Identifying the Challenges for Optimizing the Process to Achieve Reproducible Results in E-science Applications

Mohammad R. Huq
University of Twente
Enschede, The Netherlands.
m.r.huq@utwente.nl

Andreas Wombacher
University of Twente
Enschede, The Netherlands.
a.wombacher@utwente.nl

Peter M. G. Apers
University of Twente
Enschede, The Netherlands.
p.m.g.apers@utwente.nl

ABSTRACT

One of the major requirements for e-science applications handling sensor data, is reproducibility of results. Several optimization and scalability problems exist where the reproducibility of results remains guaranteed. Firstly, various data streams need to be coordinated to optimize the accuracy and processing of the results. Secondly, because of the high volume of streaming data and a series of processing steps to be performed on that data, demand for disk space may grow unacceptably high. Lastly, reproducibility in a decentralized scenario may be difficult to achieve because of data replication. This paper introduces and addresses these challenges which arise for optimizing the process of achieving reproducibility of results.

Categories and Subject Descriptors

E [**Data**]: Miscellaneous—*Sensor Data*

General Terms

Theory

Keywords

E-science applications, Optimization, Reproducibility, Sensor data

1. INTRODUCTION

Generally, sensor data includes both *streaming* and *manually sampled* [4, 9] data. In most e-science applications, sensor data are acquired and processed to higher level events used in applications for decision making and process control. In e-science applications, it is important that the origin of processed data can be identified to understand the semantics of the event and to validate the correctness of the generated events. Most significantly, researchers often need to reproduce the previous results in e-science applications.

Data provenance can be used to reproduce results. Data provenance [5] documents the origin of data by explicating the relationship among input data, algorithm and the processed data. Provenance can be defined either at the tuple level or at the relation level known as fine grained and coarse grained data provenance respectively [5].

Fine grained data provenance can achieve reproducible results. In stream data processing, we often use window-based techniques to execute a query on the set of finite data. If two subsequent windows overlap, a data tuple is processed in several query executions resulting in several provenance information. Thus, the storage cost for fine grained data provenance is linear with the number of data tuples and the re-use of data tuples which is expensive.

Maintaining coarse grained data provenance does not require as much extra disk space as fine grained data provenance does. However, it cannot achieve reproducible results if the database state changes. Maintaining coarse grained data provenance and applying a temporal data model can achieve the same as fine grained data provenance does, with reduced storage costs [13].

However, several optimization and scalability problems exist in the process of achieving reproducible results. Firstly, sensors may have different sampling rate. In addition to that, streaming data has propagation delay associated with it. Sometimes, due to the longer processing chains, tuples may arrive for a particular processing step only after a considerable period of time.

Secondly, sensors are sending data tuples in a streaming fashion and these data tuples may be processed through a sequence of processing steps. The result obtained at each processing step is stored in a persistent storage to ensure reproducibility. Due to the high volume of streaming data and several processing steps, storing all the results in a persistent storage requires a lot of storage space.

Thirdly, in a decentralized scenario, streaming and manually sampled data can be stored in several databases distributed over the globe. Sometimes, a particular data tuple may be replicated from one database to another which creates multiple versions of that tuple. Therefore, maintaining a consistent database state globally becomes difficult.

This paper is organized as follows. In Section 2, we provide a detailed description of our motivating scenario. In Section 3, we state our research questions, discuss each research question in detail, explaining the scope for research as well as highlighting some potential solutions. Then, we discuss related work in Section 4. Finally, we conclude by stressing novelty of these problems and provide an outline of some future work.

PIKM'10, October 30, 2010, Toronto, Ontario, Canada.

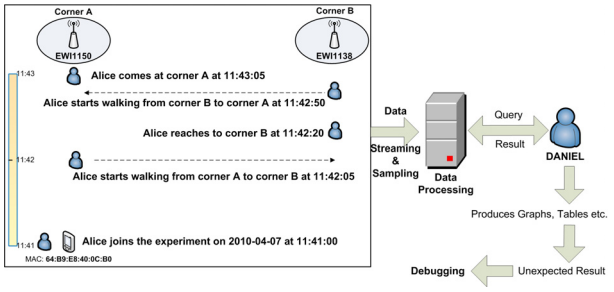

Figure 1: Bluetooth localization scenario in SensorDataLab

2. SCENARIO

A Bluetooth localization scenario built in the SensorDataLab [1] is our motivating scenario. Here, we have installed several Linksys NSLU2 systems which are used for acquiring signal strength measurements of all Bluetooth devices and reports detected devices via a UDP packet to the data processing system. In Figure 1, we have two NSLU2 systems which are represented as EWI 1138 and EWI 1150.

In addition to these UDP packets, sampled data is required, for example, representing the deployment location of NSLU2 device. Further, the mapping of MAC address of a handheld device to an actual person at a specific point in time is documented and made available as sampled data. The data processing unit allows to correlate sensed and sampled data and provides a query interface to access the data.

In Figure 1, the user Alice is joining the experiment on 2010-04-07 at 11:41. She is carrying a Bluetooth enabled handheld device. After a while, Alice takes a walk through the corridor of the laboratory. She starts from the corner A where the system EWI 1150 is installed at around 11:42:00 and goes to the other corner B where EWI 1138 is deployed. Alice reaches at corner B at around 11:42:20. At corner B, Alice waits for approximately 25-30 seconds and then turns back. She again starts walking to corner A. Approximately at 11:43:05, Alice comes at corner A and stops.

The supervisor of Alice, Daniel wants to prepare some graphs and tables based on the experiment. To serve his purpose, Daniel collects data pertaining to the Alice's movement. He uses the Sensor Data Web [2] platform for collecting, processing and publishing data. *Processing elemets (PE)* are used to execute any operation in this platform. Some of them are known as *source processing elements (Source PE)* which are used to acquire data tuples from various sources. All the PEs generate a *view* containing data tuples after executing the respective operation. These *views* can be used by another PE as the input dataset. All tuples in *views* contains two extra temporal attributes which are added to ensure the timestamp-based database versioning.

- **valid time**: refers the point in time when a measurement from sensors has been sensed or a sample has been taken.

- **transaction time**: is the point in time when a tuple is inserted into the database.

[1] http://www.sensordatalab.org/wiki/index.php5/Main_Page
[2] https://sourceforge.net/projects/sensordataweb/

3. RESEARCH QUESTIONS

Several optimization and scalability problems can be identified which are associated with the process of achieving reproducible results. These are:

- *RQ-1*: How to **coordinate various data streams** in order to optimize the processing of results?

- *RQ-2*: How to **optimize storage space requirement** within a workflow?

- *RQ-3*: How to **keep a database state consistent globally** in a decentralized scenario?

3.1 Coordination of Various Data Streams

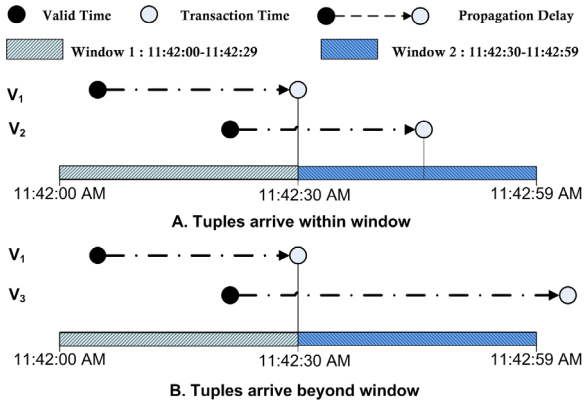

Figure 2: Different cases describing coordination problem

Sensors may produce data at different rates. Moreover, based on the network characteristics, tuples may have *propagation delay*. Figure 2 shows two cases: A and B. Tuples are initially stored in *view* V_i and V_j. These views are generated by source processing elements PE_i and PE_j respectively. These tuples from two different views will then be joined together. The window size associated with this *join* query is 30 seconds.

- **Case A**: In Figure 2.A, one tuple of V_1 and one tuple of V_2 are considered. Since both tuples were originated at *window 1* and also arrive for processing during *window 2*, they will be processed normally.

- **Case B**: Here, a tuple of V_1 and a tuple of V_3 are considered (see Figure 2.B). The tuple from V_1 arrives at the start of the *window 2* but the tuple from from V_3 arrives after the window expires because of different sampling rates and added propagation delay. Because of the delayed arrival of the tuple from V_3, these two tuples may not be processed together by *join* operation.

If two tuples which originated from different streams at the same window period do not arrive within the same processing window, the outcome of join operation will not be what the user expected (see Figure 2.B). That is, if the difference of propagation delay (*transaction time* − *valid time*) of these two tuples is greater than the window size, they will not be processed together which will result into unexpected

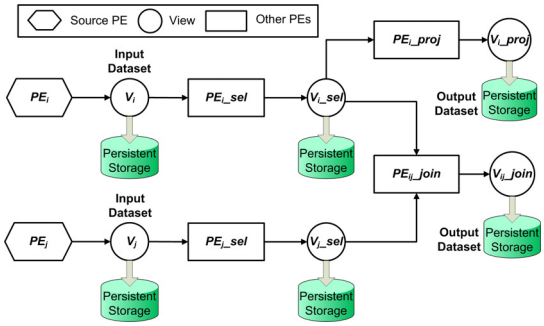

Figure 3: Maintaing persistent storage for each step of a workflow

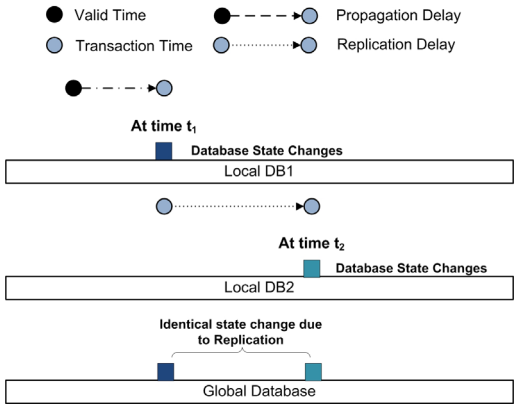

Figure 4: Replication of tuples results in identical, multiple state changes

outcome. Therefore, tuples from various sources need to be coordinated to increase the accuracy of the result. Based on this problem, our research question is **how to coordinate various data streams to optimize the processing of results**.

We can solve this problem in two different ways. The first alternative might be not to synchronize different streams, but instead accept the possibility of error in the result set. The second alternative solution is to coordinate the two streams to optimize the processing result thus to provide more accurate result to the user. This can be accomplished by delaying the execution of the processing of specific windows. In this approach, the query response time will be longer but it increases the level of accuracy of the result. Thus the design decision to solve this problem is a trade-off between response time and accuracy of the result.

3.2 Optimization of Storage Space

A set of data tuples may go through different processing steps to achieve the final outcome. Each processing step provides a *view* as an output which can be considered as a persistent storage where the resultant tuples are stored. Thus, this approach requires a lot of storage space to maintain persistent storage for each processing step.

Figure 3 shows that for a particular workflow, persistent storage are maintained for each view. Two different data streams acquired by processing elements PE_i and PE_j are considered and joined together. To complete *join*, there are some intermediary processing elements. As for example, PE_i_sel only selects data tuples from view V_i based on user given conditions. For each intermediary processing element, a view is given as output and it is maintained in a persistent storage.

Since the volume of streaming data is expected to be very high, keeping all the views persistent and store the tuples in the database is costly. Therefore, we state our research question as **how to optimize storage space requirement within a workflow**.

To overcome the aforementioned problem, our proposal is not to make all the views persistent. In figure 3, processing element PE_i_sel and PE_j_sel select tuples from view V_i and V_j respectively. Since the output of these two processing elements can be easily derived from previous views and associated query, we may not create any persistent storage for view V_i_sel and V_j_sel. Instead, we only keep the transaction time of the tuples and a pointer to point to the original tuple in previous view V_i and V_j respectively.

Since we maintain relation-based provenance, we can reproduce the result with this solution. This approach also optimizes the storage space requirement of the overall processing steps. However, it may take longer response time for regenerating views which are not persistent. Therefore, depending on the application, we need to set a trade-off between the query response time and storage space required by processed tuples.

3.3 Keeping Database State Consistent Globally

Decentralization of the database is a common phenomena, found especially in e-science applications. However, sometimes data may be replicated from one DB to another to increase data availability.

In figure 4, we have two local databases known as Local DB1 and Local DB2. A data tuple is entered into Local DB1 which causes database state change. Furthermore, this state change in Local DB1 will be also reflected in the global database state. After a while, the same data item is replicated from Local DB1 to Local DB2. This replication process takes some time and after successful replication it causes a state change in Local DB2 which also changes the state of the global database.

If any query is then executed on the global database state in the presence of multiple copies of the same tuple, it may be difficult to retrieve original global database state because of having different *transaction time* in each copy of that tuple. For a given timestamp which is larger than the *transaction time* of the original copy of the replicated tuple, we will get multiple copies of the same data item in global database. Therefore, based on this problem, we mention our research question in such way: **How to keep a database state consistent globally in a decentralized scenario**.

To solve this problem, we can introduce an operator Ξ, which can align different local database states to a consistent global database state. This operator Ξ will be used to transform the transaction time of the secondary copy to the transaction time of the primary copy thus avoiding the conflicts in the global database state. Another approach is to introduce a new temporal attribute *global transaction time* which will hold the value of the transaction time of the primary copy. However, this solution will increase storage overhead.

4. RELATED WORK

Sensor data are transmitted in form of data stream which is different than traditional data. To process sensor data, continuous queries are defined which use push-based query semantics. Recent interest in this field has generated several projects facilitating the execution of continuous queries [6, 7, 8, 14].

In this paper, we have presented three key research questions related to optimizing the process of achieving reproducibility of results for sensor data especially streaming data. We will keep our discussion limited only focusing on the closely related existing work.

STREAM [3] is a data stream processing project whose focus is on computing approximate results and on understanding the memory requirements of posed queries. It has proposed *Synopsis data structure* which is an approximate data structure rather than an exact representation and is used to optimize storage requirement. However, this technique cannot guarantee reproducibility.

The Aurora [2] system manages data streams for monitoring applications. It has a *resample box* which is used to coordinate between different data streams. Aurora minimizes storage need for data tuples by using *load shedding*. It can only optimize the storage need for a single processing step; not for the entire workflow.

Borealis [1] is an extended version of Aurora and also includes distributed functionality. It can dynamically revise the query results if the previously generated result is imperfect or partial. Moreover, it optimizes the distributed processing of sensor data in terms of processing speed and memory consumption. However, it does not address the notion of globally consistent database state in decentralized scenario.

System S [10] is a large-scale, distributed data stream processing middleware. It introduces *barrier* operation which is used as a synchronization point. It does not address the optimization of storage requirement and the maintenance of globally consistent database state.

In [15], authors have proposed an approach for reliable event streams in distributed data flow. Distributed data flow is a set of events in distributed application or protocol which actually invokes methods from one software components to the other. However, they have no mechanism included in their approach which can guarantee the reproducibility of those events.

Data reduction techniques are useful to decrease the response time but it sacrifices the level of accuracy. Some general techniques for data reduction and synopsis construction includes sampling, histograms, wavelets and other sketch based techniques. These techniques also minimize storage need but cannot achieve reproducibility because of data loss [11, 12, 16].

5. CONCLUSION AND FUTURE WORK

In this paper, we have introduced several research questions related to the optimization of the processing to achieve reproducibility of results in e-science applications. We have also presented some potential approaches that might be used to solve the aforementioned challenges. In future, we would like to provide concrete methods and procedures for the solution of each problem.

6. REFERENCES

[1] D. Abadi, et al. The design of the borealis stream processing engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA*, pages 277–289, 2005.

[2] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the Symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[4] D. Brus and M. Knotters. Sampling design for compliance monitoring of surface water quality: A case study in a polder area. *Water Resources Research*, 44(11):95 – 102, 2008.

[5] P. Buneman and W. C. Tan. Provenance in databases. In *Proc. of International conference on Management of Data*, pages 1171–1173, USA, ACM, 2007.

[6] S. Chandrasekaran, et al. TelegraphCQ: continuous dataflow processing. In *Proc. of International conference on Management of Data*, page 668. ACM, 2003.

[7] S. Chandrasekaran and M. Franklin. PSoup: a system for streaming queries over streaming data. *The VLDB Journal*, 12(2):140–156, 2003.

[8] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. *ACM SIGMOD Record*, 29(2):379–390, 2000.

[9] J. De Gruijter, D. Brus, and M. Bierkens. *Sampling for natural resource monitoring*. Springer Verlag, 2006.

[10] B. Gedik, H. Andrade, K. Wu, P. Yu, and M. Doo. SPADE: The System S declarative stream processing engine. In *Proc. of International conference on Management of Data*, pages 1123–1134. ACM, 2008.

[11] P. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. of International conference on Management of Data*, pages 331 – 342. ACM, 1998.

[12] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 171–182. ACM, 1997.

[13] M. R. Huq, A. Wombacher, and P. Apers. Facilitating fine grained data provenance using temporal data model. In *DMSN*, VLDB workshop, 2010.

[14] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. of International conference on Management of Data*, pages 49–60. ACM, 2002.

[15] K. Ostrowski, K. Birman, D. Dolev, and C. Sakoda. Implementing reliable event streams in large systems via distributed data flows and recursive delegation. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–14, USA, 2009.

[16] J. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.