CHALMERS
UNIVERSITY OF TECHNOLOGY

eChIDNA: Continuous Data Validation in Advanced Metering Infrastructures

Citation for the original published paper (version of record):
Van Rooij, J., Swetzén, J., Gulisano, V. et al (2018). eChIDNA: Continuous Data Validation in Advanced Metering Infrastructures. 2018 IEEE International Energy Conference (ENERGYCON): 1-6. http://dx.doi.org/10.1109/ENERGYCON.2018.8398800

eChIDNA: Continuous Data Validation in Advanced Metering Infrastructures

Joris van Rooij*†, Johan Swetzén†, Vincenzo Gulisano†, Magnus Almgren† and Marina Papatriantafilou†
*Göteborg Energi, Gothenburg Sweden
joris.vanrooij@goteborgenergi.se
†Chalmers University of Technology, Gothenburg Sweden
{jorisv,vinmas,almgren,ptrianta}@chalmers.se, johan@swetzen.com

***Abstract*—New laws and regulations increase the demands for a more data-intense metering infrastructure towards more adaptive electricity networks (aka smart grids). The automatic measuring, often involving wireless communication, introduces errors both in software and during data transmission.**

These demands, as well as the large data volumes that need to be validated, present new challenges to utilities. First, measurement errors cannot be allowed to propagate to the data stored by utilities. Second, manual fixing of errors after storing is not a feasible option with increasing data volumes and decreasing lead times for new services and analysis. Third, validation is not only to be applied to current readings but also to past readings when new types of errors are discovered.

This paper addresses these issues by proposing a hybrid system, eChIDNA, utilizing both the store-then-process and the data streaming processing paradigms, enabling for high throughput, low latency distributed and parallel analysis. Validation rules are built upon this paradigm and then implemented on the state of the art Apache Storm Stream Processing Engine to assess performance. Furthermore, patterns of common errors are matched, triggering alerts as a first step towards automatic correction of errors.

The system is evaluated with production data from hundreds of thousands of smart meters. The results show a performance in the thousands messages per second realm, showing that stream processing can be used to validate large volumes of meter data online with low processing latency, identifying common errors as they appear. The results from the pattern matching are cross-validated with system experts and show that pattern matching is a viable way to minimize time required from human operators.

I. Introduction

The development of an enhanced electricity grid has moved forward both technologically and politically in recent years. One driving force is an increased measurement rate, as a number of countries introduce mandatory hourly billing. When consumers are given the option to pay for their electricity based on the demand at that particular time of day, they gain a higher degree of control over their energy bills by facilitating energy efficiency. Shorter measurement intervals also open up new possibilities in load forecasting with the potential of tailoring electricity production to the consumption and increasing the amount of renewable energy.

These new requirements have lead to the construction of Advanced Metering Infrastructures (AMIs): networks of smart meters and data concentrator units that measure consumption and report it to the utility. These devices offer reduced computational power and can be remotely controlled.

Together with AMIs, the utility data analysis infrastructure also needs to be adjusted to accept the roughly 700 times increase in data that hourly readings give compared to monthly readings. Shorter sampling periods, such as 15 minutes, have been evaluated in a research setting and can improve load forecasting accuracy compared to hourly readings [1]. It is therefore reasonable to expect further increases in data rates.

This increasing amount of data collected by AMIs is known to be error prone, far from the clean and complete measurement series required for load forecasting or billing. Measurements can arrive out-of-order, duplicated or not at all, sometimes owing to communication losses and retransmissions particularly common in wireless infrastructures. Erroneous measurements can be caused by meters that are broken, software issues or even by malicious users looking to decrease their electricity costs by unlawful means. These errors threaten the increasing quality demanded from the meter values by new applications such as demand-response and real time pricing. Therefore, all data collected must be swiftly validated and corrected if necessary.

A. Challenges

A system aiming to solve this validation problem faces several challenges. The first challenge is to process and validate this large volume of data with low latency before storing, billing or other analysis is run on the data. Fluctuating data rates and out-of-order readings contribute to this challenge as well as erroneous measurements.

The second challenge is continuously updating the validation rules and leveraging system expert knowledge. This is needed because new errors may arise at any moment and are usually identified by a human operator at some point in time. Closing the control loop creates an iterative learning process where the human controller finds new problems and subsequently uses the validation engine to find other instances of these problems (possibly re-running the validation for previous data to find earlier occurrences of a certain error).

Finally the third challenge is to run the analysis in a distributed and parallel fashion leveraging the cumulative computational power of AMIs' smart meters and data concentrators. Data concentrators could for example validate and compare voltage levels in the network for all meters in a specific area, forwarding only validated data to the utility.

B. Related work

Since the early 2000s, the data streaming paradigm has emerged as an alternative to traditional databases when it comes to processing large amounts of data in real-time. Among the many applications are financial trading and market analysis, as well as network intrusion detection and monitoring of denial of service attacks. Traditional databases are not designed to support this kind of streaming data and in response to these new challenges, Aurora was developed as one of the first SPEs [2]. Aurora was designed to run on a single machine without the possibility of distributing the work. Building upon the functionality of Aurora, among other works, Borealis, an SPE that can scale to several machines, was developed [3]. More recently, systems like StreamCloud [4] have been presented for improved scalability. Others have been released as open source software, like Apache Flink [5], Apache Spark [6], Apache Storm [7] and S4 [8].

Stream processing in the context of AMIs has been explored since 2010. Several different applications have been found for SPEs in this area, including Intrusion Detection Systems (IDSs) [9], real-time pricing [10] and adaptive measurement rates [11]. Data validation with a simplified system model was investigated in [12]. Here we designed and implemented a system that is directly deployable at a real world utility.

Out-of-orderness is a problem where two concepts have become prevalent: punctuation [13] and variations thereof (heartbeats [14], Window ID [15], out-of-order-processing [16]) and K-slack [17]. Common for these methods is that processing is halted until missing data has arrived unless too much time has passed after which special action is needed. Here we instead implemented a custom sorting mechanism that allows for processing all arrived data even if some values are missing.

C. Contributions

In this paper we introduce eChIDNA, a *Kappa* architecture [18] that addresses these challenges relying both on the traditional store-then-process (database) paradigm and the data streaming processing paradigm. The streaming paradigm has been proposed as an alternative to the traditional store-then-process (database) paradigm by applications demanding high processing capacity and low processing latency.

A prototype of eChIDNA has been implemented, extending the system of the utility providing the real use-cases we studied. The streaming analysis is performed relying on the state of the art Apache Storm [7] Stream Processing Engine (SPE). We evaluate the system both in terms of efficiency as well as performance and the results are compared with the ground truth provided by system experts.

The rest of the paper is organized as follows: the system model is introduced in Section II. In Section III, we discuss the architecture of eChIDNA. The evaluation is presented in Section IV. Finally the conclusions are presented in Section V.

II. System model

Metering data at a utility typically goes from the smart meters in the AMI through concentrator units and the head-end before being stored and processed in a Meter Data Management system (MDM).

Table I
Format of tuples containing meter readings.

Attribute	Description
t_n	timestamp
met	meter ID
loc	location ID
cc	cumulative consumption
cd	consumption delta

A. Advanced Metering Infrastructure

AMIs are composed of a multitude of different devices; smart meters that measure the energy or water consumption, concentrators that collect the readings from the meters and forward these to the central server. The meters and concentrators in the field have limited memory and processing capabilities and can be organized in different topologies (e.g., point-to-point, hierarchical or mesh ones).

The smart meters can save the amount of consumed energy in different ways. The *cumulative consumption* since startup can be stored together with a timestamp at regular time intervals, a *consumption delta* - the amount of energy used between two timestamps - can be stored, or both. Readings are sent to the central server regularly after which they are stored in the Meter Data Management (MDM) system. The bulk of the data will have arrived within 24 hours but the server will continue waiting for data to arrive. If it has not arrived with a certain amount of days D_{max}, it is considered lost. D_{max} is typically in the order of days or weeks, yet regulations and future applications like real time pricing demand fast processing of data. Data must therefore be processed as soon as it is available even if some intermediate readings in the stream may not have arrived yet.

B. Data streaming

In the streaming data paradigm, a Stream Processing Engine (SPE) is used to process streams of data in a distributed and parallel fashion. Data is represented in tuples with a common schema, composed by a timestamp t_n and a set of attributes $\langle A_1, A_2, \ldots, A_n \rangle$. A schema for tuples with meter readings is presented in table I. The operation of an SPE relies on viewing data as a flow of such tuples from beginning to end. The incoming data therefore consists of an unbounded stream of tuples. To extract meaningful information from these tuples, continuous queries in the form of Directed Acyclic Graphs, DAGs, are formed. Edges in the DAG correspond to the flow of data between the operations, represented by vertices. Operations can be for example filters that drop tuples that do not fulfill certain criteria, mappers that transform tuples or union operations that merge several streams into one. All these are examples of stateless operations, they consider only a single tuple when producing a result. Another type of operators are the stateful ones, for example aggregate which performs a computation over several tuples in a single stream and join

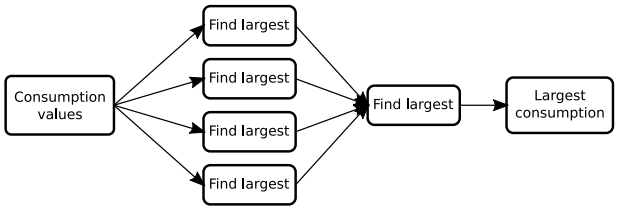

Figure 1. A topology that finds the highest consumption value, the highest number of kWh drawn for specified time period, in a stream of consumption values. The first group of "Find largest" nodes are running in parallel, possibly on different machines. The last "Find largest" needs to be a single process to summarize the results.

which combines tuples from several streams. A common way of performing aggregate operations is by using a (sliding) window. To handle the unbounded stream of tuples, only the latest tuples are considered. For example, a time-based window can contain all tuples from the last five hours and a tuple-based window can contain the 10 latest tuples.

The results of a continuous query can be alarms that are triggered or a modified data stream saved to a dedicated database. An example of a continuous query that finds the highest measured power consumption in a distributed manner can be found in figure 1.

III. Architecture

Here we describe the architecture of a system that can deal with the problems and challenges mentioned in Section I. eChIDNA consists of two modules: data ingestion and validation. The latter in turn is composed of two submodules, single- and composite event detection, as shown in figure 2. The data ingestion module takes care of the incoming real time data from the different AMI systems as well as the historical data from the MDM system. The module parses the incoming data into tuples with a uniform format in order to maximize the systems interoperability. The tuples outputted from the data ingestion module are read into the validation module. Errors identified by the single event detection are sent to the composite event detection submodule, which finds common composite errors made up by sequences of these single events.

A. Data ingestion

The data that is read into eChIDNA can either come from one or more AMI systems or, when data needs to be reprocessed, from the MDM. The data from different AMI systems, which in turn could have multiple meter brands and models, can have different formats. For example meters could send the cumulative consumption, a consumption delta for a fixed time period (eg. day, hour, 15 minutes), or both. Therefore the data ingestion module parses the data into a uniform tuple format presented in table I. Because many validation rules require a consumption delta, these are calculated by eChIDNA, if not already present, by taking two consecutive tuples and computing the difference in the cumulated consumption. Data that has been processed by the entire system can be moved away or simply removed since reprocessing can always be done from the persistent storage in the MDM.

The calculation of the consumption delta is straightforward when all tuples arrive in order, but becomes more challenging when tuples are missing or out of order. There is extensive research dealing with tuples arriving out of order, as described in Section I-B. Common for these methods is that the processing for a meter is halted until missing tuples have arrived. This makes these methods not compliant with the system requirements:

1) Data must be processed as soon as possible.
2) Data may arrive with a maximum delay D_{max}

Therefore we developed a tailored sorting mechanism described below.

A tuple $\tau.t_n$ with timestamp t_n is used to calculate the consumption delta between t_{n-1} and t_n as well as the delta between t_n and t_{n+1}. This implies that $\tau.t_n$ cannot be discarded until both $\tau.t_{n-1}$ and $\tau.t_{n+1}$ have arrived.

The sorting mechanism must thus keep track of which tuples have been fully processed and does this by using intervals. An interval spans between two timestamps and denotes that all tuples inside of the interval have been processed. An interval $[n..m]$ for example indicates that tuples with a timestamp between n and m have been processed. A tuple $\tau.t_n$ which can not attach to a existing interval will start a new interval $[n..n]$, spanning between n and n. At the arrival of $\tau.t_{n+1}$, this interval is expanded to $[n..n+1]$ while neither of the tuples is discarded yet. As $\tau.t_{n+2}$ arrives the interval is expanded to $[n..n+2]$ and $\tau.t_{n+1}$ is discarded after the consumption deltas have been calculated. This approach allows us to store only the tuples that are still waiting to be processed. In case of out-of-order tuples, multiple intervals will be present to account for the tuples that are processed. Two intervals merge when the missing tuples between them arrive and are processed. Intervals and tuples with timestamps that exceed the maximum allowed delay D_{max} are deleted.

This mechanism accumulates state during system operation, the tuples that have not been processed completely as well as the intervals themselves. Persisting the state at a regular time interval and saving the input tuples processed during this interval ensures a robust system that can cope with crashes.

B. Validation modules and rule design

A basis for further analysis, the validation rules lay the groundwork on which the system is built. eChIDNA's architecture allows for rapid implementation and testing of new validation rules, both for single- and composite events.

1) Single event detection sub-module: Single event triggers contain a check for some condition and will output an identifier if this condition is met. The identifiers are stored for every meter in a sliding window of size D_{max}. Triggers can indicate an error of some kind in the meter reading, but can also be used to find other events. These events could then be used as a building block for composite events. Some examples for both types of triggers are described below.

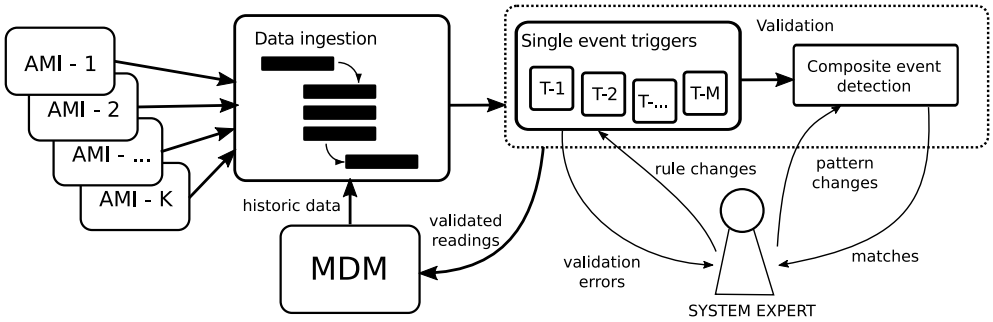

Figure 2. An overview of the eChIDNA architecture. Data from one or more AMI systems and/or historical data from the MDM are read by the data ingestion module. The validation module is composed by two submodules: The single event triggers validate individual tuples and the composite event detection that looks for patterns in the single events. Validated data is stored in the MDM, while validation errors are sent to a system expert. New validation rules and patterns can be submitted by the system expert.

Table II
SINGLE EVENT TRIGGERS AND THEIR IDENTIFIERS.

Representation	Validation error
N	Negative consumption
Z	Zero consumption
H	Above fuse level

1a) Negative values: As described in Section II-A, the system model does not allow negative consumption deltas. Such deltas therefore always indicate an error of some kind.

1b) Zero consumption: Zero consumption is not an error in itself, but can be part of a bigger problem. This trigger but can be a useful building block for finding composite errors. For example, a longer period without energy consumption may be important to look at more closely.

1c) Above fuse level: As a maximum value for electricity consumption, the fuse provides a hard limit. For every individual meter, information about the installed fuses is available and has been used to calculate this limit. Although a fuse may temporarily allow values above its rating, it cannot sustain it over time and therefore consumption above this limit triggers an alert. This alert can be caused by a technical error, but it can also be caused by a customer who has installed a fuse with higher rating. Whether intentional or not, this needs to be corrected since the network tariffs for the consumer depends on the installed fuse.

2) Composite event detection sub-module: The errors described above can be part of a larger problem. Issues with hardware or software can give rise to so-called composite errors that can be identified by considering the ordering and kind of errors that have been reported for a smart meter. Automatic identification of these errors is a first step towards automated correction, which would save substantial amounts of time for human operators. In order to identify such errors automatically, a formal pattern matching framework is needed.

One common system for pattern matching is *regular expressions*, a search pattern defined by a character string. In order to apply regular expressions to the errors caught by the validation rules, they first need to be transformed into a string that can be matched by a regular expression. There is additional information that needs to be captured apart from the string of errors, for example finding if an error occurred at a specific time of day. If a match is found that also passes through these kinds of validation, it can be considered an instance of a composite error.

The sliding window where the errors found by the validation rules are saved contains timestamp ordered errors. Since there is a maximum delay for meter values, the size of the window is bounded by D_{max}. Matching is performed every time an error is found. The patterns for composite errors have a maximum length which is used to only look for matches within a partition of the stored errors. Matching an error with length l at timestamp t_n will consider values with a timestamp between $t_{n-(l-1)}$ and $t_{n+(l-1)}$ to ensure that every possible interval of length l containing the timestamp t_n is processed.

In order to match the errors with a regular expression, representations of the errors within the matching interval are concatenated into an error string. The expression 'HN' for example, using the identifiers specified in table II will match every time a consumption delta exceeding the fuse value is followed by a negative consumption value. Matches to the regular expression are complemented with the start and end times for the match, so that additional properties, like the timestamp of a specific event, may be tested for.

Due to the fact that partitions of the stored errors that are matched by regular expressions can overlap, there is a risk of finding some composite errors several times. This is illustrated by the following example: Consider the string with three high values followed by a negative value: 'HHHN'. The expression 'H.0-2N', a high consumption value followed by a negative consumption and at most 2 values in between, will match three times for the substrings 'HHHN', 'HHN', and 'HN'. For this reason, the start and end of any matched composite error is stored. In case a match is completely covered by a previously matched error, that match is not considered.

IV. EVALUATION

eChIDNA's accuracy is evaluated and compared with results from a system expert. The efficiency is evaluated as well. The

Table III
NUMBER OF MATCHES FOR EACH VALIDATION RULE IN THOUSANDS. THE TOTAL NUMBER OF VALUES ANALYZED IS APPROXIMATELY 200 MILLION.

Validation rule	Matches
Zero consumption	4000
Negative consumption	1.5
Above fuse	5.5

Table IV
EXAMPLE OF CONSUMPTION VALUES FROM A METER WITH A RUSHING AND REVERSING PATTERN.

timestamp	18	19	20	21	22	23	24
consumption	2.1	*134.6*	*78.9*	4.7	3.8	2.7	***-204.2***

throughput measured in processed tuples per second while the latency is defined as the time between system output and the most recent tuple that caused the output. The evaluation ran with 30 days of data for the validation rules, of which 12 days were used to assess the composite event detection capabilities.

A. Evalution setup

The data that is used for eChIDNA comes from an AMI with approx. 270 000 meters, 7 000 concentrator units and a central server. The meters register the cumulative consumption every hour. Produced energy is either not measured or recorded in a different register, guaranteeing that consumption values are monotonically increasing. The meters are queried for their readings by the concentrators twice a day. The concentrators send the data to the central server for processing. The maximum delay D_{max} for this AMI is 40 days. eChIDNA taps into the data stream between the central server of the AMI and the MDM, where the data is batched and persistent on disk. The tuples contain a meterID, locationID, timestamp and cumulative consumption for every reading.

The efficiency of the system is evaluated on existing hardware available at the utility. It runs on a virtual server with two AMD Opteron processing cores at 2.6 GHz and 4GB of RAM. This server runs Apache Kafka [19] as input and message queue in the data ingestion module while the consumption deltas are calculated by the SPE Apache Storm [7]. The validation module also runs on Apache Storm. Pyleus, a framework enabling the use of Python topologies on Storm, was used in consultation with system experts for usability reasons. Storm (and Pyleus) use topologies to represent the Directed Acyclic Graphs. The vertices in the DAG are called *bolts*. Reading the persistent file data into Apache Kafka is also performed on the server.

B. Validation accuracy

The single event detection was evaluated using the data produced by smart meters during one month, out of which twelve days were used to compare the results from the composite event detection with the results from a system expert.

1) Single event detection: All single event triggers in Section III-B1 were enabled: zero and negative consumption, and above fuse. The number of errors found for each validation rule can be seen in table III. Approximately 2% of all processed values trigger a validation rule, the absolute majority of these values have zero consumption.

2) Composite event detection: To evaluate this in eChIDNA, an expression was defined in collaboration with a system expert, to identify meters suffering from a specific hardware failure common in the data from the used AMI. This failure expresses itself in the consumption values with a specific pattern: *rushing and reversing*. This pattern contains one or more extremely high consumption deltas during the day and a large negative consumption delta, that compensates the earlier large ones, at midnight. An example can be seen in table IV. The regular expression for this pattern was inputted in the system and was matched 640 times in one month of data. During a twelve day testing period, a system expert was asked to note all meters diagnosed with the hardware failure. 190 meters were identified. eChIDNA diagnosed 116 meters as experiencing this problem during the testing period, so a true positive rate of 61% was obtained for the hardware failures using the rushing and reversing pattern. The 39% of false negatives were investigated with help from the system expert and decomposed in the six cases following below. See also figure 3.

Missing data is the main cause for the false negative meters, the hardware failure not only causes errors in the consumption values but also affects the communication between the meter and the concentrator unit. Three cases where the root cause is missing data were identified: *1: only negative*, *2: only high* and *3: reversed*. The only negative case accounts for 23% of the false negative meters. For these meters no extremely high consumption deltas were found, only the negative delta at midnight was identified. The only high case, accounting for 22%, is the opposite: No negative delta was found, only the extremely high consumption deltas. Reversed, at 14%, is a combination of the previous two cases were only the negative delta was found during one day, and only extremely high values were found the day after, ie. a reversed version of the original pattern. Another large group of false negatives, *4: end time* at 38%, did have both extremely large consumption deltas and negative deltas, but the negative deltas did not occur at midnight. Instead these deltas occured at some other hour during the day. A case with 1% of the false negatives, *5: two days*, contains meters where the correcting negative consumption delta did not occur the same day as the extremely high values but instead the day after. Finally, *6: unseen data* with the last 2% is made up of meters where the system expert used data that arrived after the twelve day test period to diagnose the meters. This data was never analyzed by eChIDNA and therefore the pattern was not matched.

This decomposition shows that the original expression was not broad enough to identify all meters experiencing the hardware failure. The knowledge gained by the system expert during the decomposition can now be used to improve the expression and the kappa architecture can be leveraged to re-validate the data when new expressions are in place. This

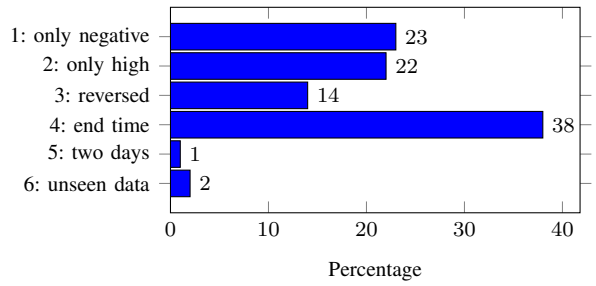

Figure 3. The six different cases for errors caused by a specific hardware failure not being matched by the original pattern.

shows the importance of system expert feedback to the system, enabling swift improvements and maximizing the validation accuracy, closing the control loop.

C. Efficiency

The *throughput* of the system is defined as the number of tuples processed per second and the maximum throughput was found by running the topology with live data and increasing the input rate step by step until the CPU usage was at 100%.

The *latency* of the system is defined as follows: The time between output from the system and the arrival of the latest tuple that triggered the output. With this definition, the highest latency observed while processing data from 270 000 smart meters, was 4 seconds.

The throughput measured was 1 500 tuples per second on hardware with the processing power of contemporary deployed or immediately-deployable equipment. This means that the system has the potential to validate 5.4 million consumption values every hour. Only 5% of this capacity is required for the validation of the 270 000 hourly readings in the system. The remaining capacity can be used to deal with possible arrival bursts or for validation of 19 days of historic data every day.

V. Conclusions

The introduction of smart meters and legislation has given rise to a greatly increasing number of processed meter values at utilities. At the same time the quality demanded of these values has been increasing as well due to new applications, driving the need for a fast and accurate validation. Validation of live production data is complicated by out of order data and fluctuating data rates as well as the appearance of new problems. These problems can be cause by updates, hardware failure or other, often hard to foresee, sources. For this reason a live architecture, with system experts in the loop, is required until full automation can take over. In this paper, we have discussed how the data streaming processing paradigm in a kappa architecture can be used to provide scalable and adaptable validation for both real time and historical data. An implementation built in Kafka and Storm, eChiDNA, shows that data for millions of meters reporting hourly values can be validated on commodity hardware, with a configurable validation rule set. These results encourage the continued efforts in this direction, with more advanced types of validation detecting deviations from known distributions, rapid changes, even patterns that may indicate malfunctioning measurements.

Acknowledgment

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the EU Horizon 2020 Framework Programme under grant agreement 773717 and by the collaboration framework of Chalmers Energy Area of Advance with Göteborg Energi and project INDEED.

References

[1] M. Ghofrani, M. Hassanzadeh, M. Etezadi-Amoli, and M. Fadali, "Smart meter based short-term load forecasting for residential customers," in *North American Power Symposium (NAPS)*, 2011, pp. 1–5.

[2] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, and others, "Monitoring streams: a new class of data management applications," in *Proceedings of the 28th international conference on Very Large Data Bases*, 2002, pp. 215–226.

[3] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, and others, "The design of the borealis stream processing engine." in *CIDR*, vol. 5, 2005, pp. 277–289.

[4] 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*, vol. 23, no. 12, pp. 2351–2365, 2012.

[5] "Apache Flink." [Online]. Available: http://flink.apache.org

[6] "Apache Spark." [Online]. Available: http://spark.apache.org

[7] "Apache Storm." [Online]. Available: http://storm.apache.org/

[8] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 170–177.

[9] V. Gulisano, M. Almgren, and M. Papatriantafilou, "METIS: a two-tier intrusion detection system for advanced metering infrastructures," in *Proceedings of the 5th international conference on Future energy systems*. ACM, 2014, pp. 211–212.

[10] B. Lohrmann and O. Kao, "Processing smart meter data streams in the cloud," in *2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe)*, 2011, pp. 1–8.

[11] 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*, ser. ScienceCloud '11, 2011, pp. 33–38.

[12] V. Gulisano, M. Almgren, and M. Papatriantafilou, "Online and scalable data validation in advanced metering infrastructures," in *Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2014 IEEE PES*, 2014, pp. 1–6.

[13] P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras, "Exploiting punctuation semantics in continuous data streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 15, no. 3, pp. 555–568, 2003.

[14] U. Srivastava and J. Widom, "Flexible time management in data stream systems," in *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2004, pp. 263–274.

[15] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "Semantics and evaluation techniques for window aggregates in data streams," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 311–322.

[16] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier, "Out-of-order processing: a new architecture for high-performance stream systems," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 274–288, 2008.

[17] S. Babu, U. Srivastava, and J. Widom, "Exploiting k-constraints to reduce memory overhead in continuous queries over data streams," *ACM Transactions on Database Systems (TODS)*, vol. 29, no. 3, pp. 545–580, 2004.

[18] "Kappa architecture," 2014. [Online]. Available: http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html

[19] J. Kreps, N. Narkhede, J. Rao, and others, "Kafka: A distributed messaging system for log processing," in *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB)*, 2011.