compiled and optimized to an efficient query evaluation plan. Moreover, the Esper SQL-like query language allows a straightforward side-by-side comparison with equivalent queries written in DBMS's SQL. Which is of major importance since we are interested in compare the suitability of both DSMS and DBMS query languages to pose queries on the domain of monitoring energy metering networks. Therefore, Esper was used as the key building component of Data Processing Tier.

[2]http://www.espertech.com/products/esper.php

III. Requirements Analysis

In order to implement a reliable prototype of our solution, a requirements analysis was performed to assess the requirements that must be taken into account to conceive a representative solution in the domain of monitoring building energy metering networks. More specifically, we survey: *(i)* the type of data analysis that should be performed by the Data Processing Tier, and *(ii)* the properties of the data produced by a real energy metering network deployed in a large facility.

A. Use-Case Queries

We survey the features of the queries resulting from two application domains composing the scope of this work:

Sensor Network Monitoring. The class of queries used to timely evaluate sensor data streams, which enable the real-time monitoring of a sensor network, such as a building energy metering network [16] [12][25][14][26][27][15][18].

Building Energy Management Techniques. The building energy management techniques that has to timely evaluate

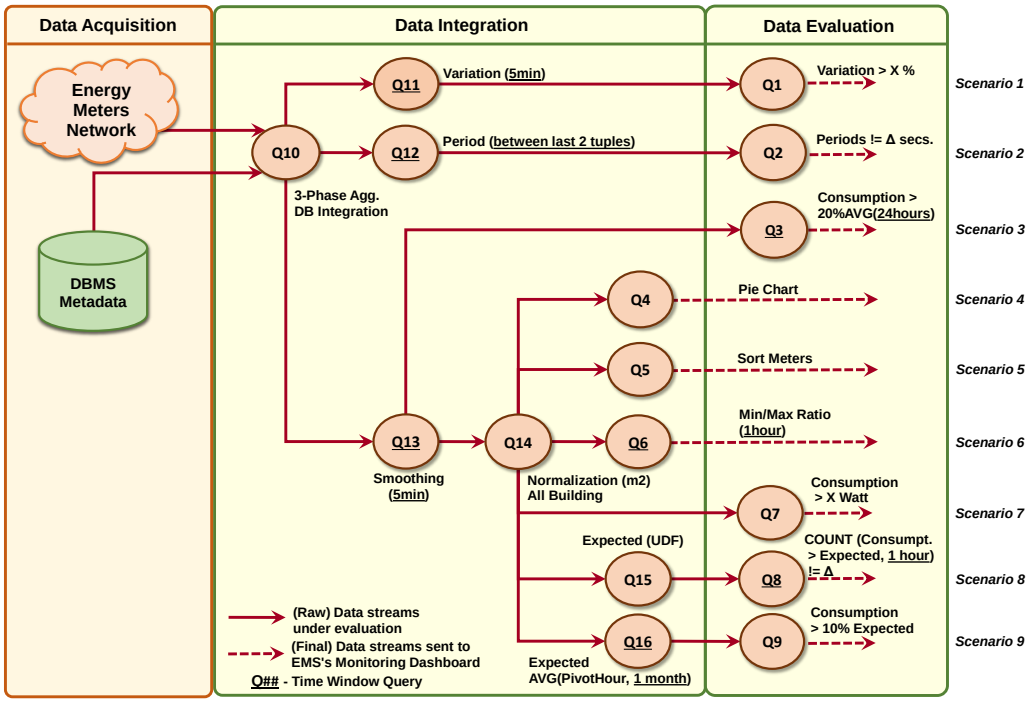

Fig. 3. Graph of Queries used in the Case Study. In a pipeline of data transformation queries, each query represents a data transformation step. Queries Q10–16 form a backbone of data integration queries used to convert raw data into a set of improved data streams that are able to simplify the implementation of queries Q1–9, the use-case scenario queries. Time Window Queries are flagged with an underscore (Q##).

huge quantities of sensor data, in order to effectively support the decision making process of building energy managers [4].

Such literature review allow us to take the following conclusions on the type of queries that must be supported by the proposed architecture:

Class of Queries. Monitoring queries are continuous queries, a class of long-running queries used to process time-series that are continuously consuming, evaluating, and producing time-variant data.

Main Operators. Those which evaluate data through complex time correlations, by performing sensor data aggregates over time windows.

Data Sources. Correlate data from different sources, such as sensor networks producing volatile and time-variant data streams, and from databases holding persisting static data.

Produced information. The required information lies on the scope of two main techniques: *(i) Load profile*, evaluates how energy consumption varies along the time and; *(ii) Peak Load Analysis*, evaluates the relationship between the minimum and maximum energy consumption along a given period of time (see [4] for details).

This survey allows us to conceive a representative set of use-case queries on the domain of monitoring energy metering networks, used to guide the implementation of our solution. The use case consists of 9 Use-Case Scenarios (Q1–9), that are supported by a backbone of 7 Integration Queries (Q10–16). The **9 Use-Case Scenarios** are as follows:

Q1 Identify the meters reporting an energy consumption variation 20% above the average over last 5 mins.

Q2 Identify faulty meters that are not being able of produce energy consumption measurements every 60±5secs.

Q3 Identify the energy meters reporting an energy consumption 20% above the respective average over the last 24 hrs.

Q4 For each meter, computes the fraction of its consumption relative to the total amount of energy being consumed by the building.

Q5 Sort the meters by decreasing order of the energy being consumed that they are reporting.

Q6 For each meter and building as a whole, compute the Min/Max energy consumption ratio over last-hour.

Q7 Identify the meters reporting measurements of energy consumption above a given threshold.

Q8 Identify the meters for which the number of reported measurements above its respective expected value, along last hour, lies between 5 and 10.

Q9 Identify the energy meters that are reporting energy consumption measurements 10% above the average of current hour, computed over last month.

The **7 Use-Case Integration Queries** are as follows:

Q10 For each meter, aggregate its three-phase measurements into the total amount of energy being consumed, and add to the resulting stream the location being monitoring, that is available through a database of metadata.

Q11 For each meter, tell me how energy consumption is varying (%) from the average over last 5 mins.

Q12 For each meter, compute the period between its two last measurements.

Q13 For each meter, smooth its produced data stream (for noise removal) through a 5-mins. moving average.

Q14 For each meter, normalize its energy metering mea-

surements according to the area covered by the meter ($Watt/m^2$), and also normalize all building current energy consumption by its total area.

Q15 For each energy meter, return its current and expected energy consumption value. With expected value computed through an User Defined Function.

Q16 For each meter, give me its current and expected energy consumption. Being the expected value of each meter given by the average consumption of the current hour, computed over last (sliding) month.

The Graph of Queries composed by these 16 queries is depicted in Fig. 3, presenting the *data processing pipeline* that must be implemented by our Data Processing Architecture. Each continuous query is a data transformation step of the pipeline, and the graph topology determines how queries interact with each other, in order to produce the required results. The 7 integration and 9 scenario queries are, respectively, supported by the Data Integration and Evaluation components of the proposed architecture (see Fig. 2).

B. Building Energy Metering Network

According to our solution, the Data Acquisition Tier is the component that is in charge of provide input data to the Data Processing Tier. In this work, the Data Acquisition Tier was supported by a real energy metering network deployed at a large facility, the Taguspark [3] University Campus of Instituto Superior Técnico. Allowing us to conduct our experiments on a dataset produced in a real scenario.

The network consists of 8 energy meters that are continuously monitoring the energy being consumed in different types of rooms. Each meter produces a new measurement every 15 seconds. Being each measurement composed by 3 readings (tuples), each one measuring the energy consumed by each phase of the 3-phase current that power supplies the building.

IV. Benchmark Evaluation Methodology

In order to fairly evaluate our proposed solution, we deploy two prototype versions of the proposed Data Processing Tier. One supported by a DSMS (see Fig.5 (a)), to assess the implementation feasibility of the proposed solution; and the other supported by a DBMS (see Fig.5 (b)) with the purpose of perform a side-by-side benchmark evaluation between the two architectural approaches. In this manner we assess the performance of the proposed architecture by comparing it with a state-of-the-art DBMS based solution. Thus, enabling to validate the hypothesis that: an EMS based on a DSMS performs better than common solutions based on a DBMS. By better performance, we mean:

1) Provide a most suitable query language to develop energy management applications.
2) The ability to process energy metering data streams in real-time.

To validate this hypothesis we have proceeded as follows:

[3]http://tecnico.ulisboa.pt/pt/sobre-IST/localizacao/#tagus

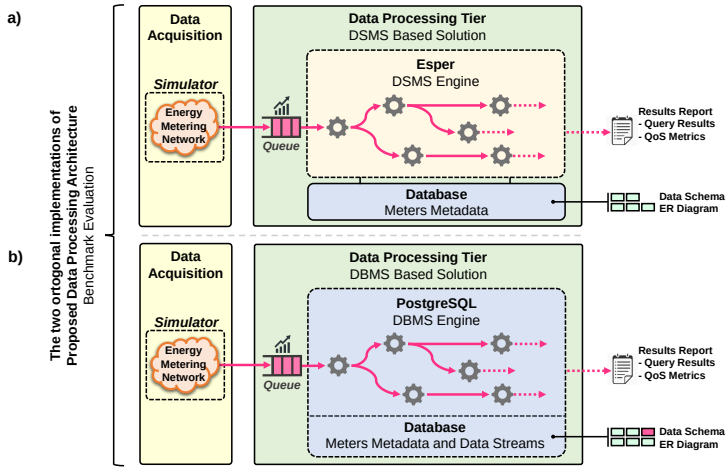

Fig. 5. Data Processing Architecture supported by two distinct query engines. Two types of query engines were used: DSMS and DBMS. To support the implementation of two orthogonal versions of the proposed Real-Time Data Processing Architecture. DSMS version assess the implementation feasibility of proposed solution (a), and DBMS version is required to run a benchmark validation between the two versions (b).

1) We implemented the use-case queries identified in Section III-A in both solutions of the Data Processing Tier. The implementation of these 9 use-case scenarios allows us to assess the ability of each query engine language to implement queries related with this problem domain.
2) We measured the time it takes for each solution to process energy metering data streams in each use-case scenario. Allowing to assess the capacity of each version of the Data Processing Tier to process data streams in real-time.

The relevant components of this setup are detailed below:

1) Selection of Query Engines: To implement both prototypes of the proposed Data Processing Architecture, we choose the following open-source engines:

Esper to support the DSMS version of Data Processing Architecture.

PostgreSQL to support the DBMS version of Data Processing Architecture, since it represents the most widely chosen type of DBMS to support EMSs [1].

2) Input Energy Metering Data Streams: The input data streams used in the evaluation experiments of the Data Processing Architecture are produced by the energy metering network introduced in Section III-B. Meaning that the solution evaluation experiments were supported by data from a real world scenario.

3) Input Data Queue: Both solutions were implemented with an Input Data Queue (introduced in Section II-A), used to hold data produced by the energy metering network. The query engine that continuously consumes data from the queue, blocking in the presence of an empty queue. The purpose of the queue is to hold the excess of data that results when the network production rate exceeds the engine consumption rate. When it happens, the quantity of queued measurements increases, making the time it takes to process a fresh mea-

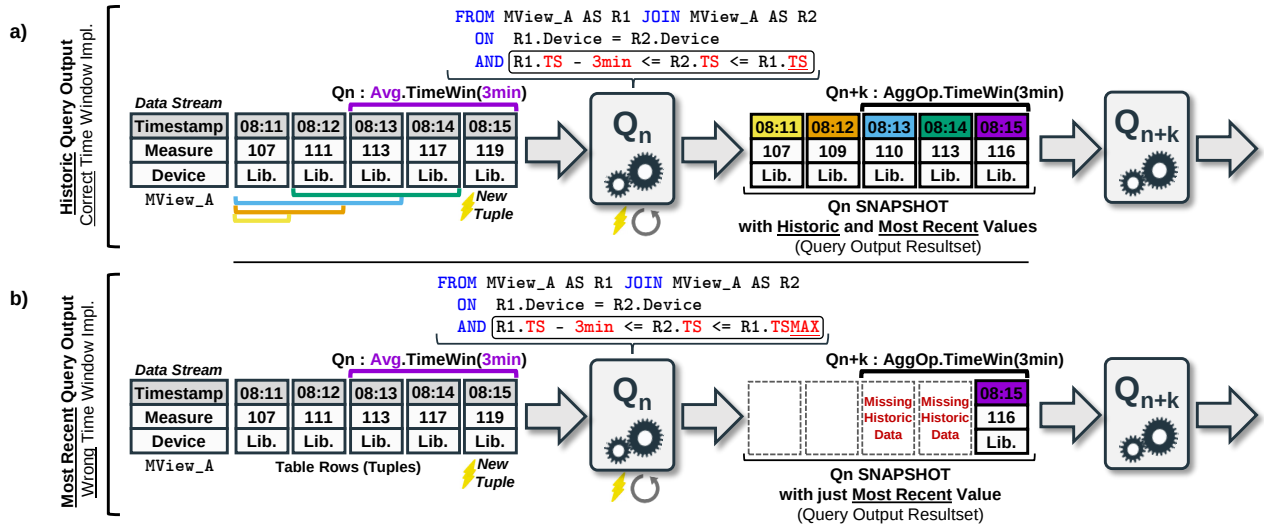

Fig. 4. Implementation of a Sliding Time-Window query in SQL with a Self-Join operation. The design of the self join condition determines the type of resultset—snapshot—produced by the windowing query. The queries of this work must be designed in order to output the entire data stream processed so far—Historic Snapshot (a). Instead of just output the evaluation results for the most recent measurements of each meter—Most Recent Snapshot (b). Otherwise, would not exist enough data to compute some windowing queries.

surement that has just arrived also increase—indicating a degradation of the system capability to processing data in real-time. Therefore, we will monitor the size of the queue and its waiting time during the system operation, to assess if data is being processed in real-time.

4) Data Schema: The data schema is used in the database of both prototype versions to store data, yet there is a remarkable difference on how each version rely on the schema. DSMS Based Solution only uses the entities related with static data, the ones used to store energy meters metadata. The energy metering data stream are not stored, being processed online and in-memory. By contrast, DBMS Based Solution uses the data schema to store both meters metadata and their produced data streams, meaning that the streaming data will be processed offline. This is a key difference between the two different solution approaches of the Data Processing Tier, they differ on how each one processes the energy metering data streams produced by the network (see[4] for details).

5) Produced Output and Query Results: A Results Report is maintained to log information produced in the course of data processing. For each energy meter measurement consumed from the queue and processed through the data transformation graph, a new entry is added to the log, recording the result value together with QoS metrics collected along the evaluation process. Such as the time that it takes to a meter measurement to traverse the graph topology (i.e. scenario evaluation latency), the quantity of queued measurements, and the total number of tuples already processed. This report will be used to assess the ability of each solution to process data in real-time.

6) Development Technologies: Both prototype versions were implemented in Java[5], being the DBMS solution supported by PostgreSQL[6] and the DSMS solution supported by Esper[7]. For auditing purposes, both prototypes are available at GitHub[8].

V. Query Language Evaluation

We aim to evaluate how suitable is the query language of each query engine to write and evaluate queries on this application domain, therefore we will describe the implementation of both prototype versions of the Data Processing Architecture according to the case study presented in Section III-A.

A. Time Windows and Temporal Data Correlations

Many of the use-case queries that were identified are Time-Window Queries. Requiring the performance of complex time correlations over sensor data streams through the evaluation of data aggregate operators over time windows.

1) In the DBMS Solution: Windowing queries are typically implemented through a Self-Join of the table holding the data stream, with a condition on the Timestamp attribute to specify the time window boundaries. Fig.4 exemplifies the computation of a three minute sliding average. However, there is a remarkable aspect in how DBMS queries produce their output that leads to a quite significant impact on the specification of time windows (self joins).

DBMS queries are processed in batch, meaning that from the entire dataset that is evaluated only a single result set will be produced as output. This output is produced from the current state of the database at the time the query is evaluated, and it is called a query snapshot. Each query results in a single snapshot at a time and it is replaced for a new one whenever the query is re-evaluated. A snapshot is a materialization of the result of a query. Therefore, at the graph, the snapshot of a query Q_n will be used as input by an upcoming query Q_{n+1}. If Q_{n+1} is a windowing query, then the snapshot of Q_n has to hold this query evaluation results for all the data

[4]https://github.com/diogo-gsa/EnergyMeteringNetwork/wiki/Data-Schema
[5]Java compiler version: 1.7.0 51-b13
JVM version: Java(TM) SE Runtime Environment (build 1.7.0 51-b13), Java HotSpot(TM) Client VM (build 24.51-b03, mixed mode)
[6]PostgreSQL 9.3.4, 64-bit (build 1600)
[7]Esper 5.0.0 - http://www.espertech.com/download
[8]https://github.com/diogo-gsa/data-processing-architecture

stream processed so far, see Fig.4 (a). Otherwise, upcoming queries Q_{n+1} will have missing data to properly compute their time windows, as illustrated in Fig.4 (b). As a result, the existence of time window queries requires all other queries (even the ones that are not windowing queries) to produce snapshots comprising the results of all data stream processed so far, which we designate as historic snapshots, instead of snapshots comprising only the most recent computed value for each device, which we designate as most recent snapshots.

The requirement of computing historic snapshots introduces a quite significant overhead in the query evaluation process. Each use-case query has to be (re)evaluated for each new arriving energy meter measurement, meaning that, besides the new measurement, all stored data stream will also be (re)evaluated in the process, in order to compute the historic snapshot, otherwise it will be erroneously produced a most recent snapshot. Meaning that, the computations that have already been preformed on the previous executions of the query will be repeated, in order to output the (same) historic snapshot updated with the evaluation result of the new tuple that has just been processed. As we will see in Section VI, this severely affects the ability of DBMSs to timely evaluate these queries. Moreover, this penalty overhead is exacerbated by the time window queries due the expensive self join operations that they have to perform along the entire data stream.

2) In the DSMS Solution: The windowing behaviour is trivially performed by the Data Window operators, provided by the EPL query language for this specific purpose. Those operators, used in the queries `FROM` clause, retain the arriving data stream tuples in a data buffer (i.e. window), that is dynamically updated according to a given windowing policy. Moreover, several windows can be combined together (chain of windows) in order to achieve complex windowing behaviours.

A DSMS query produces its output as a data stream of tuples, which greatly differs from the single snapshot produced by a DBMS. DSMS queries compute a single data stream tuple for each energy meter measurement that is evaluated, producing an output data stream that will be used as input by the remaining queries, avoiding to produce historic outputs. This happens due the DSMS ability of evaluate its queries in an incremental non-blocking manner, making the output values to be built incrementally through intermediate results. Therefore, the performance scalability of such queries is not affected by the overhead of having to produce historic snapshots, contrary to what happens in DBMSs.

To summarize, in the DBMS queries, time windows must be explicitly implemented with expensive Self-Join operations that become even more expensive due the need of output historic snapshots. On the other hand, in the DSMS queries, time windows are straightforwardly implemented with a set of window operators specifically provided for this purpose. Moreover, DSMS queries produce their output incrementally as a data stream—a new tuple is outputted each time the query evaluates a meter measurement—, which highly differs from the single historic snapshot produced by DBMS queries, which severely penalizes their performance, as we will see in Section VI.

B. Incremental Evaluation of Data Queries

The query evaluation process of each query engine is of utmost importance for the implementation and performance of our proposed solution. The different approaches are:

1) DSMS queries evaluation process: is of one tuple evaluated at a time, being produced a result for each evaluated tuple. That is, the query evaluation behaviour is *single tuple oriented*, and thus the input data stream is evaluated in a continuous manner, being the results computed incrementally and outputted along the evaluation process.

2) DBMS queries evaluation process: evaluates all dataset "at once", in a batch manner, producing a single output (snapshot) with the evaluation results computed over all dataset. Thus, the query evaluation behaviour is *all dataset oriented*, which defines the One-Time Queries approach.

In order to understand the impact of these two query evaluation models in the implementation of use-case queries, consider the following query:
"For each meter, return its current measurement and timestamp, together with the respective average of measurements received so far".
Although its simplicity, it holds a not so simple detail: this is a grouped/aggregated query that requires the projection of attributes that are neither aggregated nor grouped, the: `TS` (timestamp) and `Measure` attributes. The query is trivially implemented in the DSMS, the individual evaluation of each tuple (one at a time) makes it easy to solve the previous issue: the values of non-aggregated/grouped attributes (`TS` and `Measure`) that have to be projected are the ones belonging to the—single—tuple under evaluation, and the aggregation value (`AVG(Measure)`) that has to be projected is the one that matches the `Device` (energy meter) of the single tuple that is being evaluated, as depicted in Fig.6 (a). Conversely, in the DBMS such query is somehow cumbersome to implement. Batch evaluation approach makes it impossible to write this query in SQL-92 as it was written in the DSMS-EPL's language, since it would lead to the computation of an inconsistent output, as illustrated by Fig.6 (b). It depicts why the issue identified above could not be addressed with such query design that would cause a mismatch between the number of rows that are computed for the grouped/aggregated attributes (`Device` and `AVG(Measure)`) and the number of rows computed for the attributes that are neither aggregated nor grouped (`TS` and `Measure`). To overcome this kind of issues SQL-99 introduced the `WINDOW` clause, which like `GROUP BY` allows to specify a set of rows over which we could compute an aggregate operation. Yet, the `WINDOW` clause produces an output row for each evaluated row of the input dataset, differing from `GROUP BY` that outputs a single row for each dataset partition under evaluation. Fig.6 (c) shows how query under consideration could be implemented[9]

[9]By resorting to Self-Join operations this query could be written without using the Window clause, yet the query implementation would become even more complex.

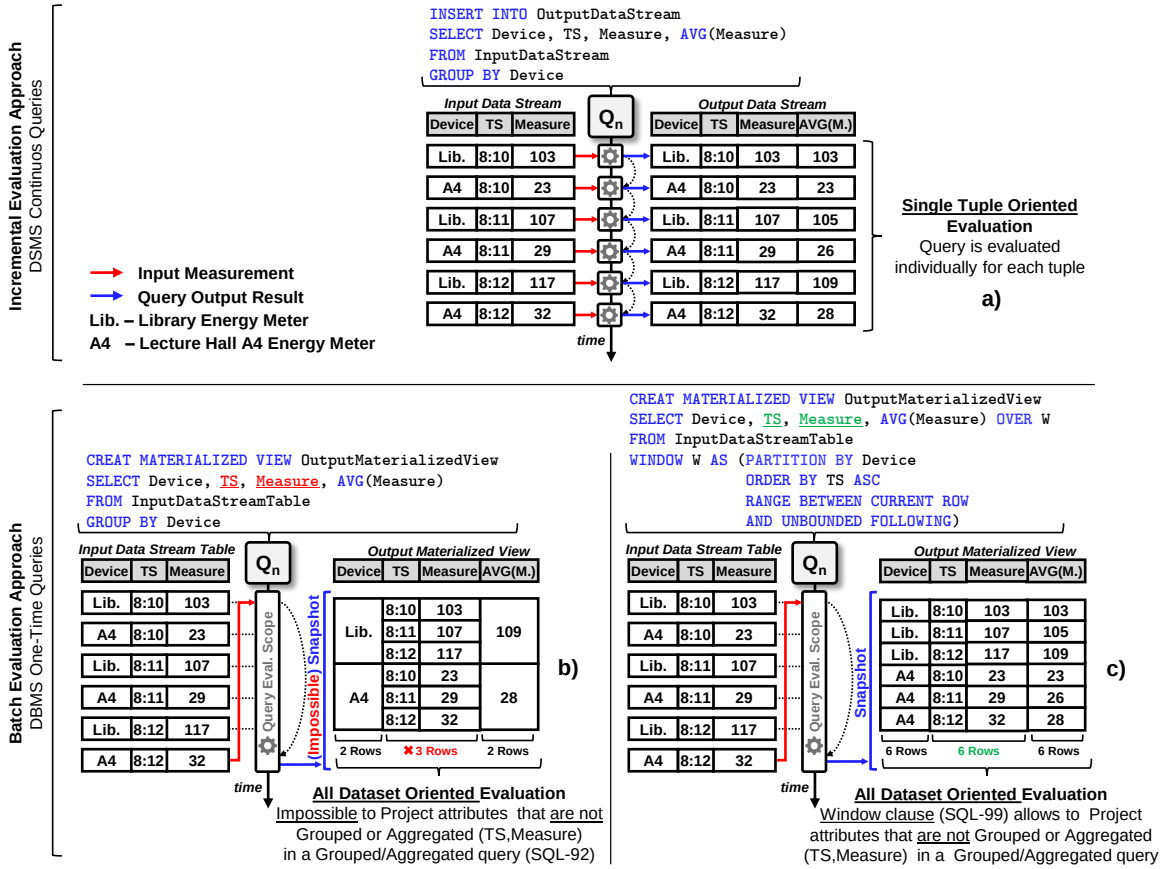

Fig. 6. Query evaluation model of DSMS and DBMS engines. DSMS Queries are evaluated incrementally, following a tuple oriented approach, where each tupled is evaluated individually by the query (a). For DBMS queries implemented in SQL-92 it is impossible for aggregated/grouped queries to project attributes that are neither aggregated nor grouped (b). If implemented in SQL-99, `WINDOW` clause makes it possible for aggregated/grouped queries to project attributes that are neither aggregated nor grouped (c).

in the DBMS using this novel operator. However, although the former issue has been solved, the query implementation is quite more complex in the DBMS than in the DSMS (e.g. see the implementation[10] of Q3 and Q16 in both solutions).

To conclude, the incremental evaluation approach that features the evaluation process of DSMS queries appears the most suitable one to implement and execute our use-case queries on monitoring energy metering networks.

The *lesson learned* that comes from these two results is that: the EPL query language provided by the DSMS is more effective and efficient for writing and evaluating queries on the domain of real-time monitoring of energy metering networks, than the SQL query language provided by the DBMS. Therefore, EMSs requiring a continuous evaluation of energy metering data must be supported by a DSMS. For further analysis, the implementation of all use-case queries in both versions of the solution is available at GitHub [10].

VI. Performance Evaluation

We aim to demonstrate the ability of the DSMS solution to monitor an energy metering network in real-time, and the failure of the DBMS solution on trying to do so. To validate this statement, the use-case scenarios were executed on both prototype versions of the Data Processing Tier, being tracked the following performance metrics:

Queue Size and Waiting Time. The quantity of queued measurements waiting to be processed and the time each measurement has to wait in the queue to be processed.

Latency of Scenario Evaluation. The time the scenario is taking to evaluate a measurement taken from the queue. That is, the time a measurement is taking to traverse the pipeline of queries composing the scenario.

Quantity of Processed Tuples. The amount of tuples that were processed so far by the scenario. Recall that each energy meter measurement is composed by three tuples (the readings of the 3-phase current), meaning that each processed measurement counts as three processed tuples.

By measuring these three metrics in each solution, we aim to evaluating how the *scenario latency* varies according to the *quantity of tuples that were already processed*, and determine the impact of this in the *size and waiting time of the queue*.

The *scenario evaluation latency* will tend to grow as a function of the *quantity of tuples processed along the test*. At least in the DBMS solution, that has to persist the arriving measurements in order to process them, continuously growing the dataset over which the queries must be executed. Regarding the *queue size*, it will remain around zero as long as the scenario latency remains lower than the *average inter-arrival period of new measurements*, which we denote by P. However,

[10] https://github.com/diogo-gsa/UseCaseQueriesImpl

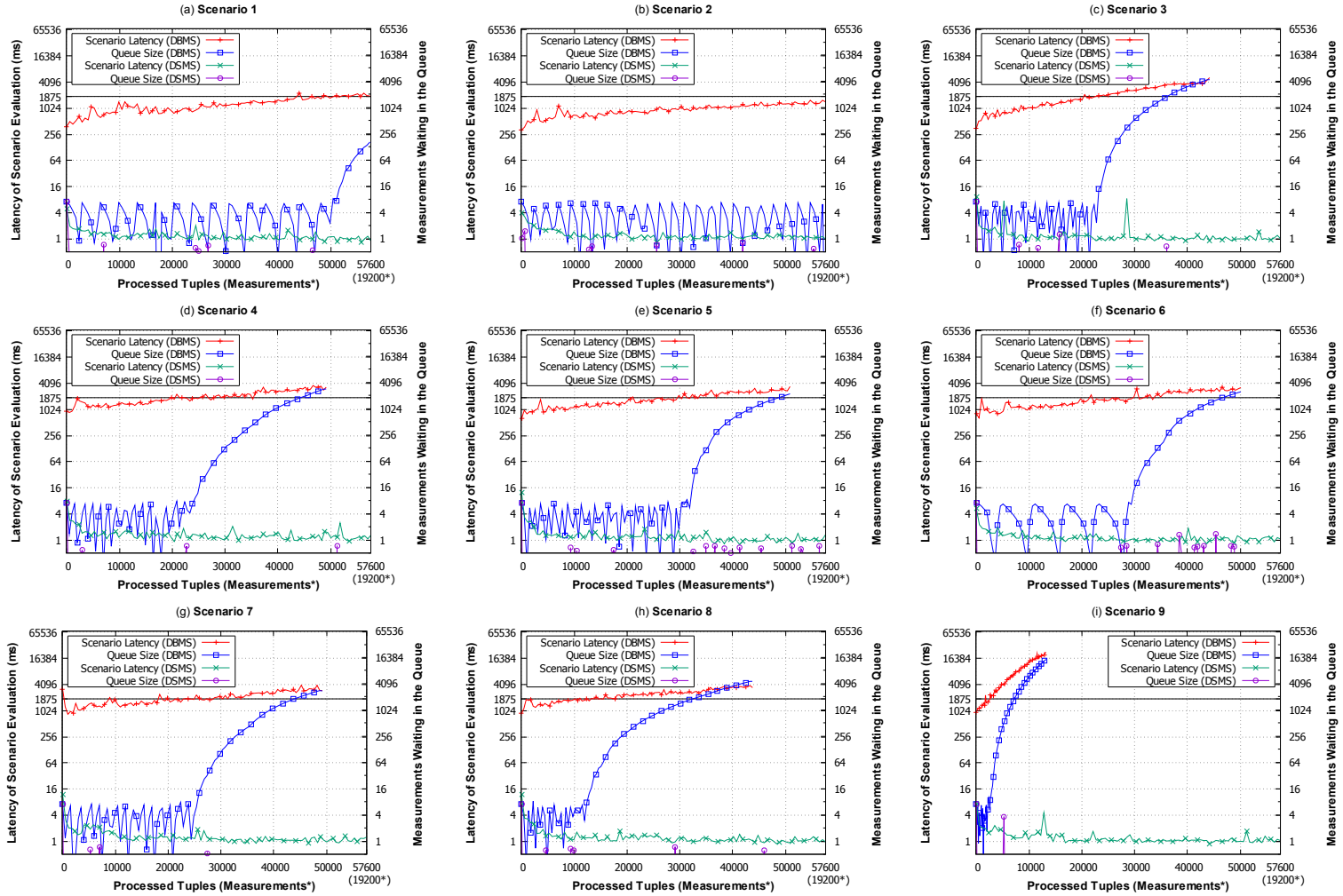

Fig. 7. Performance evaluation results for use-case scenarios 1–6 (part 1 of 2). Each plot shows, for both solutions, how the scenario latency $T_{Scenario(\varepsilon)}$ (left axis) and queue size (right axis) varies according to the quantity of tuples processed along the test (horizontal axis). 1875ms is the average arrival period of a new measurement to the system, and (*) the quantity of processed measurements is 1/3 of the quantity of processed tuples (i.e. 19200 measurements = 57600 tuples). To improve readability, the plotted values were smoothed with a Spline function.

if this inequality is reversed (i.e. $ScenarioLatency > P$) the amount of measurements waiting in the queue to be processed will start to grow infinitely, since the quantity of measurements that is being received is bigger than the one the system can process per unit of time [28].

A. Methodology of the Experiments

The evaluation was conducted by running all the *9 use-case scenarios* in both prototypes of the Data Processing Tier, performing a total of *18 tests*. Each test was executed individually, being each scenario deployed and evaluated one at a time. The building energy metering network introduced in Section III-B was used to produce the input dataset.

Each test ran for about *10 hours*, along which the simulator continuously delivered energy metering data produced by the *8 energy meters*, each one with a frequency of *4 measurements/minute*. Along these 10 hour test, the simulator gradually pushed into the Data Processing Tier a total of **19200 measurements**, i.e. **57600 tuples**, to be processed by the respective scenario under evaluation. Thus, each new measurement was pushed according to an average period of $P =$**1.875 secs.**.

B. Resource Allocation Fairness

For the sake of fairness in the benchmarking process, we ensure that both prototypes of the Data Processing Tier were evaluated with the same computational resources, namely: the same amount of memory and CPU capacity. Therefore, a limit of 512MB was defined as the maximum amount of memory available for each solution prototype, being this value the one that maximizes the performance of the DBMS solution[11]. According to the CPU usage, each test was executed in a machine solely dedicated to this purpose, meaning that both prototypes were equally limited by the maximum capacity of the CPU.

C. Experimental Environment

The experiments were conducted in a PC equipped with off-the-shelf hardware. The specifications of the machine are an Intel Core i5-3317U[12] processor running at 2.6GHz (3MB cache), 8GB of RAM (DDR3), and a 500GB HDD (5400rpm,

[11] http://www.postgresql.org/docs/9.3/static/runtime-config-resource.html (see `shared buffers`)

[12] 2 cores, 4 threads

S-ATAII). Running Windows 7 Ultimate 64-bit (kernel version 6.1.7601.18409). The database was placed in the HDD and the Java components of the Data Processing Tier prototypes were executed in the Java HotSpot(TM) Client VM (build 24.51-b03, mixed mode).

D. Results of the Experiments

The assessment of the performance metrics identified above—Scenario Latency, Queue Size, and Processed Tuples—are depicted by Fig.7. They show the variation of the *scenario latency* (left axis) and *queue size* (right axis) according to the *quantity of tuples already processed* [13] along the 10 hours duration of each test (horizontal axis).

It is important to analyse these metrics since the total amount of time it takes to process a measurement ε is given by: $T_{Process(\varepsilon)} = T_{Queue(\varepsilon)} + T_{Scenario(\varepsilon)}$ Being $T_{Queue(\varepsilon)}$ the time that ε has to wait in the queue to be evaluated, and $T_{Scenario(\varepsilon)}$ the time it takes to evaluate ε according to the respective scenario (that is, the *scenario latency*). Therefore, the Data Processing Tier is able to process energy metering data in real-time if and only if: $\forall \varepsilon \in QueuedMeasurements$: $T_{Queue(\varepsilon)} + T_{Scenario(\varepsilon)} \leq Real\text{-}Time\,Threshold$
We denote $Real\text{-}Time\,Threshold$ as the least demanding deadline that a system has to meet to be capable of real-time data processing, and it assumes the value of **5 minutes** [1].

As discussed before, the Data Processing Tier becomes unstable if at any point of the experiments we verify the condition $T_{Scenario(\varepsilon)} > 1,875\,secs.$. In the DBMS solution such state of instability was reached during the evaluation of scenarios 1 and 3—9; while the DSMS solution was able to evaluate all nine scenarios without spoil the system stability, see Fig.7. The consequence of such instability for the DBMS solution was its incapacity to process all the 19200 measurements (57600 tuples) produced along each test, within the 10 hours period. Having been left in the queue the measurements that were not timely processed, see Fig. 8 (left). In contrast, the DSMS solution was capable of process all the data produced by the simulator, since there was no measurements left in the queue at the end of each test, see Fig. 8 (right). These results tell us that, in the DBMS solution the $T_{Scenario(\varepsilon)}$ at instant t is directly affected by the amount of tuples that were processed until t (i.e. persisted in the database); whereas in the DSMS solution such relationship does not exist. This means that eventually, as the DBMS solution makes progress in the quantity of processed data, the $T_{Scenario(\varepsilon)}$ will increase until become greater than $1,875 secs.$, which will make the queue grow infinitely. Hence the sudden growth of the queue in the evaluation of scenarios 1 and 3–9; by contrast, the queue of the DSMS solution never leaves its steady state, which tends to zero.

The ability to manage the size of the queue is a critical issue because of the $T_{Queue(\varepsilon)}$ value, which is essential for a system to respond in a timely manner, that will naturally increase as the queue increases. Therefore, if along an

[13]Recall that: $ProcessedMeasurements = ProcessedTuples/3$

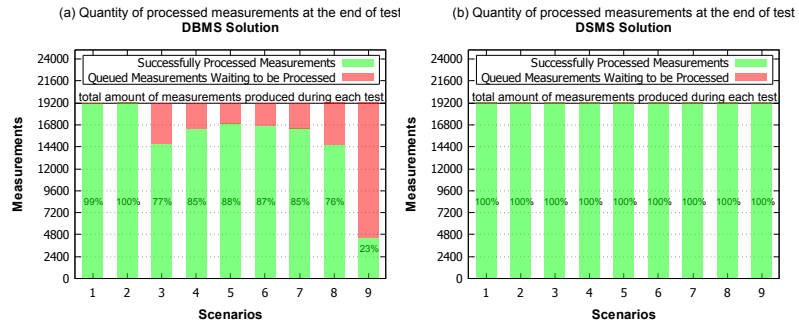

Fig. 8. Quantity of processed measurements at the end of each test. Percentage of the test dataset successfully processed (from a total of 19.2k measurements) at the end of each tested scenario (i.e. along 10 hours). In DBMS solution (left), there was a significant quantity of tests were was not possible to process all the measurements within the period of 10 hours, see scenarios 1 and 3–9. In the DSMS solution (right), all the data, of all nine scenarios under evaluation, was successfully processed within the 10 hours period of each test.

experiment the growth of the queue leads to $T_{Queue(\varepsilon)} > Real\text{-}Time\,Threshold$, then we may conclude that, from this point of the test, the system is no longer capable of process data in real-time. Such results are depicted by Fig. 9, they show how long each measurement had to wait in the queue to be processed. DBMS solution was not capable of process in real-time all the energy metering data that was produced along the evaluation of scenarios 1 and 3–9, since at a given point of each experiment the condition $T_{Queue(\varepsilon)} > 5\,mins.$ becomes true. On the other hand, the DSMS solution was capable of process all the data in real-time, since the condition $T_{Queue(\varepsilon)} + T_{Scenario(\varepsilon)} \ll 5\,mins.$ was verified along the execution of all nine scenarios.

To conclude, the DBMS solution fails to timely process eight of the nine scenarios under evaluation, the performance of the system does not scale with the increased amount of processed data (i.e. stored in database), and therefore the *DBMS solution is not capable of process energy metering data in real-time*. Regarding the DSMS solution, the system successfully processed all the data of the nine scenarios under evaluation, its performance was not affected—at all—by the increased amount of processed data, and therefore the *DSMS solution is capable of process energy metering data in real-time*.

VII. Conclusion

EMSs are used to support the decision making process of energy building managers, helping them to actuate in order to use energy in a more efficient way. To achieve this, those systems monitor buildings energy consumption through in order to identify potential problems and assess how taken actions affect energy efficiency. Effective problem solving requires early interventions, only possible with an early detection of problems. Typically, a problem takes days or weeks to be detected, reducing this time to hours, or even minutes, would be a major contribution. However, to achieve this EMSs should be able to detect volatile and ephemeral situations, which, in a real scenario, requires the continuously gathering of energy related data, that also must be continuously

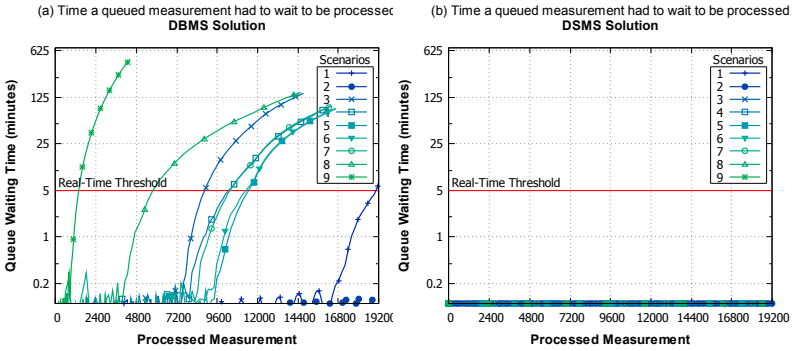

Fig. 9. Time that a measurement had to wait in the queue to be processed ($T_{Queue(\varepsilon)}$). "Real-Time Threshold" (of 5 minutes) is the least demanding deadline that a system has to met to be responding in real-time. In the DBMS solution (left), it was not possible to maintain the queue waiting time bellow 5 minutes along the evaluation of scenarios 1 and 3–9; while in the DSMS solution (right), all tested scenarios were capable of maintain the queue waiting time always below the threshold.

evaluated in a timely manner. Since as we discuss, an EMS supported by a DBMS is not the best solution to timely monitor a network of energy meters, which led us to propose an EMS supported by a DSMS as a more appropriate solution. This work validated the hypothesis that: an EMS based on a DSMS performs better than the state-of-the-art DBMS based solutions, by being capable of: *(i)* process energy metering data streams in real-time and *(ii)* provide a more suitable query language to cope with the requirements of this application domain. Our methodology was to introduce a new EMS's Data Processing Architecture, that has the novelty of being supported by a DSMS, and which we proved to have a superior performance by proceeding as follows. Two prototypes of the proposed architecture were implemented: one supported by a DSMS, representing our proposed solution; and another supported by a DBMS, representing a state-of-the-art based solution. The performance of both solutions was assessed through a benchmark evaluation, that demonstrated both the implementation feasibility of the proposed architecture, as well its superior performance over to the state-of-the-art based solution.

Acknowledgements

This work was supported by *Fundação para a Ciência e Tecnologia* (FCT) under the research project DATASTORM, EXCL/EEI-ESS/0257/2012 and PEst-OE/EEI/LA0021/2013.

References

[1] D. Anjos, P. Carreira, and A. P. Francisco, "Real-time integration of building energy data," in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 250–257.

[2] L. Pérez-Lombard, J. Ortiz, and C. Pout, "A review on buildings energy consumption information," *Energy and Buildings*, vol. 40, no. 3, pp. 394–398, Jan. 2008.

[3] D. Chwieduk, "Towards sustainable-energy buildings," *Applied Energy*, vol. 76, no. 1-3, pp. 211–217, Sep. 2003.

[4] J. Granderson, M. Piette, B. Rosenblum, and et al. L. Hu, *Energy Information Handbook: Applications for Energy-Efficient Building Operations*. Lawrence Berkeley National Laboratory, LBNL-5272E., 2011.

[5] L. Copin, H. Rey, X. Vasques, A. Laurent, and M. Teisseire, "Intelligent Energy Data Warehouse: What Challenges?" *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, pp. 337–342, Oct. 2010.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[7] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM Sigmod Record*, vol. 32, no. 2, pp. 5–14, 2003.

[8] M. A. Beyer and D. Laney, "The importance of big data: a definition," *Stamford, CT: Gartner*, 2012.

[9] A. H. Kazmi, M. J. O'grady, D. T. Delaney, A. G. Ruzzelli, and G. M. P. O'hare, "A review of wireless-sensor-network-enabled building energy management systems," *ACM Trans. Sen. Netw.*, vol. 10, no. 4, pp. 66:1–66:43, 2014.

[10] X. Ma, R. Cui, Y. Sun, C. Peng, and Z. Wu, "Supervisory and Energy Management System of large public buildings," *2010 IEEE International Conference on Mechatronics and Automation*, pp. 928–933, Aug. 2010.

[11] J. Granderson, M. Piette, G. Ghatikar, and P. Price, *Building Energy Information Systems: State of the Technology and User Case Studies*. Lawrence Berkeley National Laboratory, LBNL-2899E., 2009, no. November.

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002, pp. 1–16.

[13] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[14] B. Chandramouli, M. Ali, J. Goldstein, B. Sezgin, and B. S. Raman, "Data stream management systems for computational finance," *Computer*, no. 12, pp. 45–52, 2010.

[15] A. Mukherjee, P. Diwan, P. Bhattacharjee, D. Mukherjee, and P. Misra, "Capital market surveillance using stream processing," in *Computer Technology and Development (ICCTD), 2010 2nd International Conference on*. IEEE, 2010, pp. 577–582.

[16] N. Akhtar and F. H. Siddiqui, "UDP packet monitoring with stanford data stream manager," in *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*. Ieee, Jun. 2011, pp. 533–537.

[17] X. Jiang, S. Yoo, and J. Choi, "Dsms in ubiquitous-healthcare: A borealis-based heart rate variability monitor," in *Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on*, vol. 4. IEEE, 2011, pp. 2144–2147.

[18] Q. Zhang, C. Pang, S. Mcbride, D. Hansen, C. Cheung, and M. Steyn, "Towards health data stream analytics," in *Complex Medical Engineering (CME), 2010 IEEE/ICME International Conference on*. IEEE, 2010, pp. 282–287.

[19] X. Li, B. Plale, N. Vijayakumar, R. Ramachandran, S. Graves, and H. Conover, "Real-time storm detection and weather forecast activation through data mining and events processing," *Earth Science Informatics*, vol. 1, no. 2, pp. 49–57, 2008.

[20] P. Vassiliadis and A. Simitsis, "Near Real Time ETL," *Springer journal Annals of Information Systems*, vol. 3, pp. 1–38, 2008.

[21] R. M. Bruckner, B. List, and J. Schiefer, "Striving towards Near Real-Time Data Integration for Data Warehouses," in *DaWak*, 2002, pp. 317–326.

[22] A. Karakasidis, P. Vassiliadis, and E. Pitoura, "ETL queues for active data warehousing," *Proceedings of the 2nd international workshop on Information quality in information systems - IQIS '05*, p. 28, 2005.

[23] M. Nguyen and A. Min, "Zero-Latency Dara Warehousing for Heterogeneous Data Sources and Continuous Data Streams."

[24] P. Vassiliadis, "A Survey of Extract Transform Load Technology," *International Journal of Data Warehousing & Mining*, vol. 5, no. September, pp. 1–27, 2009.

[25] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Mobile Data Management*. Springer, 2001, pp. 3–14.

[26] C. Cranor, T. Johnson, and O. Spataschek, "Gigascope : A Stream Database for Network Applications," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM, 2003, pp. 647–651.

[27] S. Madden and M. J. Franklin, "Fjording the Stream : An Architecture for Queries over Streaming Sensor Data," in *Data Engineering, 2002. Proceedings. 18th IEEE International Conference on*, 2002, pp. 555—-566.

[28] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.