Stream Computing Based Synchrophasor Application For Power Grids

Jagabondhu Hazra
IBM Research, India
jaghazra@in.ibm.com

Kaushik Das
IBM Research, India
kausdas8@in.ibm.com

Deva P. Seetharam
IBM Research, India
dseetharam8@in.ibm.com

Amith Singhee
IBM T J Watson Research Center, USA
asinghe@in.ibm.com

ABSTRACT

This paper proposes an application of stream computing analytics framework to high speed synchrophasor data for real time monitoring and control of electric grid. High volume streaming synchrophasor data from geographically distributed grid sensors (namely, Phasor Measurement Units) are collected, synchronized, aggregated when required and analyzed using a stream computing platform to estimate the grid stability in real time. This real time stability monitoring scheme will help the grid operators to take preventive or corrective measures ahead of time to mitigate any disturbance before they develop into wide-spread. A protptype of the scheme is demonstrated on a benchmark 3 machines 9 bus system and the IEEE 14 bus test system.

Categories and Subject Descriptors

J.2 [**Physical Sciences and Engineering**]: Engineering

General Terms

Algorithms

Keywords

Stream Computing, Power Grid, Synchrophasor, Voltage Stability

1. INTRODUCTION

Modern power grids are continuously monitored by trained system operators equipped with sophisticated monitoring and control systems. Despite such precautionary measures, large blackouts, that affect more than a million consumers, occur quite frequently. Analysis of large blackouts show that one of the major cause of these events are lack of real time situation awareness of the grid [1]. In many blackouts, system operators were unaware that the grid had split into several sub-systems and hence could not initiate corrective or preventive controls in time to mitigate such events. This has necessitated efficient tools and technologies for real time grid monitoring to identify and deal with faults before they develop into wide-spread disturbances.

In recent years, electric grids are undergoing dramatic changes in the area of grid monitoring and control. Key factors behind such transformation includes tremendous progress in the areas of power electronics, sensing, control, computation and communication technologies. For instant, conventional sensors (e.g. Remote Terminal Units) provide one measurement or sample per 4-10 seconds whereas new sensors like Phasor Measurement Units (PMUs) could provide upto 120 measurements or samples per second. Moreover, PMUs provide more precise measurements with time stamp having microsecond accuracy [2]. Hence, phasor measurement units could provide a precise, comprehensive view of the entire grid.

A synchrophasor system (as shown in Fig 1) includes phasor measurement units (PMUs) to collect real-time data and a communications system (such as a private utility line, the public switched telephone network, or the internet) to deliver the data from many PMUs to a local data concentrator (usually hosted by the utility that owns the PMU) called Phasor Data Concentrator(PDC). Concentrated data are relayed on a wide-band, high-speed communications channel to a higher-capability data concentrator sometimes called Super Phasor Data Concentrator (SPDC), that feeds the consolidated data from all the PDCs into analytical applications such as a wide-area visualization, state estimator, stability assessment, alarming, etc. Synchrophasor applications need to ingest, process, and analyze continuous data streams from heterogeneous sources. The high volume of streaming data often makes it impossible to fully store and process all the data from disk. Fortunately, emerging stream computing paradigm not only capable of dealing with high volume of streaming data but also enables the extraction of new insights from data in real-time. Further, it provides functionalities like reconfigurability, scalability etc. to the applications which are key requirements for these power system applications.

This paper shows how streaming synchrophasor data could be collected, synchronized, aggregated (when required) and analyzed for real time power system application like voltage stability monitoring. Monitoring and analysis of these synchrophasor data let observers identify changes in grid conditions, including the amount and nature of stress on the system, to better maintain and protect grid reliability. Proposed scheme is evaluated on benchmark 3 machines 9 bus test system and IEEE 14 bus test system.

HiPCNA-PG'11, November 13, 2011, Seattle, Washington, USA.

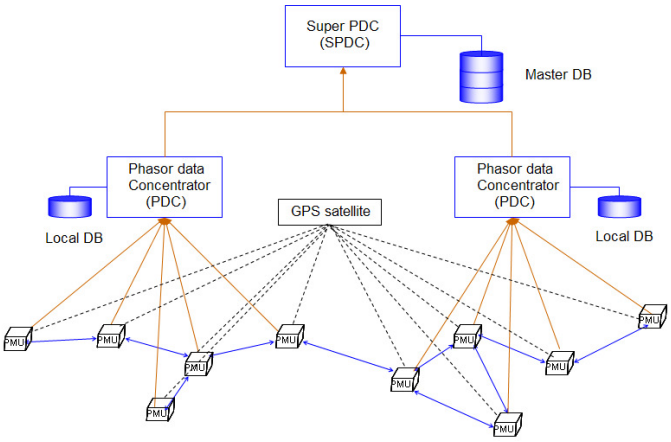

Figure 1: A synchrophasor network consisting of PMUs, PDCs and communication network

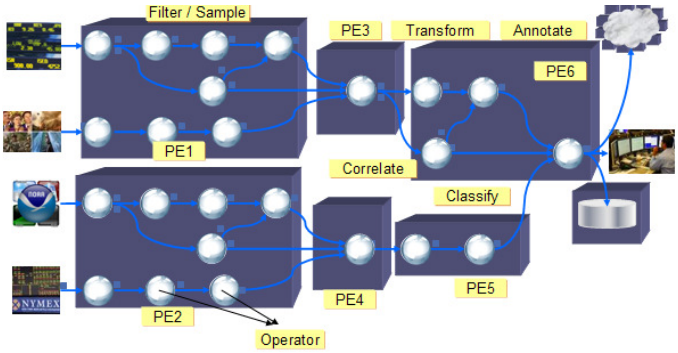

Figure 2: InfoSphere stream processing graph

2. STREAM COMPUTING

Several power system applications that would need to perform very low latency computations on real-time streaming data. It needs a computational framework that can scale well with increasing large amounts of streaming data and increasingly complex and numerous applications running in parallel on this data. Similar challenges have been faced in other fields. An example is financial engineering, where split second investment decisions have to made based on computations on large volumes of streaming data [3], often involving data analytics, pattern discovery and model training tasks similar to the case of power system or presently known as smart grid applications. A popular solution emerging for these scenarios is stream computing.

Stream programming is typically done by creating a dataflow graph [4] of operators (as shown in Fig. 2), which performs the computation required by the application. The inputs to the dataflow graph can be data from a variety of sources, such as internet sockets, spreadsheets, flat files, or relational databases, and may be consumed by one or more input operators in the graph. A synchronization primitive is an example of an operator which consumes data from multiple data streams and then outputs data only when data from each stream is read. This can be a useful operator for PMU data which can arrive at different times from different PDCs due to variable network delays and sampling times. Other relevant operators would be a fast Fourier transform (FFT) operator or a moving average operator. Each data element arriving at the input to an operator is typically treated as an event and the operator takes appropriate action when events occur at its input. Operators may be pipelined to perform in a sequential manner: output data from one operator is consumed by the next downstream operator. Operators may also perform in parallel if they do not have data dependencies: output from one operator may be consumed by two or more different operators working in parallel. There operators are typically contained within containers called stream processing elements. For fast, parallel execution, the processing elements are automatically partitioned onto parallel processors and/or machines. The optimal partitioning depends on factors such as the amount and type of data streaming through different processing elements, the resource requirements for each of them and the dependencies between them. Hiding the details of parallel programming from the user greatly improves productivity and efficiency of streaming application deployment. The flexibility of input formats, the ease of developing and connecting the operators, and the automatic compilation onto parallel processors makes stream processing attractive.

Even though stream computing languages make application development quite easy, one may need to redesign traditional algorithms and applications to optimally use the stream processing flow. Reference [5] gives an overview of algorithmic advances and challenges in implementing efficient streaming versions of traditional algorithms. A specific example in this context is [6], where the authors implement decision tree learning for a streaming system. Real-time data will be available from increasingly numerous data sources across the grid. Stream computing frameworks hold the potential to enable scalable real-time applications that can extract information from this data to enable more complete situational awareness and very fast operational response and control.

Although several stream computing platforms [7], [8] have been developed recently, we are using IBM InfoSphere Streams [9], a stream processing middleware from IBM Research that supports high-performance, reconfigurable stream computing.

3. APPLICATION REQUIREMENTS

Real-time analytical applications for synchrophasor data demand some functional and non-functional requirements from the implementation platform.

3.1 Functional requirements

Key functional requirements of the application for the real time operation are as follows:

- *Synchronization:* Although the clocks of all the PMUs are synchronized at the accuracy of 1 microsecond, but the sampling rate and the latency for different PMUs are different. So, the data traveling from different locations have to be synchronized again by the programming platform for the analysis. There are some operators like Barriers and Delay which are specifically suitable for these type of tasks. Barrier does not operate on the data until data from all the ports arrive. Delay operator is used to delay a stream by a given amount of time while keeping the inter-arrival times of the tuples intact.

- *Analysis engines:* Analytics like transient prediction, voltage stability index calculation and state estimation form the core of real-time synchrophasor applications. Streams allows implementation of a variety of computations in the Functor operator using the Streams Programming Language (SPL). For

complex analytics, which would typically be implemented in C++, Streams provides a very flexible and efficient interface between SPL and C++ to implement custom Streams operators using C++. This interface is optimally efficient because the Streams compiler converts all SPL to C++ code and build the final operator object code using C++ compilation. All user-define C++ code would then be compiled together with the pre-defined operators.

- *Application health visualization:* Such applications are used to monitor and control critical infrastructure (electric power grids). Hence, it becomes quie important to monitor the health of the application itself so as to respond to any failures in the software (e.g., broken TCP connection to the data sources). InfoSphere Streams provides the capability for real time visualization of the operators and flow of data streams in the running application.

- *Historian:* It is expected to store large amounts of time series synchrophasor data for post fault analysis. The application should be able to store and retrieve such a vast amount of data whenever required. This facility is easily achieved by integrating InfoSphere Streams operators with a historian of choice, for example a time series data base. Such integration is enabled by the C++ and Java interfaces available in Streams.

- *Filtering noisy data:* Due to instrumentation, communication or other errors, some noise might be incorporated in the data. It is helpful if the platform provides some methods to filter such data. InfoSphere Streams provides an operator called 'FILTER' which can filter these noisy data if those data have values which are beyond a pre-specified range. Further, it might be helpful in diagnosing permanent instrumentation faults if history of the data shows similar noise for a longer amount of time.

3.2 Non-functional requirements

The software platform should also provide the following non-functional capabilities:

- *Low latency:* Since real time operation is envisaged using large amount of data flowing over large geographical and network distance, it is always desirable to have minimum latency introduced by the programming platform or the application itself. The latency introduced by the operators in the InfoSphere Streams is negligible since it plays with the data in the fly without the requirement of storing them.

- *High data throughput:* In a large power system, data will flow from large number of PMUs deployed over the system which produce data at a maximum rate of 120 samples per second. So it is expected for the application to handle large amount of data from 1000 of PMUs flowing at the rate of MB/s. Since InfoSphere Streams can operate on these data in parallel in multiple nodes without the need of storing them therefore, InfoSphere Streams can apparently handle infinite streams of data in real time.

- *Reconfigurability:* In power system, equipments are put out for maintenances or due to outages. So the software platform should allow for dynamic reconfiguration of the system without requiring to stop the software.This facility is provided by InfoSphere Streams since it allows for dynamic connections and submitting as many jobs as needed. Streams generated by one application often serve as input to another, and the second application may be deployed when the first is already running. The Import and Export special operators support dynamic application composition, where one application exports a stream, another application imports the stream, but both applications are instantiated separately and may even be mutually anonymous.

- *Scalability:* The software developed should allow for different size and configuration of the system so that the future upgradation of the system is not limited by the software capabilities. InfoSphere Streams achieves this goal by combining a simple and flexible flow language with an extension mechanism for integrating native code as fast, flexible, and reusable stream operators.

- *Highly available* Software should be highly available in the sense that even during any configuration change or software upgrades the software should not be shut down. Further, even if communication to any of the PMUs is lost due to any reasons, still then the software should be available. In InfoSphere Streams, if any of the nodes fails, the tasks of that node can be taken by the other nodes, thereby making the software available even during faults or maintenances of the nodes.

- *Easily re-deployable:* The software should be easily re-deployable to any system regardless of any system parameters. InfoSphere Streams dynamically deploys the operators so dynamically it makes the codes ompatible for re-deploying.

- *Easy integration with downstream applications/engines:* It should be easily integrable to any other power system analysis engine regardless of the format of that engine. InfoSphere Streams allows configuring output port into various format(TCP socket, UDP socket, File).

4. VOLTAGE STABILITY INDEX

Having above mention features and capability, InfoSphere Streams could be efficiently used for various real time power grid applications like state estimation, visualization, voltage stability monitoring, transient stability monitoring, adaptive protection, High voltage AC and DC line control, frequency control, congestion management, real time energy pricing, etc. This paper shows an example how stream computing could be used for real time voltage stability monitoring.

A voltage stability monitoring system measures voltage stability which is a serious concern for grid operators due to its importance in ensuring system security and power quality. Loss of voltage stability, also known as a voltage collapse, can either result from the inability of the power system to supply reactive power or by an excessive absorption of reactive power by the system itself. Grid operators use a metric known as Voltage Stability Index (VSI) to determine the probability of a voltage collapse and to identify vulnerable buses (a bus is also known as a substation). Voltage stability index of bus i is defined as [10]:

$$V_{si} = \frac{\partial P_i / \partial \delta_i}{\sum_{j=1, j\neq i}^{n} B_{ij} V_j} \quad (1)$$

Where, n is number of buses in the system, P_i is the real power injection at bus i, V_j is the magnitude of voltage at bus j, δ_i is the phase angle of voltage at bus i and B_{ij}'s are elements of the network admittance matrix. $B_{ij} = 0$ if no transmission line connects bus i directly to bus j in the grid. The stability index of a grid is simply the minimum of stability indices of all its buses and therefore depends on the most vulnerable bus. For a stable grid, the stability index is generally greater than 0.8. However a value close to 0.5 indicates that the grid is on the verge of a collapse.

4.1 Aggregation for Graceful Degradation

In a voltage monitoring system, PMU devices measure and publish streams of voltage parameters (magnitude V_j, phase angle δ_i) of buses within the grid. Applications subscribe to these streams in order to compute the VSI's of buses. During network overload, PMU streams traveling from the publishers toward a subscriber can be aggregated at intermediate nodes to minimize load. Voltage monitoring system, like a lot of smart grid applications, can safely work with such aggregated data. As discussed in [11] different aggregation functions can be applied in-network during conditions of overload, thus gracefully degrading the quality of VSI's inferred at the subscriber.

Data Prioritization Not all data from all PMUs is equally important to compute the VSI's of buses. In particular, the following heuristic can be used to drop low priority data and reduce network load. Data from PMUs deployed at transmission/distribution substations whose reported values are below their rated values must be given higher priority because such a dip indicates proximity to voltage collapse. Therefore the following steps can be used for data prioritization at an intermediate node for voltage stability monitoring:

1: Collect PMU voltage magnitudes $X_t = [v_1, v_2, ..., v_n]$ (in per units) measured at a common time instant t.
2: Sort X_t in ascending order.
3: **for** $i = 1$ to n **do**
4: **if** $X_t^i <= v_{th}$ **then**
5: select X_t^i
6: **end if**
7: **end for**

where, v_{th} is threshold value of voltage that is usually set at 1.0.

Data Dropping In power systems, voltage phasor measurements do not change abruptly unless there are disturbances or faults in the system. During normal operation, the state of the system changes gradually. Therefore during overload, data from PMU streams originating from grid subsystems that are operating normally and do not change significantly over time, can be dropped. The following function can be used to drop data for voltage stability monitoring:

1: Let $X_t = [x_t^1, x_2^2, ..., x_t^n]$ and $X_{t-1} = [x_{t-1}^1, x_{t-1}^2, ..., x_{t-1}^n]$ be the data at time instants t and $t-1$ respectively
2: **for** $i = 1$ to n **do**
3: **if** abs$(x_t^i - x_{t-1}^i) <= x_{th}^i$ **then**
4: drop data x_t^i
5: **end if**
6: **end for**

where, x_{th}^i is threshold value of the data. In a realistic setting, a threshold of 0.005 per unit can be used for voltage magnitudes and 0.5 degree for phase angles.

Data Clustering Several PMUs in the grid may report voltage values that are identical or numerically close to each other. These could be clustered to reduce data volume. Following steps can be used for clustering:

1: Let $X_t = [x_1, x_2, ..., x_n]$ be the data of time instant t. Let k be the number of clusters.
2: Choose k random data points as initial cluster center μ
3: Calculate the distance, $d(i, j)$ from the center of each cluster to each data point
4: **for** $i = 1$ to k **do**
5: **for** $j = 1$ to n **do**
6: $d(i, j) = (\| x_j - \mu_i \|)^2$
7: **end for**
8: **end for**
9: Assign each data point, x_j to the cluster i where $d(i, j)$ is minimal
10: Compute mean center of each cluster μ_i for all clusters
11: Repeat Steps 3-10 until all the data points are assigned to their optimal cluster centers.

Partial Computations VSI computations can be performed in a distributed manner since VSI of a bus only depends on measurements at that bus and those buses directly connected to it. Therefore instead of forwarding raw PMU streams, nodes can participate in computations and only forward partially computed results. With usual notation, Eq. (2) shows that VSI of bus i can be estimated using partial summations that can be performed at different intermediate nodes of the network:

$$\frac{1}{V_{si}} = \frac{\sum_{j=1, j\neq i, j\in A}^{n} B_{ij}V_j}{\partial P_i/\partial \delta_i} + \frac{\sum_{j=1, j\neq i, j\notin A}^{n} B_{ij}V_j}{\partial P_i/\partial \delta_i} \quad (2)$$

5. EXPERIMENT

Application is evaluated on benchmark 3 machines 9 bus system as shown in Fig. 3 and IEEE 14 bus test system. Benchmark 9 bus system has 3 generators, 6 transmission lines, and 3 transformers. It is divided into three zones: Zone 1 - buses 2, 5, and 7; Zone 2 - buses 1, 4, and 6 and Zone 3 - buses 3, 8, and 9. Each bus or substation has a PMU that generates a stream of samples at a rate of 20Hz. PMUs report their streams to respective local Phasor Data Concentrator (PDC) nodes. That is, PMUs in Zone 1 report measurements to PDC1, PMUs in Zone 2 to PDC2 and PMUs in Zone 3 to PDC3. These PDCs in turn forward the measurements to a central super PDC (SPDC) that hosts the voltage stability monitoring application. Similarly, 14 bus system is also divided into 3 zones and in each zone one PDC is placed.

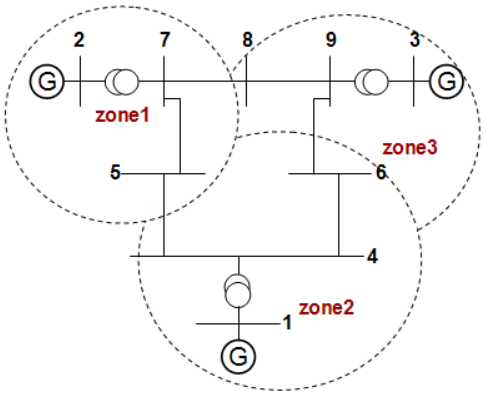

Figure 3: Benchmark 3 macines 9 bus test system

For 9 bus test system, InfoSphere stream processing graph is shown in Fig. 4. In this graph, *Source* operators are used to col-

lect live data from the grid sensors. *Source* operator creates stream from data flowing from the external sources. This operator supports universal resource locators (URIs) like file, UDP datagram-based socket, and TCP socket connection. Tuples within the data streams are then sorted using *Sort* operator. *Sort* operator creates a temporary window based on either tuple count or time and sort the tuples within the window based on tuple attributes. In this application tuples are sorted based on data time stamp. In Fig. 4 first 3 *sort* operators run within PDC1, next 3 run in PDC2, and last 3 run on PDC3. Within each PDC, data streams from sensors are synchronized using *Barrier* operator which consumes tuples from multiple streams, outputting a tuple only when a tuple from each of the input streams has arrived. Similarly, streams from three local PDCs are again synchronized in SPDC using another barrier operator as shown in Fig. 4. Local PDCs could host applications like fault detection, fault analysis, circuit breaking switching, disturbance recording, etc and generate appropriate feed back controls. Local PDCs also host the data aggregation functions to manage communication congestions. Each local PDC combines the tuples from each sensor and creates one tuple and send to the super PDC. Super PDC unbundles the tuples using *functor* operator, sorts the tuples based on time stamp and hosts the application. In the data stream, all the tuples having same time stamp are separated by punctuation mark. This punctuation mark is incorporated using *punctor* operator as shown in Fig. 4. *Aggregate* operator creates a temporary data window, fills it with the tuples having same time stamp (between two punctuation mark) and extracts the required information for a particular application from the raw data streams. Extracted information are used within *functor* operator to determine the stability of the grid in real time.

In order to demonstrate the real time voltage stability monitoring scheme, two case studies are made. In the first case, grid is assumed to operate in rated condition and it is gradually overloaded by up to 20% of its rated capacity using a step size of 1%. In the second case, with the increase in grid load, we have incorporated a fault in the system and transmission lines. Fig. 5 shows the voltage stability indices (VSIs) of all the substations of 9 bus system. Fig. 5 shows that VSIs decrease correctly in response to overloading and last three curves corresponding to buses 5, 6, and 8 have low stability indices. Hence these are most affected buses by overload and voltage instability is more likely to happen in these buses. Thus real time visualization of these stability curves will provide situation awareness of the complete grid to the system operators. Fig. 6 presents grid behavior with a fault in the grid. It clearly shows the fault inception point and also shows as soon as fault is incepted in the grid, stability indices of buses 4, 5 and 6 drastically reduces and become unstable within few seconds. However, such situation could be avoided if proper controls are initiated as soon as fault is detected by the monitoring scheme.

We evaluated the impact of in-network aggregation of PMU streams on the accuracy of voltage stability index calculations using aggregation algorithms described in section IV. Figure 7 shows the computed VSI for bus 14 under different operating conditions when the grid is overloaded from 0-20%. We can see that the aggregation functions alter the VSI only by a small factor. The VSI estimate computed using various aggregation algorithms remains close to the VSI computed using raw PMU streams and in general all the aggregation algorithms perform well.

Figure 8 compares the different aggregation algorithms and plots the error in estimated voltage stability indices averaged across *all* 14 buses in the grid. Overall, all aggregation algorithms perform well and the errors vary between 1-8%. One can observe that partial computation method perform best compared to other aggregation functions. This is not surprising given that distributed computations do not introduce any error. The error in partial computations results from loss of time synchronization between measurements. On the other hand, error for data prioritization is high and fluctuates widely. The error for data clustering is high as well, but remains constant. Lastly, the error level for data dropping fluctuates, but within a small range.

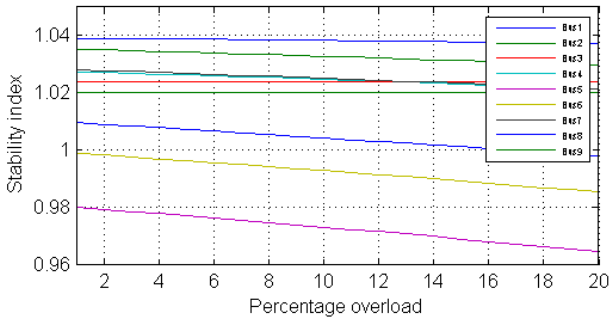

Figure 5: Real time voltage stability monitoring using InfoSphere Streams

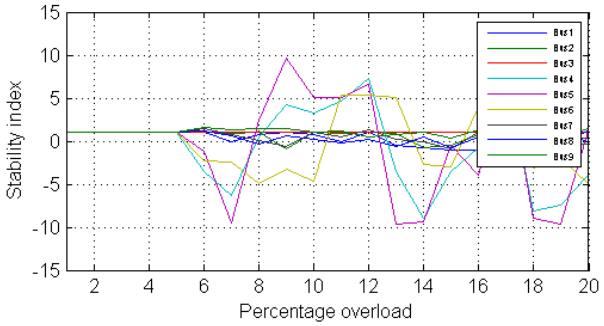

Figure 6: Real time voltage stability monitoring using InfoSphere Streams

Figure 9 shows the percentage reduction in traffic volume with different in-network aggregation algorithms. The figure shows that in general 20-50% data reduction is possible by using aggregation algorithms designed for voltage monitoring application. In case of data prioritization, the possibility of reduction in traffic gradually decreases from 32% to 21% as most of the substations suffer from

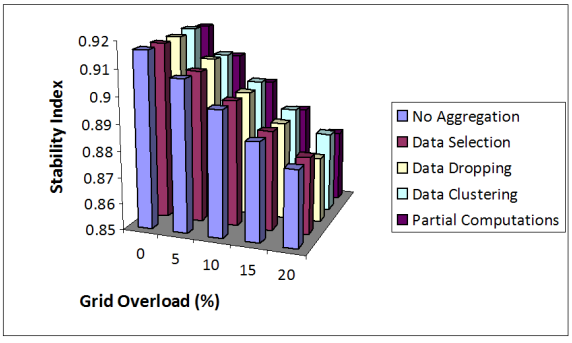

Figure 7: Voltage Stability Index of bus 14 computed using a number of different in-network aggregation algorithms.

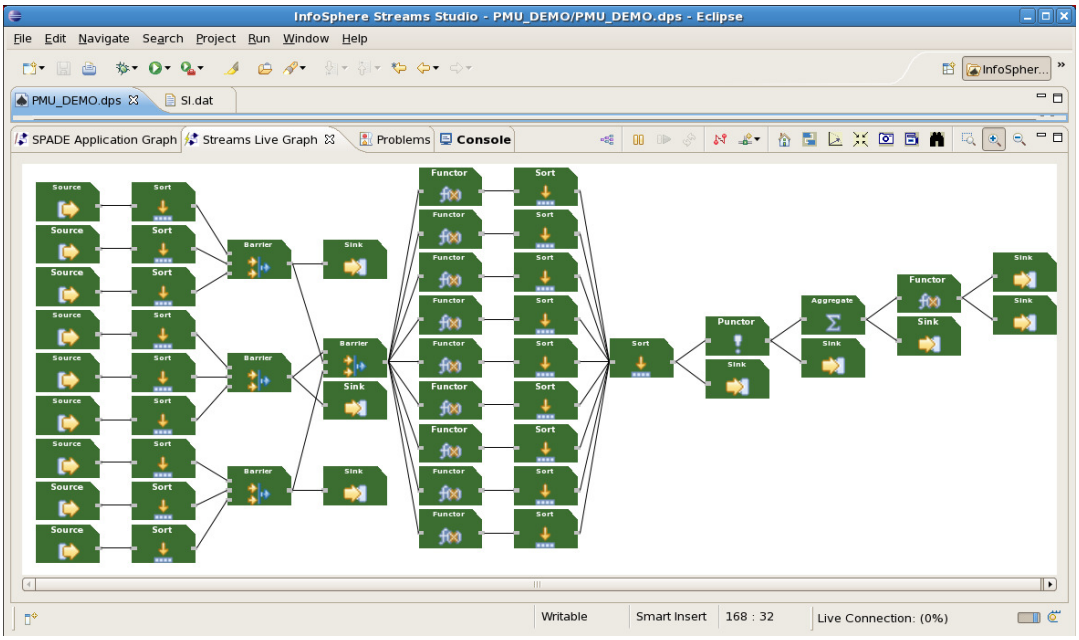

Figure 4: InfoSphere Streams processing graph for real time voltage stability monitoring

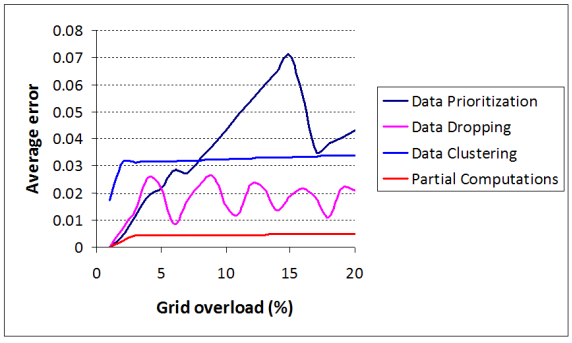

Figure 8: Errors in Voltage Stability Index averaged over 14 buses with different in-network aggregation algorithms.

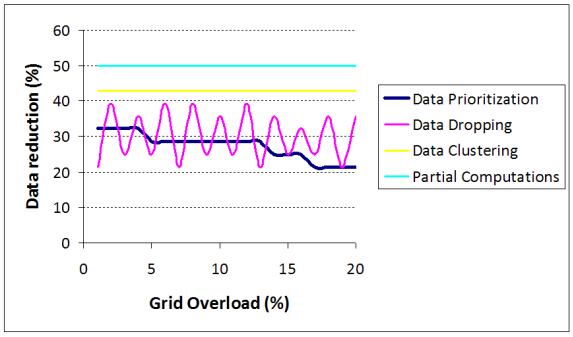

Figure 9: Reduction in traffic volume with different in-network aggregation algorithms.

voltage problems with increased overload. For data dropping algorithm, reduction in data fluctuates between 20-40% as voltage phasors change significantly with roughly 2-3% change in load. Similarly, clustering (with 8 clusters) results in about 40% reduction of traffic while partial computations (with 3 PDC nodes) results in the largest reduction of about 50%.

The above experiments clearly demonstrate that voltage stability monitoring can safely work with in-network aggregated data. Thus this property can be exploited for any practical grid of several thousand substations to reduce traffic volume during network overload and gracefully degrade application performance.

6. CONCLUSIONS

This paper illustrated how stream computing analytic could be used for real time voltage stability monitoring of electric grid. This monitoring scheme will provide a real time situation awareness to the grid operator to help in taking preventive/corrective control ahead of time. As stream computing supports scalability, the application could be easily deployed to any real grid irrespective of size.

7. REFERENCES

[1] G. A. et al., "Reasoning about naming systems," *IEEE Trans Power Systems*, vol. 20, no. 4, pp. 1922–1928, November 2005.

[2] K. Zhu, L. Nordstrom, and L. Ekstam, "Application and analysis of optimum pmu placement methods with application to state estimation accuracy," in *IEEE PES General Meeting*, 2009, pp. 1–7.

[3] H. Wu, B. Salzberg, and D. Zhang, "Online event-driven subsequence matching over financial data streams," in *ACM SIGMOD Intl. Conf. on Management of Data*, 2004.

[4] B. Gedik, H. Andrade, K. L. Wu, P. Yu, and M. Doo, "Spade: The system s declarative stream processing engine," in *SIGMOD*, 2008, pp. 1123–1133.

[5] S. Muthukrishnan, *Data streams: algorithms and applications*. Now Publishers, 2005.

[6] G. Hulten, L. Spencer, and P. Domingos, "Mining time changing data streams," in *ACM Conf. on Knowledge Discovery and Data Mining*, 2001.

[7] D. J. A. et al, "The design of the borealis stream processing engine," in *Proc. CIDR*, 2005, p. 277-289.

[8] T. S. Group, "Stream: The stanford stream data manager," 2003.

[9] IBM-Research, "Infosphere streams," http://www-01.ibm.com/software/data/infosphere/streams/.

[10] A. K. Sinha and D. Hazarika, "Comparative study of voltage stability indices in a power system," *Electrical Power and Energy Systems*, vol. 22, no. 8, pp. 589–596, 2000.

[11] V. Arya, J. Hazra, P. Kodeswaran, D. Seetharam, N. Banerjee, and S. Kalyanaraman, "Cps-net: In-network aggregation for synchrophasor applications," in *Third International Conference on Communication Systems and Networks (COMSNETS)*, 2011.