A Comparative Analysis of Smart Metering Data Aggregation Performance

Dejan Ilić, Stamatis Karnouskos, and Martin Wilhelm
SAP Research, Karlsruhe, Germany
Email: {dejan.ilic, stamatis.karnouskos, mar.wilhelm}@sap.com

Abstract—**In the Smart Grid era fine-grained energy information pertaining real world processes can be collected and may reveal new insights if these can be analyzed in real-time. Energy "Big Data" analytics can lead to a plethora of new innovative applications and enhance decision making processes. However, to do so, we need new enterprise tools and approaches that can take into consideration the specifics of the energy domain and offer high performance analytics on its raw data. In this work, experiments are conducted to measure the performance of the different levels of energy data aggregation. Thousands of smart meters are aggregated, by usage of the collected energy readings from a real-world trial. Using a selected dataset, the traditional database system (row-based) performance is compared to the emerging column-based approach in order to assess the suitability for real-time analytics in such scenarios.**

I. Introduction

The emergence of the Smart Grid [1] brings into foreground a modern information-centric infrastructure [2] where a new generation of innovative applications and services can be realized. Due to the prevalence of networked embedded devices [3], fine grained energy related data is expected to be collected, resulting to huge amounts of information in minutes, or even seconds, that can provide new insights. The acquisition of this "Big Data" as well as its assessment within a business specific context and extraction of information, preferably in real-time, poses a grand challenge. As the real value creation in big data is analytics [4], their high quality, effectiveness and timely delivery may lead to significant competitive business advantage.

Although there are several application domains, complex infrastructure such as the emerging smart cities can benefit from real-time analytics. There, monitoring of energy related aspects is seen as an integral aspect of several key performance indicators that are considered in decision making processes. For instance analytics on the vast amount of energy data [5] can lead to better prediction of energy customers [6] and offer new energy-related services [7] both in residential as well as in industrial [8] environments. Timely assessment and understanding may lead to qualitative better decisions and assist for instance city administrators to better run them. As an example a smart city energy cockpit providing city-wide information on energy usage and comparative analysis may enhance empower city officials to take decisions towards better energy management, CO_2 reduction, dynamic RES integration, EV charging, public infrastructure energy cost reduction, city investment planning, simulation of "what-if" scenarios, etc. To do so however, integration of multiple sources of data is needed, and subsequently highly complex processing should be applied e.g., at city, neighborhood, or even smart house or building level.

Many of the enterprise systems are relying on the Online Transaction Processing (OLTP) for their operations; however the need for high-performance analytics has given rise to separate specialized systems delivering Online Analytical Processing (OLAP). When we drill down to real-time business analytics, while also taking into consideration the drastic performance improvements of in-memory systems, using an in-memory column database has some profound implications [9]. Independent of the row vs. column comparisons [10], the performance of the (in-memory) column-based solutions gained attractiveness as one is able to efficiently work in analytical as well as transactional workload environments [11].

Today there are several open source e.g. MonetDB (www.monetdb.org) [12] and commercial e.g. [13] column-based databases. In this work we focus on smart metering data assessment and do an assessment by using a traditional DB (i.e. MySQL) including its in-memory variant, and an in-memory column-based DB (i.e. MonetDB). Our aim is to assess some aspects with respect to energy measurement aggregations by using out-of-the box existing DBs without really diving deep to their tuning which could yell some additional performance benefits.

II. Data Processing

Several experiments were carried out measuring the aggregation performance of smart metering data. For all of them, we have used a real-world dataset acquired during the trials of the NOBEL (www.ict-nobel.eu) project in 2012. The data has been collected by an enterprise Integration and Energy Management system (IEM) [7] and contains the cumulative time series of smart meter energy readings. The meters have an energy resolution of 1 kWh that is sampled in resolution from 15 minutes to one hour (depending on the meter). Since the resolution is 1 kWh many samples were constant and therefore were removed from the raw data set to reduce the overall amount. This has resulted in having measurements with different time distances and are not available for every hour. As aggregation of time series data needs regular time resolutions, this implies that we need to use interpolation in order to provide data that fulfills this requirement. Their removal not only reduced the size of the set, but as such the precision is improved after an interpolation step.

Two different approaches for executing the data aggregation step are considered for this work i.e.,

i) Interpolating the raw smart meter data of a specific group and subsequently aggregating it: here the advantage is flexibility on the required sampling resolution, since the interpolation is done during runtime. Although, this approach leads to reduced usage of storage, the disadvantage is that the individual smart meter interpolation is done during runtime (for a selected group) that possibly can lead to lower performance.

ii) Use pre-interpolated data and only execute the aggregation during runtime: the advantage may lie in skipping the individual interpolation within a group, but this approach requires much higher storage. Additionally, the flexibility is constrained by fixing the time resolution of the interpolation. However, the aggregation simplicity in runtime of the fixed resolution is expected to result in better performance.

To clearly depict the difference between traditional and emerging tools, we have used two open source DBs i.e. MySQL which represents the traditional row-based domain and the MonetDB representing the emerging column-based world. MySQL can store smart metering data on the hard drive and in memory, while the MonetDB by default stores all data in memory. To show the benefits of the in memory storage over the traditional solution InnoDB (hard drive) and in-memory were compared (both are part of MySQL). The selection of the InnoDB engine was due its caching algorithm of the frequently accessed data. As cached data is kept in memory, time to access the data can be reduced, but it still may require to access some data located on the hard disk. The in-memory storage engine by contrast, stores all the data completely in memory and no access on the hard disk is required.

Based on the need for interpolating the raw data set and the investigation of the smart meter grouping behavior, a stored procedure for group interpolation is implemented for the traditional solution. This procedure is invoked by a thin client for a specified group (of numerous smart meters), the time frame and the resolution of the interpolated series. The boundary points, before the first and after the last data point, are also required in order to calculate the entire interpolation time frame. For simplicity reasons the algorithm is implemented as the linear interpolation method. As such, within the group interpolation procedure, every single device of the group is interpolated individually and subsequently all the smart meters are aggregated to a single time series. The experiments conducted in section IV will help us understand why the interpolation stored procedure was required, and how it differs from a distinctive feature of column stores that can apply aggressive data compression.

III. Experimental Environment

Our original dataset consists of 5032 different smart meter devices and more than 3 million unique meter readings. For the experiments, two subsets were created from the original dataset; the first subset contains the data for only one month, whereas the second subset is complete with about six months of smart metering data. A detailed overview of the created datasets is provided in Table I.

Table I
Overview of the two data sets used in experiments

	Set A (reduced)	Set B (complete)
Device count	4 020	4 382
Days	30 (1 month)	170 ($\approx$ 6 months)
Time resolution	15 minutes	15 minutes
Sample count	537 604	3 365 627
Sample %	4.6%	4.7 %
Samples per device	2 880	16 332
Interpolated count	11 581 620	71 571 206

As one can note in the table, the 15 minute interpolation will significantly increase the total number of points. The original sets will be referred to as A and B, while pre-interpolated sets will be noted as A' and B'. Furthermore, the set distributions of the meter readings play a significant role in understanding the experimental results. As duplicates and constant values of energy readings were removed from the set beforehand, the final set description was calculated based on percentage of meter readings present in the set. More real meter readings available in the set will result in less interpolated points (as seen in section II). The set density of the calculated percentage values are presented in Figure 1.

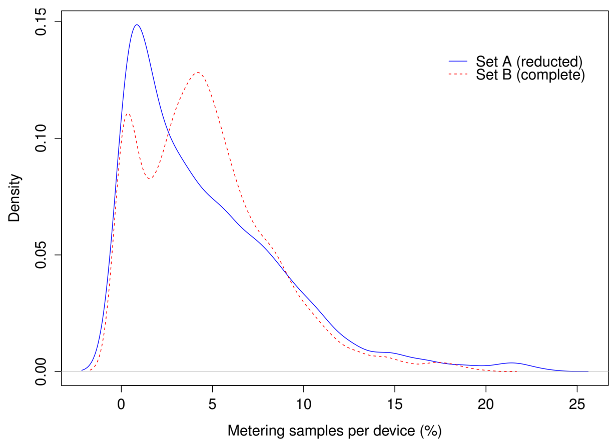

Figure 1. Density of percentage of meter readings in the complete set

The described datasets (see Table I) are used for the experimental purposes with both DBMS. In MySQL, the B-Tree index for the device and timestamp columns was assigned to the database table. The B-Tree index is appropriate for the aggregation queries because they make use of bigger-than and lower-than comparisons. The MonetDB creates indices managed internally by the DBMS itself with the different compression techniques executed automatically at many levels. Since the selected DBMSs for comparison were running on two different machines, certain impact of different hardware composition can be expected to influence the presented results. However, both machines featured the standard configuration of the DBMSs, thus further optimizations can be realized (but are beyond this work) in both cases which may provide better results. The machine configurations are depicted in Table II.

Table II
OVERVIEW OF THE TWO EXPERIMENT MACHINES

	Machine 1	**Machine 2**
DBMS	MySQL	MonetDB
Storage type	row store	column store
Storage engine	InnoDB/in-memory	in-memory
IO latency	local, $<$ 1 ms	local, $<$ 1 ms
CPU	2 x Core2Duo E8400 (3 GHz)	4 x Intel Core i7 860 (2.8 GHz)
RAM	4 GB	8 GB

Although the presented machines in the table should not make significant impact to process the defined datasets, possible overheads still need to be considered. These overheads are expected to be eliminated (to certain extent) in the assessments that follows. Since the overall execution time is correlated to the number of devices, that significantly increases, the execution time per device will be mostly impacted by small group sizes. If the number of smart meters within a group is small, the division with overall execution time will show low performance results. This (constant) overhead will however disappear for larger groups as the overhead is shared among all devices in the group composition.

IV. Aggregation Performance

Aggregation of the energy reading may be performed with multiple approaches as mentioned in section II. The first approach executes aggregation on the original measurements that contain no constant readings of energy. As already explained, this approach requires individual interpolation of the smart meters in order to be aggregated. The aggregation in the second approach is done on the pre-interpolated data, which actually requires more storage space, but the aggregation operation approximates to the weight of the regular *GROUP BY* statement of SQL. In this section, performance of both approaches is evaluated by the aforementioned DBMSs on data sets of different sizes. The results of the experiments are always referring to the time frame of one month (i.e. Sept. 2012), thus are independent of the selected set.

A. Interpolation and aggregation

If sampling of energy readings of smart meters is not made on equal frequency, or if samples are lost, the sampling frequency needs to be adjusted. The original datasets, presented in section III, have the distorted samples of energy readings collected from the smart meters. As such, the aggregation step will require data (of interest) to be interpolated at runtime. Once raw data is collected (by submitting an SQL query to a DBMS), the individual interpolation is executed and the aggregation step is performed. The performance results of those complete operations is presented as the execution time in relation to the group size for the two different MySQL engines. However, the relevance of the overhead as explained in section III should be noted as can be witnessed in all experiments, and resulted in higher execution times for all small groups e.g. of less than 50 devices.

For the InnoDB case, the experiences are conducted on the reduced and complete set, respectively set A and B. A performance comparison is made by the count of rows in the table, thus the storage size required, to be processed by the MySQL DBMS instance. Their comparison will offer a better understanding of how the execution time differs, when data sets of significant different sizes are stored on a hard disk. The interpolation algorithms presented in section II are used for both sets and the results are shown in Figure 2.

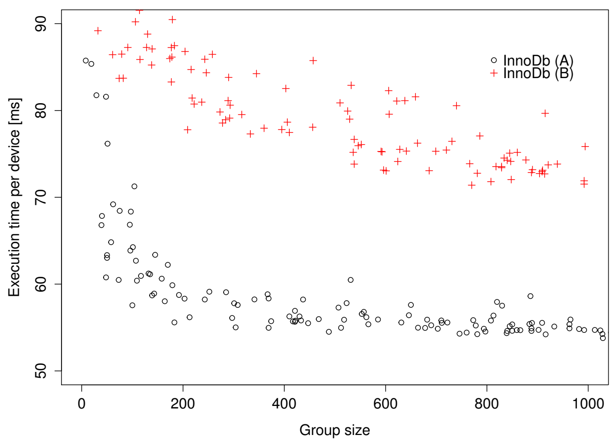

Figure 2. Execution time for interpolation and aggregation with the MySQL InnoDB engine

The execution time per device decreases with a higher group size for both datasets. However, one can also witness that the performance of the InnoDB engine suffers from the increase in the dataset size. This execution difference does not scale linearly with the size of the set, since the size of the set B is more than 6-fold. Still, because hard disk access is too expensive, one can immediately notice how performance suffers from the overhead for smaller group sizes. Finally, one can conclude that both scenarios continuously converge to the constant execution times per device as the group size increases.

The same experiment is also conducted with the MySQL in-memory engine. In contrast to the InnoDB engine, where performance of hard disk I/O operations must be considered, the in-memory engine stores the complete data set in memory. As for InnoDB, experiments here also use the stored procedure interpolation algorithm (as explained in section II), and Figure 3 shows the runtime results of numerous experiments.

One can see the slight performance drop of the algorithm running on the complete set B. A significant convergence rate can be also noticed, in comparison to InnoDB, even for very small group sizes. However, it is interesting to see that the InnoDB engine for set B. performed almost equally to the in-memory engine on the reduced set A. Since these results show that the runtime is not significantly affected by the dataset size, further analysis is conducted.

Although comparable results with in-memory and InnoDB

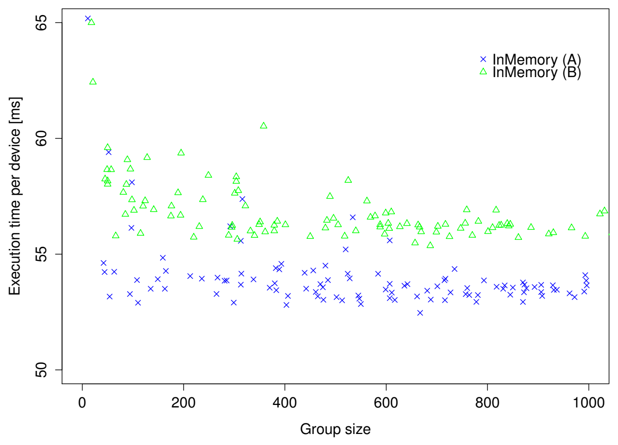

Figure 3. Execution time for interpolation and aggregation with the MySQL in-memory engine

engine for the experiments on the reduced set A can be witnessed, the experiments on the complete set resulted in performance degradation. The interpolation procedure of the InnoDB engine dropped for approximately 20 ms, while in-memory engine suffered only a drop of approximately 2 ms (between set A and B). The difference of few milliseconds is significant on a scale of thousands of devices within a group. As depicted on Figure 4, the distance is expected to further increase if a lager dataset is used.

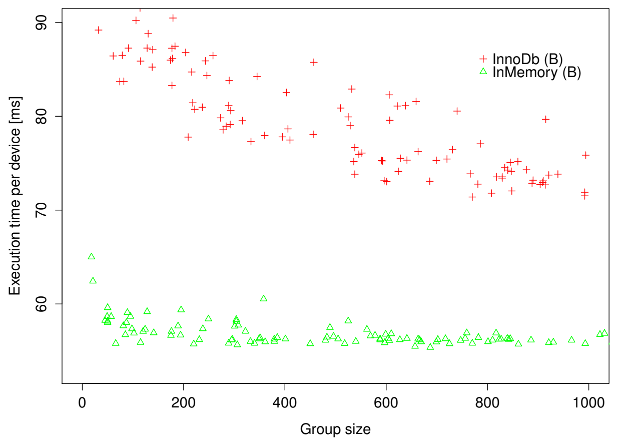

Figure 4. Execution comparison of the interpolation and aggregation

One can conclude that the in-memory engine performed better for all group sizes, while the InnoDB suffered approximately 10-fold more than the in-memory engine by the increase of the set size. Although the execution times in InnoDB keep decreasing for the bigger groups, the focus on very large groups (e.g. beyond 4000 devices) is not considered in this work. As a result, one can conclude that even for the better performing engine, the aggregation will still take approx. 60 seconds for a group of 1000 devices. In real world applications, such analysis may require much higher performance, especially in near real-time systems. In seek for a solution with better performance, and more efficient usage of the available resources [14], further experiments are conducted as these are demonstrated in subsection IV-B.

B. Aggregation on pre-interpolated data

In contrast to the experiment depicted in subsection IV-A, the aggregation here is made over pre-interpolated data. The distorted samples of energy readings within the original data sets (A and B) are pre-interpolated (A' and B') for each smart meter individually; hence now all the smart meters have the same sampling frequency (as this is stored within the DBMS). Pre-interpolation has some disadvantages such as fixing the sampling resolution and increasing the storage requirements, however it is expected to improve performance by reducing the aggregation time needed.

For the experiments carried out here, the sampling resolution was fixed to 15 minutes and contains a much higher count of sampling points, as shown in Table I. Via this preprocessing step, the interpolated data can be aggregated directly by executing a simple SQL query i.e. a *GROUP BY* statement. The performance of the complete operation is presented in form of the execution time with respect to the group size (for both MySQL and MonetDB). Similarly to the previous experiment, the relevance of the overhead explained in section III is included in all the experiments.

First the MySQL case is considered for both DBMS engines (InnoDB and in-memory) and compared to the reduced set A and complete set B. However, for the in-memory experiments the complete set of pre-interpolated meter readings could not fit the available physical memory of the machine. Still an assumption can be made, from the previous results of the in-memory experiments, that the execution time will not differ significantly in case of bigger data sets. Figure 5 depicts results of these experiments.

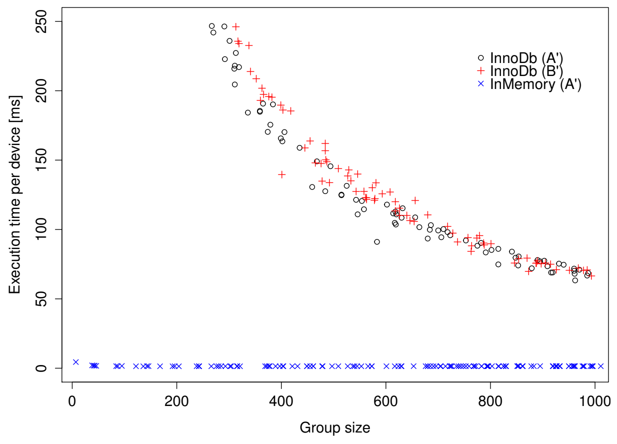

Figure 5. Execution time for the pre-interpolated data with the MySQL

Although the InnoDB shows an exponential performance improvement for higher group sizes for both sets, if compared to the results from subsection IV-A, we conclude that it faced a drastic drop in the performance. Obviously, these results show once more that the performance of the InnoDB engine is highly penalized by the size of the dataset. The in-memory engine shows exactly the opposite behavior, at least for the reduced pre-interpolated set A'. Interestingly the engine performed very good for all group sizes, especially if compared to the results of previous experiments (depicted on Figure 4). From these experiments one can conclude that the in-memory engine performed a lot faster, because the high-cost access time to the hard disk do not incur. As an indicative example, the execution time for group sizes of 1000 with the InnoDB is approximately 60 ms, while for the in-memory engine resulted in only 1.5 ms.

Due to the superior performance of the in-memory engine, it was decided to experiment with an in-memory column based DBMS i.e. the MonetDB, and conduct the same experiments. In contrast to the in-memory case of the MySQL experiments, the MonetDB solution has no problem storing reduced and complete datasets in memory. A distinctive feature of column stores is the application of aggressive data compression. In this way, one can use compression and some extra CPU cycles in order to fit the entire data set into the physical memory (what was not possible with the MySQL in-memory engine). The results of these experiments are depicted in Figure 6 for execution time per device.

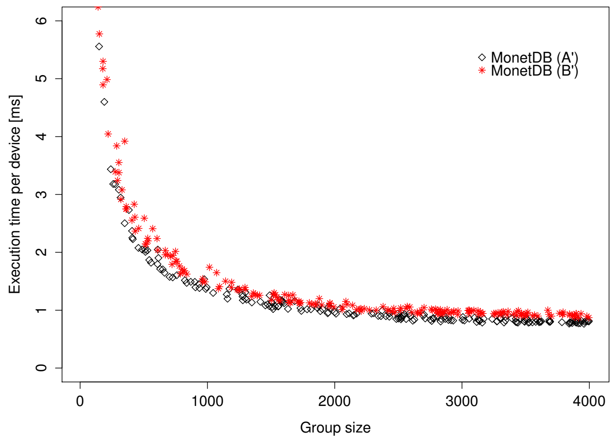

Figure 6. Execution time for pre-interpolated data with MonetDB

The performance of the MonetDB was much better when compared to the in-memory performance of the row-based MySQL. As an example, the performance of interpolation and aggregation of 1000 smart meters (on Figure 4) is $\approx 60ms$, while for this experiment we can see that it is 60-fold faster ($\approx 1ms$). The size of the datasets had a minimal impact on the performance, although the size of the data set B' is more than 6 times larger than the set A'. For both sets a fast convergence rate can be seen for group sizes greater than 1000, while the performance for the group sizes less then 1000 had a certain drop in performance. It is expected that the performance improvement rate is actually the software overhead, also being affected by the decompression time (as discussed in section III).

The acquired results are further analysed to cherry-pick the best ones. Obviously, the in-memory engine performed far better than InnoDB, thus it was selected for the overall comparison that is depicted in Figure 7. It is evident that the MySQL execution time decreases up to a group size of 200 and then starts growing, while the MonetDB execution time continuously decreases and converges to the execution time of less then $1ms$ per device. We witness that the MySQL in-memory engine exceeds the MonetDB performance for smaller group sizes, but limited only to the reduced pre-interpolated set A'. However, even though the MonetDB resulted in expensive execution times for smaller groups, it performs significantly faster for bigger group. For future industrial and business applications exactly these large groups are the main point of interest, and hence constitute our main focus.

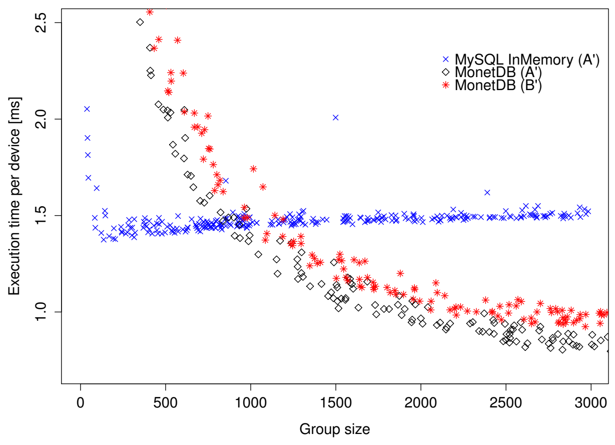

Figure 7. Execution time comparison for the pre-interpolated data per device

C. Overall Comparison

The experiments conducted for both defined scenarios depict how the aggregation performance is affected by the various storage technologies as well as the potential pre-processing of data such as the pre-interpolation. The analysis of the experiments revealed that pre-interpolation of data has significant impact on the performance boost. The column based DBMS (MonetDB) proved powerful by storing reduced and the complete dataset entirely in memory, while the traditional DBMS (MySQL) had severe limitation in our experimental environment. Still, the MySQL in-memory engine over performed MonetDB for smaller groups, while MonetDB showed continuous improve even after over performing MySQL.

To get a better understanding of the performance benefit of both solutions, the total execution time needs to be compared from the best of breed DBMS cases. The total execution time (in seconds) of experiments on the reduced pre-interpolated set A' are shown inFigure 8. The high performance improvement realized by the MonetDB even for bigger groups (as shown in Figure 6), leads to increase of the gap between the overall performance between the two DBMSs. Although overall MonetDB performed better for the experiments of bigger group sizes, Figure 8 depicts how the total execution time increases for the bigger groups (however at a much lower rate than MySQL).

These experiments can be used as a rule of thumb towards

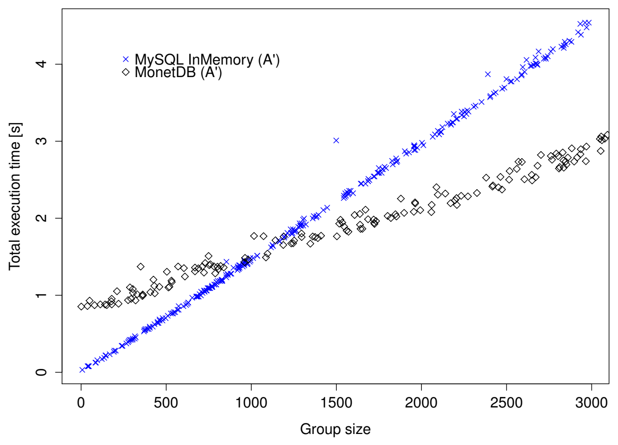

Figure 8. Execution time comparison for the pre-interpolated data

making an informed decision for solutions running in different environments. Depending on the performance requirements and data, one can select configurations that fit the business objectives. Hence for a highly limited environment, one needs to consider the performance limitations if the interpolation is executed at runtime. However, if high performance is required, one should focus on the pre-interpolated data sets as these assist towards removing the performance penalty (time) needed for the interpolation step. Pre-interpolation of data sets can be scheduled more flexibly for historical data (e.g. whenever server capacity allows it), as such action leaves only the real-time data to be analyzed. Additionally even in real-time, one could consider other aspects such as parallelization of pre-interpolation as well as processing of the data which could yell higher performance.

V. Conclusion

We have demonstrated the performance aspects of time series aggregation by using a traditional row-based DB (MySQL) and in-memory-column based approach (MonetDB). The numerous experiments reveal the overall performance superiority of the in-memory column based approach. We also propose to pre-interpolate data (if possible) for high performance, since the execution times may be better e.g. 60-fold as indicatively shown in the experiments depicted in this work.

The vision of integrating real-time analytics into modern applications such as a smart city energy cockpit, clearly points out to the need of being able to do high-performance processing of energy related data. Similar cases could be made for more industrial scenarios such as energy analytics within a factory [8]. Apart from the steps on grouping and interpolation that we have described here, other aspects need also to be considered such as the guarantee of a high quality dataset. The latter implies checks for data consistency, validation against domain-specific rules, potential transformations etc. all of which may need to be made on-the-fly as data gets streamed into the enterprise systems. To this end, we will expect to further work in the future towards integrating real-time analytics in smart city applications and evaluating them under real conditions. The latter due to increasing privacy, security and business concerns may imply to also experiment with encrypted queries [15] and assess their usage in massive-data real-time analytics.

Acknowledgment

The authors would like to thank for their support the European Commission, and the partners of the EU FP7 projects SmartKYE (www.SmartKYE.eu) and NOBEL (www.ict-nobel.eu) for the fruitful discussions.

References

[1] X. Yu, C. Cecati, T. Dillon, and M. Simões, "The new frontier of smart grids," *Industrial Electronics Magazine, IEEE*, vol. 5, no. 3, pp. 49–63, Sep. 2011.

[2] R. Katz, D. Culler, S. Sanders, S. Alspaugh, Y. Chen, S. Dawson-Haggerty, P. Dutta, M. He, X. Jiang, L. Keys, A. Krioukov, K. Lutz, J. Ortiz, P. Mohan, E. Reutzel, J. Taneja, J. Hsu, and S. Shankar, "An information-centric energy infrastructure: The berkeley view," *Sustainable Computing: Informatics and Systems*, 2011.

[3] S. Karnouskos, "Cyber-Physical Systems in the SmartGrid," in *IEEE 9th International Conference on Industrial Informatics (INDIN), Lisbon, Portugal*, Jul. 26–29 2011.

[4] J. Santaferraro, "Offloading analytics." *Business Intelligence Journal*, vol. 17, no. 4, pp. 43–48, 2012.

[5] J. Yin, A. Kulkarni, S. Purohit, I. Gorton, and B. Akyol, "Scalable real time data management for smart grid," in *Proceedings of the Middleware 2011 Industry Track Workshop*, ser. Middleware '11. New York, NY, USA: ACM, 2011, pp. 1:1–1:6.

[6] D. Ilic, S. Karnouskos, and P. Goncalves Da Silva, "Improving load forecast in prosumer clusters by varying energy storage size," in *IEEE PowerTech 2013, Grenoble, France*, 16–20 Jun. 2013.

[7] S. Karnouskos, P. Goncalves Da Silva, and D. Ilic, "Energy services for the smart grid city," in *6th IEEE International Conference on Digital Ecosystem Technologies – Complex Environment Engineering (IEEE DEST-CEE), Campione d'Italia, Italy*, Jun. 2012.

[8] S. Karnouskos, A. W. Colombo, J. L. M. Lastra, and C. Popescu, "Towards the energy efficient future factory," in *Proc. 7th IEEE International Conference on Industrial Informatics INDIN 2009, Cardiff, UK*, Jun. 23–26, 2009, pp. 367–371.

[9] H. Plattner, "A common database approach for OLTP and OLAP using an in-memory column database," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 1–2.

[10] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: how different are they really?" in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 967–980.

[11] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd, "Efficient transaction processing in SAP HANA database: the end of a column store myth," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 731–742.

[12] P. A. Boncz, M. L. Kersten, and S. Manegold, "Breaking the memory wall in MonetDB," *Commun. ACM*, vol. 51, no. 12, pp. 77–85, Dec. 2008.

[13] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA database: data management for modern business applications," *SIGMOD Rec.*, vol. 40, no. 4, pp. 45–51, Jan. 2012.

[14] S. Karnouskos, P. Goncalves da Silva, and D. Ilic, "Assessment of high-performance smart metering for the web service enabled smart grid era," in *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 133–144.

[15] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 85–100.