Online and Scalable Data Validation in Advanced Metering Infrastructures

Vincenzo Gulisano
Chalmers University of Technology
Gothenburg, Sweden
vinmas@chalmers.se

Magnus Almgren
Chalmers University of Technology
Gothenburg, Sweden
almgren@chalmers.se

Marina Papatriantafilou
Chalmers University of Technology
Gothenburg, Sweden
ptrianta@chalmers.se

***Abstract*—The shift from traditional to cyber-physical grids involves the deployment of Advanced Metering Infrastructures, networks of communication-enabled devices remotely controlled by utilities. Live information collected by these devices enables for applications such as demand/response, real-time pricing or intrusion detection, among others. In these scenarios, data validation is necessary in order to preprocess the noisy and lossy data produced by the devices and make it available to utilities' or third parties' applications. Challenges proper of data validation in this domain include the possibility of expressing validation rules specific to an Advanced Metering Infrastructure installation and analysis techniques that cope with the large and fluctuating volume of data produced by the devices.**

In this paper, we discuss and provide evidence of the online, scalable and expressive validation analysis enabled by the data streaming processing paradigm. Based on a prototype implementation on top of the Storm processing engine and using data from a real-world Advanced Metering Infrastructure, we show that streaming-based validation rules enable for the analysis of thousands of meters per second and only incur in small latency penalties in the order of milliseconds.

Keywords*—*Advanced Metering Infrastructures, Data Validation, Data Streaming.

I. Introduction

The transition from traditional to cyber-physical electric grids involves the deployment of Advanced Metering Infrastructures (AMIs). An AMI is composed by networks of communication-enabled devices that share information (e.g., energy consumption readings, energy quality measurements or power outage logs) with the utility's head-end. A considerable number of research directions surrounds AMIs: real-time pricing [3], demand-response [16], users' privacy [25], smart meters vulnerabilities [10], Intrusion Detection Systems (IDSs) [5], users' awareness and social media [19] or load forecast [12]. All these research fields depend on the data produced by AMIs' devices. Unfortunately, such data is known to be noisy, lossy and to be possibly delivered out of order and with duplicates (especially for AMIs relying on wireless communication [17]). Because of this, data validation analysis is adopted to preprocess and clean the data collected from the devices before the utility or third parties access it. Such validation analysis usually relies on a set of *validation rules*. It should be noted that noisy and missing data does not depend uniquely on AMIs' devices themselves, but can also be caused by their (possibly malicious) users. Causes of noisy and lossy data include faulty or badly calibrated devices, lossy or overloaded (or possibly jammed) communication channels or incorrect energy consumption readings manipulated by malicious users, among others.

Challenges: Millions of messages are generated on a daily basis by AMIs' devices. Some of them are generated with a certain periodicity chosen by the utility (e.g., energy consumption readings) while others can appear in bursts over time (e.g., power outage logs). To this end, a scalable validation analysis is required in order to cope with the large and fluctuating volume of data produced by the devices. At the same time, an online (i.e., in a real-time fashion) validation analysis is required for live information to be leveraged in scenarios such as real-time pricing or defense frameworks. Since AMIs are composed by heterogeneous types and brands of devices (e.g., electricity meters, meter concentrator units, heating meters, and so on) that usually rely on proprietary data formats, it is desirable for utilities to rely on validation tools with which system experts can easily compose validation rules rather than rely on a set of predefined ones.

Contributions: In this paper, we propose and provide evidence about the online, scalable and expressive validation analysis enabled by the data streaming processing paradigm. The latter has been proposed as an alternative to the traditional "store-then-process" (database) paradigm by applications demanding for high processing capacity with low processing latency guarantees (e.g., financial markets analysis [22], publicity pricing [4], fraud detection [15] or defense frameworks [7]). In data streaming, *continuous queries* are defined as Directed Acyclic Graphs (DAGs) of operators and run in a distributed and parallel fashion by Stream Processing Engines (SPEs) such as Storm [23], Yahoo S4 [26] or StreamBase [24]. As we will discuss, *streaming-based* validation rules can be composed by means of the standard data streaming operators provided by these SPEs. We provide the following contributions:

1) An analysis of the expressive and scalable validation enabled by the data streaming processing paradigm, including examples of real-world validation rules.
2) An implementation of a set of streaming-based validation rules on top of Storm [23], a state of the art SPE used in companies such as Twitter.
3) An evaluation of the performance and scalability of such streaming-based validation rules based on data extracted from a real-world AMI.

The rest of the paper is organized as follows. We introduce some preliminary concepts in Section II. We present how data

streaming operators can be used to compose streaming-based validation rules in Section III. We provide an evaluation of the performance (in terms of throughput and latency) of a set of validation rules in Section IV. Section V discussed the related work while Section VI concludes the paper.

II. System Model

A. Validation Analysis in Advanced Metering Infrastructures

AMIs networks are composed by heterogeneous devices such as smart meters (in charge of forwarding readings about electricity, gas or water consumption) and meter concentrators units (in charge of collecting consumption readings to forward them to the utility head-end). These devices are usually resource-constrained and organized in different network topologies (e.g., point-to-point, hierarchical or mesh ones). For these reasons, it is more desirable for a utility to be able to compose validation rules rather than rely on a predefined set of validation rules that might not fit local constraints (or rely on ad hoc solutions that will be hard to maintain). To this end, data streaming operators constitute excellent building blocks to compose streaming-based validation rules, as we discuss in Section III.

The large and fluctuating volume of data produced by AMIs' devices demand for a scalable and online validation analysis in order for the data to be leveraged by applications such as real-time pricing or defense frameworks. Validation rules could be deployed in different ways within an AMI. On one hand, they could be deployed at the utility head-end system. This centralized approach is the common choice in existing AMIs (e.g., for IDSs, as discussed in [13]). Data collected from the devices is preprocessed as it enters the utility's head-end and subsequently stored in order to be available to other applications. On the other hand, the deployment could push the validation analysis to the meters themselves (e.g., discarding wrong consumption readings at the meters rather than at the head-end). As discussed in the context of COUGAR [6], one of the pioneer SPEs, the performance of a given traffic analysis technique can be improved by keeping the data moved across devices to its minimum (i.e., transferring only useful information). Intermediate solutions could rely both on the devices themselves and the utility's head-end to perform the validation analysis. To our advantage, SPEs would allow for streaming-based validation rules to be executed at arbitrary numbers of heterogeneous nodes, thus leveraging any of these possible deployment strategies, as we discuss later in Section III-C.

B. Meter Data Management systems and Validation, Estimation and Editing (VEE) Rules

The systems in charge of collecting and storing AMIs' data are referred to as Meter Data Management systems [14], [20]. Such systems rely on a set of Validation, Estimation and Editing (VEE) rules that are used to preprocess the information before the latter is stored in the utility's databases.

In the context of VEE rules, *Validation* refers to the purging of noisy or possibly corrupted information. As discussed in Section I, possible causes of such information degradation include faulty devices or overloaded communication channels, among others. Possible examples of validation include removal of negative consumption values or consumption values that exceed the capacity of the fuse installed in a smart meter.

Estimation refers to the ability of producing missing information. Such information could be produced by relying on interpolation methods or by methods that take into account meters' historical data. As an example, a period of missing consumption values for a given smart meter could be estimated based on its consumption as observed during the previous six months.

Editing refers to the ability of modifying historical information. The rationale is that, in scenarios such as real-time pricing, live information (e.g., energy consumption readings) is needed within specified time intervals. To comply with such time constraints, it might be preferable for the utility to rely on estimated data rather than postponing any computation until all data is available. Nevertheless, real information that arrives at the utility after having been estimated (e.g., because of out of order delivery of messages) is still more appropriate to store than the calculated estimate.

C. Data Streaming

In data streaming, incoming data is processed by means of *continuous queries* (or simply queries), DAGs where vertices represent operators and edges specify how tuples flow between them [2]. Differently from their database counterpart, such queries are not issued at a point in time but rather stand continuously to process information on the fly, updating their computation and producing results accordingly. Data streaming queries consume streams, each defined as an unbounded sequence of tuples sharing the same schema, composed by attributes $\langle ts, A_1, A_2, \ldots, A_n \rangle$. Given a tuple t, attribute $t.ts$ represents its creation timestamp at the data source while $\langle A_1, A_2, \ldots, A_n \rangle$ are application related attributes. Figure 1 presents a sample schema for tuples referring to energy consumption readings that could be produced by the smart meters deployed in an AMI. In the example, tuples are composed by attributes *ts*, the creation timestamp, *sm* the smart meter id, and *cons*, the consumption in kWh observed during the last hour.

Attribute	ts	sm	cons
Description	Creation timestamp	Meter ID	Hourly consumption (kWh)

Fig. 1: Sample schema of tuples carrying energy information readings.

Data streaming operators are distinguished into *stateless* and *stateful*. Stateless operators (e.g., Filter, Map, Union) process each tuple individually. On the other hand, stateful operators (e.g., Aggregate, Join) perform computations based on sequences of tuples. Due to the unbounded nature of streams, stateful computations are performed over *sliding windows* (or simply windows). Windows can be *time-based* (e.g., the tuples received in the last 10 minutes) or *tuple-based* (e.g., the last 20 received tuples), and are defined by parameters *Size* and *Advance*, specifying the extent of a window and the amount of information discarded each time the latter slides.

Figure 2 present a sample sequence of tuples composed by the attributes shown in Figure 1 and the evolution of a

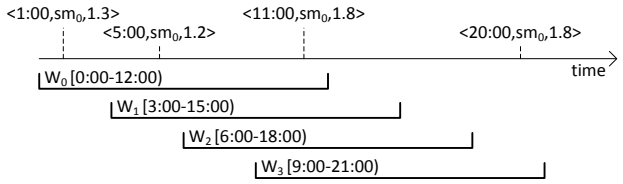


Fig. 2: Sample sequence of tuples carrying energy information readings and evolution of a time-based window of *Size* and *Advance* of 12 and 3 hours, respectively.

time-based window of size and advance of 12 and 3 hours, respectively.

III. Streaming-based validation analysis

This section discusses how data streaming can be used to run data validation analysis in AMIs. We first provide definitions and examples of a basic set of data streaming operators and continue discussing how such operators can be used to compose streaming-based validation rules. We conclude discussing how such rules are executed by SPEs at an arbitrary number of nodes, thus leveraging the possible deployment options discussed in Section II-A.

A. Basic data streaming operators

As discussed in Section II-A, it is desirable for a utility to compose validation rules specific to their AMI installation rather than using a predefined set of validation rules. By relying on the data streaming paradigm, this would be done by composing queries using the set of operators made available by a given SPE. Several relational data streaming operators are usually provided by SPEs (including Filter, Map, Union, Aggregate, Join and Sort [1]). To give evidence of the expressiveness of data streaming queries, we discuss in the following how four of these operators, usually defined by all SPEs, allow to compose validation rules: Filter, Map, Aggregate and Join. For each of these operators, we provide its definition and describe its semantic together with an example (all the examples refer to the sample schema shown in Figure 1). Each operator is defined as:

$$OP\{P_1, \ldots, P_m\}(I_1, \ldots, I_n, O_1, \ldots, O_p)$$

where OP represents the operator name, $P_1, \ldots, P_m$ represent a set of parameters that specify the operator semantics (e.g., functions used to transform input tuples, predicates used to decide which information to discard or parameters related to the windowing model), $I_1, \ldots, I_n$ a set of input streams and $O_1, \ldots, O_p$ a set of output streams. Optional parameters are defined using square brackets.

Filter: This stateless operator is a generalized selection operator that can be used to discard tuples. The operator is defined as:

$$F\{C\}(I, O)$$

where I is the input stream, O is the output stream and C is the filtering condition. Each incoming tuple is forwarded to O if the filtering condition C holds. The operator does not modify the schema of its input tuples.

The following example considers a Filter operator forwarding only input tuples referring to a consumption reading lower than or equal to 10 kWh.

$$F\{\text{cons} \leq 10\text{kWh}\}(I, O)$$

Map: This stateless operator is a generalized projection operator that can be used to transform one input tuple into one or multiple output tuples with a different schema. The operator is defined as:

$$M\{A'_1 \leftarrow f_1(t_{in}), \ldots, A'_n \leftarrow f_n(t_{in})\}(I, O)$$

where I and O represent the input and output streams, respectively. t_{in} is a generic input tuple and the attributes $\langle A'_1, \ldots, A'_n \rangle$ form the output tuples' schema.

The following example considers a Map operator converting the attribute *cons* of each input tuple from kWh to Wh.

$$M\{\text{ts} \leftarrow \text{ts}, \text{sm} \leftarrow \text{sm}, \text{cons} \leftarrow \text{cons}/1000\}(I, O)$$

Aggregate: This stateful operator is used to compute aggregation functions such $mean$, $count$, min, max, $first$ or $last$ over windows of tuples. The operator is defined as:

$$A\{\text{WType}, \text{Size}, \text{Advance}, A'_1 \leftarrow f_1(W), \ldots, A'_n \leftarrow f_n(W) \\ [, \text{Group-by=}(A_{i_1}, \ldots, A_{i_m})]\}(I, O)$$

$WType$ specifies the window type: $time$ for time-based windows or $tuples$ for tuple-based ones. Parameters *Size* and *Advance* specify the amount of tuples to be maintained and discarded each time the window slides. An output tuple carrying the result of functions $f_1(W), \ldots, f_n(W)$ is produced each time the window slides if the latter contains at least one tuple. If parameter *Group-by* is set, a separate window will be maintained for each distinct value of attributes $A_{i_1}, \ldots, A_{i_m}$. Output tuples' schema is composed by attributes ts (the timestamp of the window), *Group-by* (if defined) and attributes $A'_1, \ldots, A'_n$.

The following example considers an Aggregate operator used to compute the highest consumption reading of each smart meter over a window of 24 hours, producing a result every hour.

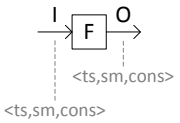
$$A\{time, 24\text{h}, 1\text{h}, \max \leftarrow max(\text{cons}), \text{Group-by=}(\text{sm})\}(I, O)$$

In the example, $WType = time$. Window $Size$ and $Advance$ parameters are set to 24 hours and 1 hour, respectively. That is, an output tuple will be produced every hour, and will contain the highest consumption reading observed during the last 24 hours for each meter separately. By dropping the *Group-by* attribute, we would instead get the highest consumption reading among all meters. The schema associated to the output stream is composed by attributes $\langle \text{ts}, \text{sm}, \max \rangle$.

Join: This stateful operator is used to match tuples from two distinct input streams, referred to as left (l) and right (r). An output tuple whose schema is the combination of the l and the r schema is produced each time a predicate P holds for a pair of tuples from the l and r streams. Similarly to the Aggregate operator, the Join uses windows to maintain input tuples. A separate window (time or tuple based) is maintained for the l and r streams. The operator is defined as:

$$J\{P, WType, Size\}(S_l, S_r, O)$$

Query I: Validation rule V_1. Discard energy consumption readings referring to negative consumption values or consumption values exceeding a given threshold, here set to 10 kWh.



$F\{\text{cons} > 0 \wedge \text{cons} \leq 10\text{kWh}\}(I, O)$

The following example considers a Join operator that matches two tuples if they refer to the same energy consumption values and are observed during the last hour.

$$J\{l.cons = r.cons, time, 1h\}(S_l, S_r, O)$$

Since the operator combines the schema of each pair of l and r tuples that match the predicate, the output tuples will be composed by attributes $\langle$l.ts, l.sm, l.cons, r.ts, r.sm, r.cons$\rangle$

B. Composing streaming-based validation queries

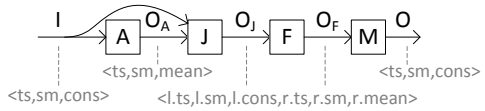
As discussed in Section II-B, data validation rules are used both for removing incoming data that should not be persisted and to estimate missing one. We present here sample queries that can be used to perform these two tasks.

When composing a validation rule that prevents incorrect data from being stored, a Filter operator can be used to take decisions about which input tuples to discard. Such a decision can be taken in different ways. On one hand, the decision could be taken based only on the tuple itself. As presented in the sample Query I (validation rule V_1) this can be done with a single Filter operator (F). On the other hand, the decision about which tuples to discard could be based on historical information. In general, an Aggregate, a Join and a Filter operator can be used to filter tuples based on historical information. The Aggregate operator can be used to compute the reference information used to decide which tuples to discard. Subsequently, the Join operator can be used to match each incoming tuple with its reference information. Finally, the Filter operator can be used to compare the incoming tuple and the reference information. Query II (validation rule V_2) presents a sample query that discards a tuple if its energy consumption exceeds more than two times the average observed for the same meter during the last three hours. Since the schema of the input tuples is modified by the Aggregate (A) and the Join (J) operators, a final Map operator (M) is used to convert the schema of the Filter's output tuples back to the same as the input ones.

When composing a validation rule that estimates missing information, the first task is to spot that one or multiple consumption readings are missing. This can be done by relying on an Aggregate and a Filter operator. The Aggregate operator can match any pair of consecutive tuples referring to the same smart meter. Subsequently, the Filter can by used to check if their distance in time exceeds a given threshold.

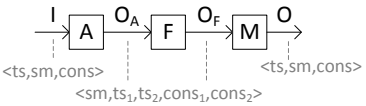
Query III presents a validation rule (V_3) that interpolates missing consumption values if the time distance between two consecutive tuples from the same meter exceeds one hour.

Query II: Validation rule V_2. Discard a meter's energy consumption readings if they exceed two times the mean consumption observed for the same meter during a given period, here set to the previous three hours.



$A\{\text{time,3h,1h,mean=}mean\text{(cons), Group-by=sm}\}(I, O_A)$
$J\{\text{time,3h,l.sm=r.sm} \wedge \text{l.ts=r.ts+3h}\}(I, O_A, O_J)$
$F\{\text{l.cons} \leq \text{r.mean} \times 2\}(O_J, O_F)$
$M\{\text{ts} \leftarrow \text{l.ts, sm} \leftarrow \text{l.sm, cons} \leftarrow \text{l.cons}\}(O_F, O)$

Query III: Validation rule V_3. Interpolate missing consumption values if the time distance between two consecutive tuples from the same meter exceeds a given threshold, here set to one hour.



$A\{\text{tuples, 2, 1, ts}_1\text{=}first\text{(ts), ts}_2\text{=}last\text{(ts), cons}_1\text{=}\ldots$
$\ldots first\text{(cons),cons}_2\text{=}last\text{(cons), Group-by=sm}\}(I, O_A)$
$F\{\text{ts}_2\text{-ts}_1 > 1\text{h}\}(O_A, O_F)$
$M\{(\text{ts} \leftarrow interp\text{(ts}_1\text{,ts}_2\text{, sm} \leftarrow \text{sm}, \ldots$
$\ldots \text{cons} \leftarrow interp\text{(cons}_1\text{,cons}_2\text{)}\}(O_F, O)$

The query is composed by three operators. The Aggregate operator (A) defines a tuple-based window of size and advance of 2 tuples and 1 tuple, respectively. A tuple containing both the first and last timestamps (ts_1 and ts_2) and consumption values ($cons_1$ and $cons_2$) is created for each smart meter (*Group-by*=sm). Subsequently, the Filter operator (F) is used to forward only such tuples whose distance between ts_1 and ts_2 exceeds one hour. Finally, the Map operator (M) is used to interpolate missing values applying the function $interp$ (which would be defined by the system expert), which will create an output tuple composed by attributes $\langle$ts, sm, cons$\rangle$ for each missing hour. An Aggregate operator could maintain historical information (similarly to Query II) to be used when estimating missing values (not shown here).

C. Scalable execution of streaming-based validation queries

A major advantage from the utility perspective is that SPEs can leverage the resources devoted to the validation analysis (e.g., nodes in a cluster or cores in a parallel architecture) independently of their deployment, as discussed in Section II-A. In this section, we further discuss how SPEs allow for such scalable execution of streaming-based validation rules.

Scalable execution is achieved by means of distributed and parallel operator execution. Distributed execution (achieved by means of *inter-operator parallelism*) allows for operators belonging to the same query to be run at different nodes. At the same time, parallel execution (achieved by means of *intra-operator parallelism*) allows for individual operators to be run in parallel at arbitrary numbers of nodes. Each operator is parallelized by deploying multiple instances of it and by partitioning its input stream(s) to such instances. The way in which

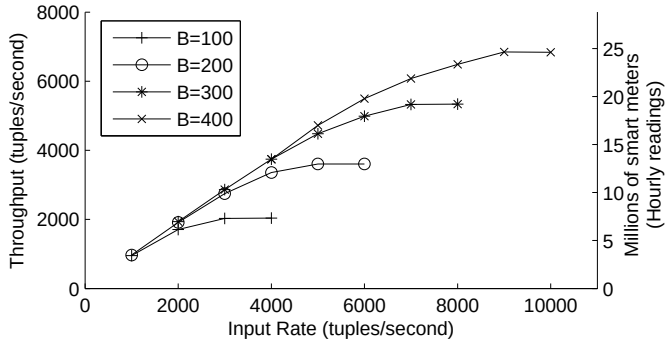


Fig. 3: Throughput (tuples/second) of validation rules V_1, V_2 and V_3 for different batch sizes.

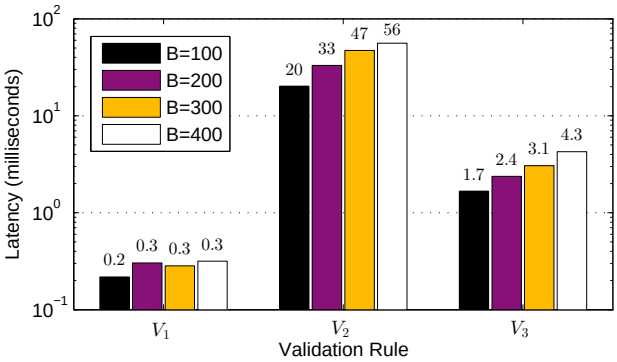


Fig. 4: Latency (milliseconds, logarithmic scale) of validation rules V_1, V_2 and V_3 for different batch sizes.

the input tuples are routed to the different instances depends on whether the parallel operator is stateless or stateful. Since stateless operators (e.g., Filter and Map) process each tuple individually, tuples can be routed to the different instances by any routing technique (e.g., round-robin). Nevertheless, the routing of tuples feeding a stateful operator must be aware of its semantics. As an example, when deploying multiple instances of the Aggregate and Join operators of Query II, all the tuples referring to the same smart meter must be routed to the same operator instance in order for the latter to produce the correct output tuple. We refer the reader to [15] for a exhaustive discussion about the parallelization of data streaming operators.

IV. Evaluation

In this section, we evaluate the applicability of the data streaming paradigm in the context of AMIs' data validation. We first provide details about the data we use and the evaluation setup. Subsequently, we evaluate the performance in terms of *throughput* and *latency* of the streaming-based validation rules discussed in Section III.

Evaluation setup

The real-world AMI used in our evaluation is composed of 300,000 smart meters that cover a metropolitan area of 450 km^2 with roughly 600,000 inhabitants. The utility extracted 13 months (From May 2012 to June 2013) of hourly consumption readings from a subset of 50 meters and made this dataset available for us. The validation rules are implemented on top of Storm, version 0.9.1. The evaluation has been conducted with an Intel-based workstation with two sockets of 8-core Xeon E5-2650 processors and 64 GB DDR3 memory. All experiments start (resp. end) with a warm-up (resp. cool-down) phase. Presented results are measured in between of these phases and are averaged over 10 separate runs. In all experiments, we process the data extracted from the AMI, modifying only the rate at which tuples are injected.

Performance evaluation

In order to make energy consumption readings available in a real-time fashion, it is important to reduce the time that goes from the measurement at the meter itself to the delivery at the utilities' or third parties' applications. Such delay depends on two main factors: the period with which energy consumption readings are pulled from or pushed by meters (plus the network latency) and the latency introduced by the data validation analysis. A trade-off exists between these two factors. On one hand, it is desirable to retrieve energy consumption readings frequently (if possible, as soon as the measurement is taken). On the other hand, it is common to batch tuples together in order for high-throughput systems to achieve better performance. In this experiment, we study the throughput and latency achieved for different batch sizes B. As it would be done in a real deployment consuming live information, all the three validation rules are executed at the same time. The throughput is measured as the rate (tuple/second) that each validation rule can sustain while latency is measured for each validation rule individually.

Figure 3 presents the processing throughput for different batch sizes of 100, 200, 300 and 400 tuples. For each batch size, the throughput initially grows linearly with the increasing input rate while it flattens down when reaching the maximum processing capacity. As expected, increasing the batch size results in higher processing throughput. For a batch size of 100 tuples, the server is able to process approximately $2,000$ messages per second. For a batch size of 400 tuples, the throughput grows to approximately $7,000$ messages per second. As shown in the figure (secondary Y axis), if meters measure energy consumption readings every hour, this processing capacity would enable us to validate almost 25 million meters.

Figure 4 presents the latency (in milliseconds), for each batch size B and validation rule (bars are plotted in logarithmic scale to better emphasize the differences). It can be noticed that different latencies are imposed by each validation rule. At the same time, an increasing latency is also observed when increasing the batch size B. The highest processing latency is imposed by validation rule V_2, since it contains both an Aggregate and a Join operator (stateful operators' computations are more expensive and thus incur higher latencies than stateless ones). However, the latency introduced by the analysis is negligible with the one introduced by transferring the data from the meters to the utility (in our AMI, this action takes a time in the order of seconds).

V. Related Work

The data streaming research field emerged around the year 2000 to overcome the limitations of traditional database

approaches for data intensive applications such as fraud detection, IDS or financial market analysis. Sensor networks sharing requirements similar to AMIs' ones played a key role in the development of pioneer SPEs such as TelegraphCQ [9], Cougar [6] or Aurora [8]. During these years, data streaming research has focused on aspects such as processing of imprecise and missing information, graceful degradation and load shedding techniques under peaks of load, and Quality-of-Service (QoS) metrics.

To the best of our knowledge, this is the first work that focuses on the expressiveness and performance of data streaming queries in the context of AMIs' data validation. Nevertheless, the data streaming processing paradigm has been taken into account in other AMI-related scenarios, including electricity load forecast [12], IDSs [11] and cloud-infrastructures [18], [21]. These works provide evidence that the data streaming processing paradigm is an appropriate candidate to address the data analysis requirements proper of AMIs.

VI. Conclusions

A considerable number of applications surrounding AMIs (e.g., demand-response, real time pricing and IDSs, among others) depend on the data produced by AMIs' devices. Its noisy and lossy nature demands for validation analysis in order to preprocess the data that is later accessed by utilities' or third parties' applications. In this paper, we have discussed how the data streaming processing paradigm can be leveraged to provide online and scalable validation analysis, composing validation rules by means of data streaming operators. Based on an implementation on top of the Storm SPE and conducted with data from a real-world AMI, we have shown that streaming-based validation rules allow for the analysis of thousands of energy consumption readings per second (with latencies in the realm of milliseconds) while relying on commodity hardware. Thanks to the distributed and parallel execution enabled by SPEs, streaming-based validation rules could also be run closer to the sources (e.g., achieving better scalability by filtering values at the sources and saving bandwidth).

Acknowledgments

This work has been partially supported by the European Commission Seventh Framework Programme (FP7/2007-2013) through the SysSec Project, under grant agreement 257007, through the FP7-SEC-285477-CRISALIS, through the collaboration framework of Chalmers Energy Area of Advance, and with support from the Swedish Energy Agency under the program Energy, IT and Design.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The International Journal on Very Large Data Bases*, 2003.

[3] H. Allcott. Rethinking real-time electricity pricing. *Resource and Energy Economics*, 2011.

[4] R. Ananthanarayanan, V. Basker, S. Das, A. Gupta, H. Jiang, T. Qiu, A. Reznichenko, D. Ryabkov, M. Singh, and S. Venkataraman. Photon: Fault-tolerant and scalable joining of continuous data streams. In *Proceedings of the international conference on Management of data*, 2013.

[5] R. Berthier and W. H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *IEEE 17th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2011.

[6] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Mobile Data Management*, 2001.

[7] M. Callau-Zori, R. Jiménez-Peris, V. Gulisano, M. Papatriantafilou, Z. Fu, and M. Patiño-Martínez. STONE: a stream-based DDoS defense framework. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[8] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams: a new class of data management applications. In *Proceedings of the 28th international conference on Very Large Data Bases*, 2002.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: continuous dataflow processing. In *Proceedings of the ACM SIGMOD international conference on Management of data*, 2003.

[10] M. Costache, V. Tudor, M. Almgren, M. Papatriantafilou, and C. Saunders. Remote control of smart meters: friend or foe? In *Computer Network Defense (EC2ND), Seventh European Conference on*, 2011.

[11] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez. Securing advanced metering infrastructure using intrusion detection system with data stream mining. In *Intelligence and Security Informatics*. Springer, 2012.

[12] J. Gama and P. P. Rodrigues. Stream-based electricity load forecast. In *Knowledge Discovery in Databases: PKDD*. Springer, 2007.

[13] D. Grochocki, J. H. Huh, R. Berthier, R. Bobba, W. H. Sanders, A. A. Cárdenas, and J. G. Jetcheva. AMI threats, intrusion detection requirements and deployment recommendations. In *Smart Grid Communications (SmartGridComm), IEEE Third International Conference on*, 2012.

[14] GTM RESEARCH. White paper, The Emergence of Meter Data Management (MDM): A Smart Grid Information Strategy Report, 2010.

[15] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez. Streamcloud: An elastic and scalable data streaming system. *Parallel and Distributed Systems, IEEE Transactions on*, 2012.

[16] Y. Guo, M. Pan, Y. Fang, and P. P. Khargonekar. Decentralized Coordination of Energy Utilization for Residential Households in the Smart Grid. *IEEE Transactions on Smart Grid*, 2012.

[17] E. Kermany, H. Mazzawi, D. Baras, Y. Naveh, and H. Michaelis. Analysis of advanced meter infrastructure data of water consumption in apartment buildings. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[18] B. Lohrmann and O. Kao. Processing smart meter data streams in the cloud. In *Innovative Smart Grid Technologies (ISGT Europe), 2nd IEEE PES International Conference and Exhibition on*, 2011.

[19] T. Mikkola, E. Bunn, P. Hurri, G. Jacucci, M. Lehtonen, M. Fitta, and S. Biza. Near real time energy monitoring for end users: Requirements and sample applications. In *Smart Grid Communications (SmartGridComm), IEEE International Conference on*, 2011.

[20] Oracle Utilities Meter Data Management. http://www.oracle.com/us/industries/utilities/046533.pdf.

[21] Y. Simmhan, B. Cao, M. Giakkoupis, and V. K. Prasanna. Adaptive rate stream processing for smart grid applications on clouds. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, 2011.

[22] M. Stonebraker, U. Çetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 2005.

[23] Storm project. http://storm.incubator.apache.org/.

[24] StreamBase. http://www.streambase.com.

[25] V. Tudor, M. Almgren, and M. Papatriantafilou. Analysis of the Impact of Data Granularity on Privacy for the Smart Grid. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, 2013.

[26] Yahoo S4. http://incubator.apache.org/s4/.