Using Data Stream Management Systems to analyze Electric Power Consumption Data

Talel Abdessalem[1], Raja Chiky[1], Georges Hébrail[1,2], Jean Louis Vitti[1]

[1] ENST Paris - CNRS UMR 5141, 46, Rue Barrault, 75013 Paris Cedex 13, France
Email: {abdessalem, chiky, hebrail, vitti}@enst.fr
[2] EDF R&D - Département ICAME, 1, Av. du Général de Gaulle, 92140 Clamart, France
Email: georges.hebrail@edf.fr

Abstract. With the development of AMM (Automatic Metering Management), it will be possible for electric power suppliers to acquire from the customers their electric power consumption up to every second. This will generate data arriving in multiple, continuous, rapid, and time-varying data streams. Data Stream Management Systems (DSMS) - currently available as prototypes - aim at facilitating the management of such data streams. This paper describes an experimental study which analyzes the advantages and limitations of using a DSMS for the management of electric power consumption data.

Keywords: Electric power consumption, Data Stream Management System, electric load curve, continuous queries.

1 Introduction and motivation

The forthcoming deployment of AMM (Automatic Metering Management) infrastructures in Europe will enable a more accurate observation of a large number of customers. Electric power consumption will be possibly measured at a rate up to one index per second. These measures are useful for operations such as billing, data aggregation and consumption control. A traditional database system (DBMS Data Base Management System) could be used to manage this information. However, methods used currently in DBMS are not adapted to data generated from AMM in a streaming way. AMM generates an overwhelming amount of data arriving in multiple, continuous, rapid, and time-varying data streams. Several Data Stream Management Systems (DSMS) have been developed these last years to meet these needs. The role of these systems is to process in real time one or more data streams using *continuous* queries.
We performed an experimental study, on two public domain and general purpose DSMS prototypes (STREAM and TelegraphCQ), to analyze the advantages and limits of using such a system. Some typical queries of electric power consumption analysis were defined: experiments and results are reported in this paper.
The paper is organized as follows. Section 2 gives a brief introduction to DSMS's in general and presents in particular STREAM and TelegraphCQ. Section 3 presents the experimental study, and results are reported in Section 4. In Section 5, we conclude this paper and give an outlook to our ongoing and future research in this area.

2 Data Stream management Systems

Data Stream Management Systems [3] are designed to perform continuous queries over data stream. Data elements arrive on-line and stay only for a limited time period in memory. In a DSMS, continuous queries evaluate continuously and incrementally arriving data elements. DSMS use windowing technics to handle some operations like aggregation as only an excerpt of a stream (window) is of interest at any given time. A window may be physically defined in terms of a time interval (for instance the last week), or logically defined in terms of the number of tuples (for example the last 20 elements).

Several DSMS prototypes have been developed within the last five years. Some of them are specialized in a particular domain (sensor monitoring, web application, ...), some others are for general use (as STREAM [2] and TelegraphCQ [4]).

STREAM

STREAM (STanford stREam datA Management) is a prototype implementation of a DSMS developed at Stanford University [2]. This system allows the application of a great number of declarative and continuous queries to static data (tables) and dynamic data (streams). A declarative query language CQL (Continuous Query Language), derived from SQL, is implemented to process continuous queries. STREAM uses the concept of sliding windows over streams. These windows are of three types: time based window ([range T] where T is a temporal interval), tuple based window ([Row N] where N is a number of tuples), and partitioned sliding window ([Partition By $A_1, ..., A_k$ Rows N]) similar to Group BY in SQL.

TelegraphCQ

TelegraphCQ is a DSMS developed at University of Berkeley [4]. TelegraphCQ is built as an extension to the PostGreSQL relational DBMS to support continuous queries over data streams. The format of a data stream is defined as any PostGreSQL table in the PostgreSQL's Data Definition Language, and created using CREATE STREAM. TelegraphCQ is a mode of PostGreSQL execution. It supports the creation of archived and unarchived streams that are fed with external sources using stream specific wrapper. Queries that contain streams support the SELECT syntax and include extra predicates RANGE BY, SLIDE BY and START AT to specify the window size for stream operations. Each stream has a special time attribute that TelegraphCQ uses as the tuples timestamp for windowed operations.

3 Experiments

We installed STREAM and TelegraphCQ V2.1, and tested them on a data file of indices of electric power consumption measured during 150 days. These indices were obtained from three electric meters in different geographical cities, and our experiments can be extended to a larger number of meters. Each meter sends a tuple every 2 seconds. Each tuple is composed of a set of attributes such as meter identifier, time, date, index shown by the meter, and some additional information. The difference between two indices at two different instants gives the electric power consumption between these two instants.
For all queries, we assume a unique input stream Stream_index merging the streams of the three meters and defined as follows:

```
Stream_index (year INT, month INT, day INT, h INT, m INT, sec INT, meter char(12), index INT)
```

Stream_index is the name identifying the data stream. The date is specified with three attributes: 'month','day','year' ; and time by three other attributes: 'h', 'm', 'sec' (hour, minute, second). The 'meter' attribute identifies the meter and 'index' indicates the measured index (in kWh).
Electric power consumption analysis requires continuous queries over stream. Among the possible queries, we study here three queries which are representative of this kind of application.

Q1. Consumption of the last 5 minutes -minute by minute- grouped by meter, or by city.

- STREAM: a CQL query can be assigned a name to allow its result to be referenced by other queries. This feature (similar to *view* definition in SQL) allows us to express subqueries which are not allowed in CQL. We solve query *Q1* using the following steps:

1. Min_index: compute the minimum index grouped by meter and minute
   SELECT year, month, day, h, m, meter, min(index) as minindex
   FROM Stream_index
   GROUP BY year, month, day, h, m, meter;

2. Cons_Minute_60 : compute the consumption from hh:59 to hh+1:00 (For example: from 07:59 to 08:00).
   SELECT c1.year as year_beg, c1.month as month_beg, c1.day as day_beg, c1.h as h_beg,
   c1.m as m_beg, c1.meter, c1.minindex as minindex_beg, c2.year as year_end,
   c2.month as month_end,c2.day as day_end, c2.h as h_end, c2.m as m_end,
   c2.minindex as minindex_end, c2.minindex-c1.minindex as Cons
   FROM Min_index as c1, Min_index as c2
   WHERE c1.month=c2.month AND c1.day=c2.day AND c1.year=c2.year
   AND c1.h=(c2.h-1) AND c1.m=59 AND c1.meter=c2.meter AND c2.m=0

3. Cons_Minute_01_59: compute the consumption from the first minute and minute 59 of each hour (For example: from 07:00 to 07:59 minute by minute)
   SELECT (*similar to Cons_Minute_60*)
   FROM Min_index as c1, Min_index as c2
   WHERE c1.month=c2.month AND c1.day=c2.day AND c1.year=c2.year
   AND c1.h=c2.h AND c1.m=(c2.m-1) AND c2.m>0 AND c1.meter=c2.meter

4. Stream_Cons: set the union of the consumptions calculated by the two preceding queries as a single stream.
   ISTREAM(Cons_Minute_60 UNION Cons_Minute_01_59);

5. Cons_per_M: compute the consumption for the last 5 minutes -minute by minute- grouped by meter.

   SELECT year_beg, month_beg, day_beg, h_beg, m_beg, Stream_Cons.meter, Tablecity.city,
   minindex_beg, year_end, month_end, day_end, h_end, m_end, minindex_end, Cons
   FROM Stream_Cons[range 150 seconds], Tablecity
   WHERE Stream_Cons.meter = Tablecity.meter;

   We join here Stream_Cons with TableCity (containing meters' identifiers and the cities where they are) to get the meter's city. We parameter a window by 150 second because a tuple arrives every 2 seconds, whereas it is indexed by a timestamp at each second.

6. The sum of consumptions grouped by city.
   SELECT year_beg, month_beg, day_beg, h_beg, m_beg, city, sum(Cons)
   FROM Cons_per_M
   GROUP BY year_beg, month_beg, day_beg, h_beg, m_beg, city;

- TelegraphCQ: We created a stream Elec.stream and assigned a wrapper to this stream. Each tuple is indexed by a timestamp 'tcqtime' specifying the creation time of the tuple, which is assumed to be monotonically increasing with a format like 'YYYY-MM-DD hh:mm:ss'.
  To solve this query, we compute the minimum of the indices for each minute. Then, we make a selfjoin of the result obtained with a delay of one minute. Thanks to PostGreSQL's operators for date management, we can easily compute the consumption between two instants delayed by one minute.
  Nested queries are not supported by TelegraphCQ, but we can use the WITH construction as an alternative. Selfjoins are not allowed either, we were constrained to create two identical streams to join them. We created the first stream Elec.minstream1 with the minimum of indices by minute using a sliding window parameterized by an interval of 6 minutes, and a second identical stream Elec.minstream2 but windowed on 5 minutes. This enables us to make the difference

between these two streams with a delay of one minute to get the consumption during the 5 last minutes minute by minute.

```
CREATE STREAM Elec.minstream1 ( meter varchar(12), minindex INTEGER,
                                tcqtime TIMESTAMP TIMESTAMPCOLUMN )
                                TYPE UNARCHIVED;
CREATE STREAM Elec.minstream2 ( as Elec.minstream1 ) ;

WITH
Elec.minstream1 AS ( SELECT    meter,min(index),DATE_TRUNC('minute', tcqtime)
                     FROM      Elec.stream [RANGE BY '6 minutes' SLIDE BY '1 minute'
                               START AT '2003-12-04 07:50:00']
                     GROUP BY meter, DATE_TRUNC('minute', tcqtime) )
Elec.minstream2 AS ( SELECT ... as Elec.minstream1 with RANGE BY 5 minutes )
(
SELECT f1.meter, f1.minindex, f1.tcqtime, f2.minindex, f2.minindex-f1.minindex, f2.tcqtime
FROM   Elec.minstream1 as f1 [RANGE BY '1 minute' SLIDE BY '1 minute'
       START AT '2003-12-04 07:50:00'],
       Elec.minstream2 as f2 [RANGE BY '1 minutes' SLIDE BY '1 minute'
       START AT '2003-12-04 07:50:00']
WHERE f1.meter = f2.meter AND f1.tcqtime = (f2.tcqtime - interval '1 minute'));
```

To get the sum of consumption by city, we transform the last query into a stream, we call it Elec.streamcons. Streams Elec.minstream1 and Elec.minstream2 are computed with sliding windows of 1 minute. We apply a join operation between Elec.streamcons and table TableCity windowed to 5 minutes.

Q2 - Historical consumption -minute by minute- grouped by meter, or by city, starting from a fixed point.

- STREAM: We apply the same queries as previously without windowing. At the last query (Cons_Per_M), we filter to keep only the dates and time exceeding the fixed starting point.
- TelegraphCQ: This query was easy to solve thanks to START AT construction. It filters the tuples starting from a fixed point. We followed the same steps as the preceding query.

Q3 - Alarm -hour by hour- at exceeding a 'normal' consumption depending on the temperature.

This query gives an alarm when a consumption exceeds 10% of a 'normal' consumption over a period from midnight to the current hour. The alarm is given after midday. 'Normal' consumption is given hour by hour in a table consNormal for a temperature of 20°C.

- STREAM: There is a basic deficiency in the STREAM prototype that makes impossible to solve this query. There are no operators for standard date format management. Indeed, to perform this query, we would have had to treat several cases: the passage from a day to the next (example: from 12/02/2004 23:00 to 13/02/2004 00:00), the passage from a month to the next (example: from 31/01/2004 23:00 to 01/02/2004 00:00), the passage from a year to the next,...
- TelegraphCQ: We use a stream Elec.temperature that gives the temperature recorded each hour during 150 days. We apply a join operator between table consNormal and stream Elec.temperature in order to build a stream of normal consumption Elec.streamconsnormal which depends on temperature (a gradient approach is used).

  To solve this query, we apply a sliding window parametered by a time interval of 24 hours to the Elec.stream consumption stream , we calculate the sum of consumption hour by hour during 24h, and we filter the consumption taken after midday. We follow the same steps for the stream of normal consumption to get a stream of normal consumption cumulated hour by hour from midday. Finally, we compare the two streams to filter consumption which exceeds of 10 % normal consumption. The query is not reported in this paper by lack of room.

4 Synthesis

As for the clarity and easiness of writing queries, TelegraphCQ is much better than STREAM, thanks to its ability of defining landmark physical windows combined with the date management facilities provided by the PostGreSQL embedded system. Moreover, TelegraphCQ allows to reuse results of queries as a stream. It allows also redirecting the output of queries into a file to be stored in a specified format. Queries can be added dynamically when others are being executed, which is not possible with STREAM.
STREAM is characterized by its graphical interface with a graphical query and system visualizer. In fact, it's important for users (administrators) to have the ability to inspect the system while it is running and to understand how the continuous queries are managed. For instance, it was possible to see in STREAM that the aggregation operations of query 1. were performed incrementally without keeping the whole stream in memory.
Comparing our expectations and requirements, we can state that the recent TelegraphCQ prototype is quite useful for online analysis of electrical power consumption data, mainly because we solved easily all our defined queries. Though the performance of TelegraphCQ appears to be much better on our queries than in STREAM, we would like to have an idea of the system performance and control the amount of memory being used to process our queries.

5 Conclusion and Future Work

We have shown that two DSMS's allows to treat individual electric power consumption data arriving in a continuous and fast rate stream. According to our experiments, it appears that TelegraphCQ is better adapted to our needs than STREAM. However, TelegraphCQ does not offer tools to view the structure of query plans, nor monitor system resources and inspect the amount of memory being used. The comparison between these two DSMS's is mainly functional and does not take into account system performance. A follow-up study is to analyze the performance of TelegraphCQ when scaling up to larger streams figuring a larger number of meters.
Some research teams work on distributed DSMS's, like the Borealis [1] proposal. Such distributed DSMS's should be more appropriate to deal with AMM data which are distributed in nature. We are going to test the Borealis system in the close future, in addition to a following of future releases of TelegraphCQ or commercial follow-up systems.

References

1. D.J.Abadi, Y.Ahmad, M.Balazinska, U.Cetintemel, M.Cherniack, J.H.Hwang, W.Lindner, A.S.Maskey, A.Rasin, E.Ryvkina, N.Tatbul, Y.Xing, and S.Zdonik (2005). The Design of the Borealis Stream Processing Engine. 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, CA, January 2005.
2. Arasu A., Babcock B., Babu S., Cieslewicz J., Datar M., Ito K., Motwani R. , Srivastava U., Widom J.(2004). STREAM: The Stanford Data Stream Management System, Book chapter.
3. Babcock B., Babu S., Datar M., Motwani R., and Widom J. (2002). Models and issues in data stream systems. In Symposium on Principles of Database Systems, pages 116. ACM SIGACT-SIGMOD.
4. S.Chandrasekaran, O.Cooper, A.Deshpande, M.J.Franklin,J.M.Hellerstein, W.Hong, S.Krishnamurthy, S.R.Madden, V.Raman, F.Reiss, M.A.Shah (2003). TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR 2003.