Adaptive Rate Stream Processing for Smart Grid Applications on Clouds

Yogesh Simmhan
University of Southern California
Los Angeles CA 90089
simmhan@usc.edu

Baohua Cao
University of Southern California
Los Angeles CA 90089
baohuaca@usc.edu

Michail Giakkoupis
University of Southern California
Los Angeles CA 90089
mgiakkoup@usc.edu

Viktor K. Prasanna
University of Southern California
Los Angeles CA 90089
prasanna@usc.edu

ABSTRACT

Pervasive smart meters that continuously measure power usage by consumers within a smart (power) grid are providing utilities and power systems researchers with unprecedented volumes of information through streams that need to be processed and analyzed in near realtime. We introduce the use of Cloud platforms to perform scalable, latency sensitive stream processing for *eEngineering* applications in the smart grid domain. One unique aspect of our work is the use of adaptive rate control to throttle the rate of generation of power events by smart meters, which meets accuracy requirements of smart grid applications while consuming 50% lesser bandwidth resources in the Cloud.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; D.2.11 [**Software Architectures**]: Domain-specific architectures; H.2.4 [**Systems**]: Query processing

General Terms

Design, Algorithms, Experimentation

Keywords

Smart Grid, Stream Processing, Adaptive rate control, Resource management, Cloud platform

1. INTRODUCTION

Increasing environmental consciousness and energy security concerns are causing a renewed focus on improving the electrical power infrastructure to enable optimal use of available power capacity. Electricity utilities are upgrading their power grids to Smart Grids, where Advanced Metering Infrastructure (AMI) (or smart meters) installed at consumer premises allow bi-directional network communication of power usage and other signals between the consumer and the utility in realtime [1].

Smart meters result in an unprecedented increase in the volume of information available to the utility and to power systems researchers – a quantum leap from one (manual) meter reading per month to one (automatic) smart meter reading per minute. Utilities like the Los Angeles Department of Water and Power [11] plan to use such fine-resolution power usage data streaming from millions of customers for realtime demand-response (DR) optimization, where peak power usage is accurately forecast and load curtailment operations initiated through pricing signals [7]. *eEngineering applications* in smart grids, such as power load forecasting, power usage analysis and grid failure detection, will require low latency stream processing at large scales that the existing IT infrastructure of utilities and power systems researchers are not designed to handle. For example, power events generated at a peak rate of 1 KB/min from 1.4 M consumers in the Los Angeles Smart Grid will require 2 TB/day of streaming data to be processed and analyzed at an average cumulative bandwidth of ∼200Mbps. We propose that such compute and data intensive stream processing operations are well suited to run on public or private Cloud platforms that offer a scalable and flexible infrastructure [12].

However, the large bandwidth and compute requirements for generating, transmitting and processing power usage events at their maximum possible rate (O(events/min) per consumer), combined with the pay-as-you-go model of public Clouds, encourages investigation into optimal usage of resources by controlling the volume of power information that is transmitted. The use of heterogeneous networks that range from low bandwidth Power Line Carriers (PLC) (∼20Kbps) to 3G cellular networks (∼2Mbps) means that network bandwidth can be a scare resource even for the consumer. In addition, the cumulative bandwidth requirement can be bursty since smart meters send data independently, causing instantaneous bandwidth needs to peak. The dy-

ScienceCloud'11, June 8, 2011, San Jose, California, USA.

[1]FERC Assessment of Demand Response and Advanced metering, Staff Report, December 2008.

namic nature of the power grid means that using a static, low rate for publishing power usage events will prove to be an inadequate solution, say, during peak power load situations when events are required at low latencies to detect and correct power usage skews.

These trade-offs motivate the need for an adaptive stream rate control mechanism for generating power usage events. Unlike prior work, we use the application's operational needs as our metric for adapting the stream rate. Specifically, for our smart grid applications, we use the difference between the available power capacity at the utility and the current cumulative power usage by the consumers as our throttle function.

In this paper, we make two unique contributions:

1. We implement a smart grid stream processing pipeline, for early warning of peak power load, on top of a private Cloud platform, and
2. We design an adaptive stream rate control, driven by application needs, which consumes 50% lesser Cloud bandwidth than a constant rate pipeline.

We demonstrate the prototype smart meter application pipeline and the adaptive stream rate control using real power usage data from 50 buildings on the USC campus, which is a micro-grid testbed for the Los Angeles Smart Grid project.

The rest of the paper is organized as follows. We describe our smart grid stream processing pipeline and its deployment on a Eucalyptus private Cloud in Section 2; introduce the adaptive rate stream processing logic and implementation in Section 3; experimentally evaluate its performance in Section 4; discuss related work in Section 5; and present our conclusions in Section 6.

2. SMART GRID STREAM PIPELINE

Our smart meter stream processing pipeline (Figure 1) is designed to detect power consumers who individually or collectively exceed certain power usage limits set by the utility. This pipeline acts as early warning indicator of a peak load event. Each smart meter is a stream source whose tuples (or events) have attributes that identify the smart meter, the power used within a time interval, and the begin and end timestamps of the interval. Additional metadata present in the event are not relevant for this discussion. Each tuple is about 1KB in size.

The pipeline first checks if each individual power event reports usage that exceeds a certain threshold, U^{max}, defined by the utility. Crossing this will trigger a critical notification to a utility manager. Next, a condition checks if the user's consumption increases by more than 25% since their previous consumption, and this triggers a less critical notification. The pipeline then archives the tuple into a sink file and proceeds to compute a running sum of the daily usage by the consumer. Subsequently, the running average over a tumbling window is updated. These operations (shaded in light brown) are performed for each smart meter stream.

Next, the pipeline aggregates smart meter events from across all streams and calculates the cumulative consumer power usage within a 15 min tumbling window. This stream operator (shaded in dark green) calculates the total load on the utility. It can be used to alert the utility manager in case, say, the total consumption reaches 90% and >100% of available power capacity. Operators shown in dotted lines

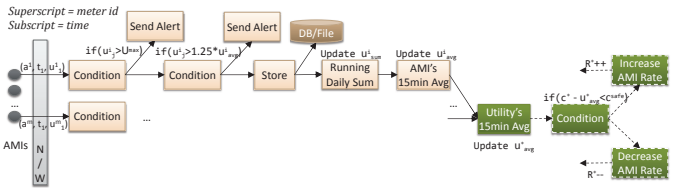

Figure 1: Stream processing pipeline used for continuous power use monitoring. Processing elements in dotted lines show the addition of throttle logic.

are not part of the application logic and form the adaptive throttling introduced later.

2.1 InfoSphere Streams

IBM InfoSphere Streams is a stream processing system to continuously analyze large volumes of streaming data for business activity monitoring and active diagnostics [5]. InfoSphere Streams' runtime environment consists of *stream instances* running on one or more hosts. Stream pipelines composed using the Stream Processing Application Declarative Engine (or SPADE) programming model are executed by the runtime environment.

The SPADE model supports *stream data sources* that continuously generate *tuples* (or events) that contain typed *attributes*. *Operators* take one or more streams as input, process the tuples and attributes, and produce one or more streams as output. Some supported operators are stream *source/sink* to read/write tuples from/to, *functors* to perform filtering and projection of tuples, and *aggregate* for grouping tuples. Streams can be consumed as sliding or tumbling windows of tuples. A *stream application* is a collection of operators connected by streams. Our smart grid pipeline is modelled using such SPADE operators.

Several stream sources are supported by SPADE [1], including TCP, UDP, files and HTTP URLs. The virtual smart meters used in our evaluation section use TCP source streams as input to the SPADE pipeline.

2.2 Eucalyptus Private Cloud

Eucalyptus [2] offers an open source Cloud fabric to deploy a private Infrastructure as a Service (IaaS) Cloud. The Eucalyptus fabric supports virtual machine (VM) images that conform to the Amazon Elastic Compute Cloud (EC2) specification[3] and can be instantiated on compute nodes. The fabric also provides shared storage services that are the syntactic and semantic equivalents of Amazon Simple Storage Service (S3) and Elastic Block Storage (EBS). Two reasons guide our selection of Eucalyptus for our initial prototype: (1) Utilities may prefer a private Cloud for reasons discussed in [12], and (2) Eucalyptus offer the ability to later migrate to the Amazon public Cloud with limited effort. The availability of a Eucalyptus Cloud at USC eased this choice.

2.3 Deploying InfoSphere Streams on Eucalyptus Cloud

InfoSphere Streams can run on a cluster but does not support out-of-the-box deployment to a Cloud platform. To instantiate a streaming environment on the Eucalyptus private

[2] `http://open.eucalyptus.com`
[3] `http://aws.amazon.com/ec2`

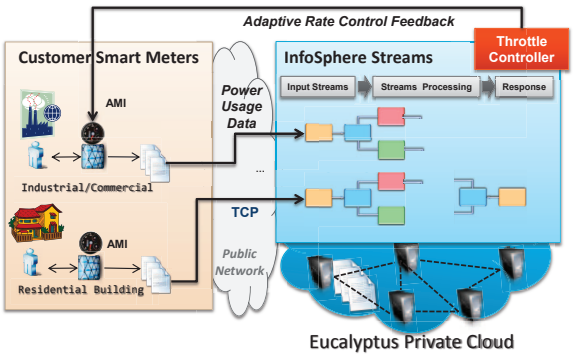

Figure 2: Stream processing architecture for Smart Meter data on Clouds.

Cloud, we customize VM images with InfoSphere Streams deployed on them. When the VM instances come online, the stream instances present in each VM can communicate with each other. This communication, however, is boot-strapped by a SPADE application that is configured with the list of named stream instances on specific VM hosts. We do not use any of the storage services of the Eucalyptus Cloud. In the absence of shared storage, VM images have a copy of the smart grid pipeline installed in them, available locally to the VM instances. The stream processing pipeline itself uses the network to exchange tuples between operators.

Figure 2 shows this architecture. Smart meters are present on the public internet and expose the power usage data as streams accessible over TCP sockets. Stream instances running on VMs in the private Cloud execute the smart grid pipeline to process the power events.

3. ADAPTIVE STREAM PROCESSING

Our pipeline (Figure 1) operates on power events generated by the smart meter, which determines the rate at which these tuples are published. This event rate is typically set statically when initially configuring the meter. The disadvantages of such a static rate are twofold. One, setting too high a rate at which power events are published will cause excess resources (bandwidth, compute VMs for stream processing) to be utilized. For example, generating power usage tuples every minute is unnecessary during the night when the utility has excess power capacity and it is less critical to detect overages by a specific consumer or a community immediately. While important to detect the breach for future diagnostics, it is not time sensitive.

Second, setting too low a static rate at which power events are published can cause the utility to miss detecting a breach of a power usage threshold during peak load periods. For e.g., during the midday, power usage by consumers edges closer to the available capacity of the utility. A freak heat wave can cause consumers to increase their power consumption beyond available capacity within a short duration and lead to brownouts and blackouts. Low latency warning is necessary during such circumstances. These factors motivate the need for smart meter stream rates that adapt dynamically to the current application need.

3.1 Rate Adaptation Logic

The adaptation logic that we use for the smart meter data analysis pipeline is based on the following intuitive premise: *as the total power usage within the utility approaches total available capacity, power usage events are required more frequently to detect/forecast a peak load event with low latency.* Our rate control logic subtracts the power usage across all consumers in the utility during the most recent time window from the total available capacity, and determines a stream rate that is inversely proportional to this value. i.e.

$$StreamRate \propto \frac{1}{(AvailableCapacity - CurrentUsage)}$$

In our implementation, we use stream rates that change in a step-wise manner that is proportion to the above equation. We also define static upper and lower bounds for the stream rates to ensure that we limit the minimum/maximum number of power events sent within a time period.

3.2 Adaptive Rate Control Implementation

The adaptive rate control (or throttle) logic is implemented as a trailing operator in the stream processing pipeline after the cumulative power usage across all consumers has been calculated. It is shown with a dotted outline in Figure 1. Unlike other operators that are provided by SPADE, we implement the throttle operator as an external *Java user defined operator* for two reasons. One, it allows finer control over the adaptation logic, such as the rate slabs that we use, and allows more complex policies to be used in the future. Second, it allows the rate control feedback to be sent to the smart meters through an independent control socket, to better manage open network sockets.

Our throttle operator extends the Java base-class, `AbstractOperator`, provided by SPADE. Its input stream receives tuples containing the current total power usage value from the preceding SPADE aggregate operator, and a target rate for the smart meter streams is evaluated. If the current stream rate matches the target stream rate then nothing further needs to be done, and the prevailing smart meter stream rate is maintained. Otherwise, the Java operator connects to the control port of each smart meter and indicates the new stream rate to be used by it. The smart meters update their internal timers to reflect the new rate and send subsequent power events accordingly.

4. EXPERIMENTAL EVALUATION

We evaluate the effectiveness of the adaptive stream rate control by its ability to conserve network bandwidth. To avoid disrupting production power systems, we use virtual smart meters that replicate the behavior and data generated by (hardware) smart meters on the USC campus micro-grid that is a testbed for the Los Angeles Smart Grid Project. Each virtual smart meter is a light-weight Java thread that generates power usage data for a single building in the micro-grid. We use 50 virtual smart meters that generate events that duplicate real measurements of historical power usage data collected for 50 buildings on campus at 1-minute intervals during a 48 hour period.

The virtual smart meter exposes two TCP server sockets: one is the data stream for transmitting power events to the stream pipeline, and the other is to receive adaptive rate control signals. For our experimental setup, we run all 50 meters on a 24-core server with AMD Opteron CPUs rated at 2.1GHz and 32GB of memory. The stream pipeline runs on a 128-core Eucalyptus v2.0 private Cloud with each VM

instance running CentOS v5 on 1 CPU core with 2GB memory. For our initial evaluation presented here, the SPADE application pipeline from Figure 1 is run on one InfoSphere Stream Instance v1.2.1 and uses just one Eucalyptus VM instance. The meters connect to the stream pipeline over 10Mbps Ethernet. This pipeline has 11 SPADE operators and three file sinks for each smart meter stream. With 50 virtual smart meters, the stream instance runs 550 SPADE operators and has 150 open file handles. The SPADE application is compiled on Eclipse using the IBM Streams Studio plugin and deployed on the VM image.

4.1 Constant Rate Stream Processing

As a baseline comparison, we run the stream processing pipeline with a static smart meter stream rate. To ensure timely completion of experiments, we perform runs at a 30x time compression where by the 48 hour (2880 minute) smart meter data is scaled to a 96 minute wall clock period, with an equivalent compression in the event latencies (and an expansion of the event rates). However, the power usage data values are unchanged.

Figure 3 plots the cumulative static event rate across all 50 smart meters on the primary Y axis (blue solid line) as the experiment time increases on the X axis. The secondary Y axis shows the cumulative power usage across all 50 buildings (red dotted line) that is computed by the pipeline over the real data timeline of 48 hours shown on the secondary X axis.

We make two observations from this plot. One, the network bandwidth consumed by the events is proportional to the area under the event rate curve (blue solid line). Two, the observed power usages (red dotted line) shows high temporal fidelity since we use a relatively high event rate of 25 events per minute (cumulative) across 50 smart meters. So while the area under the blue solid line is large when using a high static stream rate, this comes with an increased accuracy and lower latency for calculating cumulative power usage in the pipeline.

4.2 Adaptive Rate Stream Processing

In comparison, we run the stream processing pipeline with adaptive rate control, as proposed in Section 3. In this experiment too we use a compressed time scale for performing the runs.

Figure 4 shows the cumulative adaptive stream rates across all 50 smart meters on the primary Y axis (blue solid line) as the experiment time varies along the primary X axis. The secondary Y axis shows the cumulative power usage for all 50 buildings (red dotted line) that is calculated by the final aggregate operator in the pipeline over the real data duration of 48 hours shown on the secondary X axis.

The plot shows that the adaptive stream rate varies between 2 events/min to 50 events/min (cumulative), reflecting the upper and lower boundaries configured for the rate control. The adaptive stream rate plot (blue solid line) closely trails the calculated power usage plot (red dotted line) – as the observed power usage increases, getting closer to available power capacity, the stream rate increases, and vice versa. We see that there is a lag in the rate adaptation, manifest as a shift in the relative peaks of the power usage and adaptive stream rate curves. This is a consequence using a tumbling window for power usage aggregation rather than a sliding window. As a result, the rate adaptation is

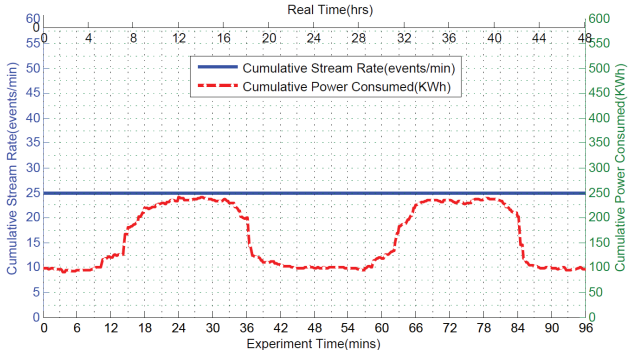

Figure 3: Cumulative static stream rate and observed power use by the stream processing system for 50 virtual smart meters over a 48 hour "real" time (96 min compressed "experimental" time).

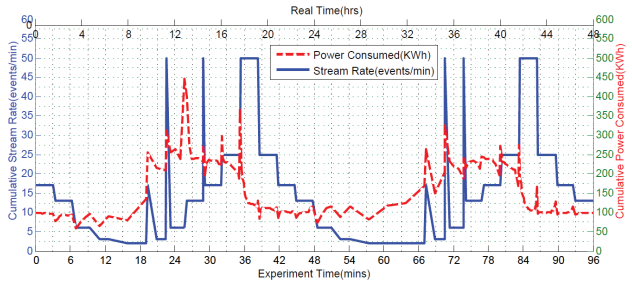

Figure 4: Cumulative adaptive stream rates and observed power use by the stream processing system for 50 virtual smart meters over a 48 hour "real" time (96 min compressed "experimental" time).

propagated only at the end of the tumbling window duration with a corresponding lag.

When we compare the area under the event rate curves (blue solid lines) for Figures 3 and 4, we see that the area under the unthrottled curve is almost double that of the throttled curve (320 vs. 163 "gridline blocks"). This shows that we are able to achieve a 50% reduction in network bandwidth usage relative to this particular static stream rate. The corresponding loss in power usage calculation accuracy for the throttled case is seen in the occasional spikes (e.g. 26min, 35min, 71min) and troughs (e.g. 7min, 11min, 49min) where the adaptive pipeline over/underestimates the cumulative power used.

5. RELATED WORK

Stream processing systems execute continuous queries on a moving window of data tuples, and have their roots in sensor network data processing. Streaming systems like TelegraphCQ [6] and Borealis [2] have made seminal contributions to this field, and a SQL-inspired Continuous Query Language (CQL) [3] has been proposed. Stream processing has also been studied in the OGSA-DAI project for Grid computing [10] and for environmental data streams on Clouds [15].

Streaming systems have used resource constraints to modulate both stream rates and their processing. [4] uses tuple dropping ("load shedding") as a means to ensure maximal

output processing rate when computational resources are limited. Another approach [13] addresses short-comings of frequency-based random input sampling and load shedding by proposing an age-based model for sliding window joins of multiple streams in a memory constrained environment. The GrubJoin algorithm in [8] performs optimal window harvesting for m-way stream joins using CPU aware load shedding. Rate control techniques for online multimedia applications have also been identified [14]. All of these approaches use computational, memory or network resource constraints for load shedding rather than an application's quality of service needs. The adaptive stream rate technique we propose uses application sensitive policies to perform stream throttling that does not drop tuples (causing data loss), but instead aggregates tuple values at the stream source and limits the rate at which the tuples are generated.

TCP congestion control uses a linear increase and exponential backoff algorithm [9] to maintain packet flow equilibrium at the transport layer. These algorithms too use transport level constraints rather than application level knowledge (such as deep packet inspection) to control flow rates.

While stream processing is a natural fit for emerging smart grid applications [4], no work to our knowledge leverages the scalability offered by Cloud platforms for deploying such smart grid analysis and detection pipelines. Our deployment of a streaming application on the Eucalyptus private Cloud provides early guidance for smart grid eEngineering applications to leverage Cloud platforms.

6. CONCLUSION AND FUTURE WORK

We have described our implementation of a stream processing pipeline for latency sensitive smart grid applications using a private Cloud platform. Our initial evaluation of adaptive rate control reveals its benefits for conserving Cloud resource usage (and hence costs) while satisfying the smart grid application's latency needs.

Our initial results hold promise for further research into advanced rate control algorithms for cyber-physical systems, such as traffic and weather analysis, where large-scale stream processing is vital. As future work, we plan to evaluate the scalability of the stream processing system with increasing number of VM instances, with the eventual goal of scaling up to 1.4 million smart meter streams that is expected in the Los Angeles Smart Grid. Both throughput and latency of the pipeline need to be measured as the stream rates adapt. Additional factors like availability of compute resources (# of VMs), throughput of the stream processing system, and cost trade-off between Cloud resource usage and power conserved can be incorporated into the throttle policy.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000192. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. The authors would like to thank Dr. Anand Ranganathan from IBM T. J. Watson Research Lab for discussions and assistance with IBM InfoSphere Streams.

[4]Oracle Solutions for Utilities, Oracle Corp., 2010.

8. REFERENCES

[1] IBM InfoSphere Streams: Programming Model and Language Reference, Version 1.2.1. Technical report, IBM Corp., 2010.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR*, pages 277–289, 2005.

[3] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *VLDB Journal*, 15:121–142, 2006.

[4] A. M. Ayad and J. F. Naughton. Static optimization of conjunctive queries with sliding windows over infinite streams. In *SIGMOD*, 2004.

[5] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. IBM InfoSphere Streams for scalable, real-time intelligent transportation services. In *SIGMOD*, 2010.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[7] E. A. Feinberg and D. Genethliou. *Applied Mathematics for Restructured Electric Power Systems*, chapter Chapter 12: Load Forecasting, pages 269–285. Springer US, 2005.

[8] B. Gedik, K.-L. Wu, P. S. Yu, and L. Liu. A Load Shedding Framework and Optimizations for M-way Windowed Stream Joins. In *ICDE*, 2007.

[9] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comp. Comm. Rev.*, 18:314–329, 1988.

[10] C. S. Liew, M. P. Atkinson, J. I. van Hemert, and L. Han. Towards optimising distributed data streaming graphs using parallel streams. In *DIDC*, pages 725–736, 2010.

[11] Y. Simmhan, S. Aman, B. Cao, M. Giakkoupis, A. Kumbhare, Q. Zhou, D. Paul, C. Fern, A. Sharma, and V. Prasanna. An informatics approach to demand response optimization in smart grids. Technical report, Computer Science Dept., USC, 2011.

[12] Y. Simmhan, M. Giakkoupis, B. Cao, and V. K. Prasanna. On using cloud platforms in a software architecture for smart energy grids. In *CloudCom*, 2010.

[13] U. Srivastava and J. Widom. Memory-limited execution of windowed stream joins. In *VLDB*, 2004.

[14] B. Vandalore, W. chi Feng, R. Jain, and S. Fahmy. A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia. *Real-Time Imaging*, 7(3):221–235, 2001.

[15] D. Zinn, Y. Simmhan, M. Giakkoupis, Q. Hart, T. McPhillips, B. Ludäscher, and V. K. Prasanna. Towards reliable, performant workflows for streaming-applications on cloud platforms. In *CCGrid*, 2011.