A HYBRID PULL-PUSH SYSTEM FOR NEAR REAL-TIME NOTIFICATIONS ON SENSOR WEB

C.Y. Huang, S. Liang *

Dept. of Geomatic Engineering, University of Calgary, 2500 University Drive NW, Calgary, Alberta, T2N 1N4 Canada - (huangcy, steve.liang)@ucalgary.ca

Commission IV, WG IV/2

KEY WORDS: GIS, Sensor, Query, Real-time, Spatial, Temporal, Web based, Application

ABSTRACT:

World-wide sensor web generates tremendous amount of sensor data stream allowing people to observe events that were previously unobservable. Sensor web has been wildly applied in many monitoring systems; some of them are extremely time-sensitive, e.g., disaster management systems. However, with the growing amount of sensor data, the traditional *request/response* communication model becomes inefficient as it is based on point-to-point *pulling* interactions between users and data providers. In order to address this issue, *publish/subscribe* communication model has been proposed and applied in many applications, e.g., web blogging. The publish/subscribe model utilizes an intermediary broker on matching predefined queries with the data *pushed* to the broker. However, we argue that the publish/subscribe model is hard to be directly applied to sensor web due to the fact that most sensor web services are based on pulling interaction model only. For instance, more and more sensor data providers are publishing their sensor data with the Open Geospatial Consortium (OGC) Sensor Observation Service (SOS) standards, and the OGC SOS services are based on the *request/response* model. Therefore, in order to address this issue, we propose a *hybrid pull-push* system to retrieve sensor web data in a timely manner. The preliminary experimental results indicate that the proposed system is able to fetch near real time sensor streams from pull-based sensor web services.

1. INTRODUCTION

1.1 Background

The World-Wide Sensor Web (Liang *et al.*, 2005) is generating tremendous volumes of real-time sensor data streams ranging from video camera networks monitoring real-time traffic to matchbox-sized wireless sensor networks embedded in the environment to monitor habitats. As these data streams enable scientists to observe phenomena that are previously unobservable, the World-Wide Sensor Web is increasingly attracting interests for a wide range of applications, including: habitats monitoring systems (Mainwaring *et al.* 2002), environment observation systems (Hart and Martinez 2006), structure health monitoring systems (Hsieh 2002), health applications (Xu, 2002), fire emergency response systems (Kassab *et al.*, 2010), etc. Among these applications, many of them are time-sensitive and require prompt notifications.

However, with the vast amount of sensor data in sensor web, the traditional *request/response* communication model becomes inefficient as it is based on point-to-point *pulling* interaction between users and data providers. In order to address this issue, *publish/subscribe* communication model (Birman and Joseph, 1987) provides an intermediary broker for users to register queries and for providers to *push* new data to. The broker sends notifications to users as new data meet their query criteria.

While the publish/subscribe model has been widely applied in other disciplines, e.g., web blogging, this model is relatively new in the sensor web field. We argue that a major reason of this slow adoption is that the most current sensor web services are currently based on pulling model only. Even though sensor data streams are pushed to the data repository of web services, users need to pull the sensor data from the sensor web services proactively. For example, Open Geospatial Consortium (OGC) Sensor Observation Service (SOS) (OGC 2007), as one of the most popular open sensor web standards, defines a standard protocol for users to retrieve sensor metadata and observations through Internet. In general, this issue impedes users from getting real-time notifications about events happening on sensor web.

In order to address this issue and achieve the goal of timely notification, this paper proposes a *hybrid pull-push* system for near real-time sensor data notification. This system contains three major components, namely (1) query aggregator, (2) adaptive feeder, and (3) sensor data cache. Users can first register queries (*i.e.*, subscriptions) to the system. Before sending requests to sensor web services to pull sensor data, the query aggregator aggregates queries in order to avoid redundant requests. Then based on the aggregated requests, the adaptive feeder pulls sensor data from sensor web services in a timely manner. Finally, after receiving responses from services, the input adaptor preserves sensor data in the sensor data cache according to users' query criteria.

In this paper, we use OGC SOS as the sensor data sources. SOS version 1.0 has been published in 2007 and SOS version 2.0 has been approved in March 2012. SOS is suitable for our experiment because SOS is already adopted by many sensor data providers and current SOS implementations have the aforementioned challenges.

To sum up, the major objective in this paper is to build a system allowing users to subscribe sensor data by setting spatio-

* Corresponding author.

temporal criteria and then receive near real-time notifications containing the data matches the criteria. The remainder of this section introduces works related to this paper. Then in section 2, we present the proposed system including the system architecture and details of each component. Section 3 shows and discusses the preliminary experimental results. Finally, section 4 offers conclusions and future work.

1.2 Related Works

Publish/Subscribe is a communication model decoupling publishers and subscribers. Subscribers first register their event of interest, and asynchronously get notifications of events generated by publishers. Unlike the point-to-point synchronous communication model, the asynchronous publish/subscribe model is more suitable for large-scale distributed applications. For example, publish/subscribe model has been widely applied in web blogging with RSS[1] (RDF Site Summary) and Atom[2] technologies. Eugster et al. (2003) wrote a well-cited summary paper about publish/subscribe systems.

Besides the publish/subscribe system, there are other types of applications aim on processing streaming data or events, such as complex event processing (CEP), data stream management system (DSMS) (similar to event stream processing (ESP)), and simple event processing. The original designs of these applications were different. However, as the evolving of these applications, their functionalities become similar.

Based on their original designs, we can differentiate these applications by the degree of query complexity they handle. In general, publish/subscribe systems handled the simplest queries and simple event processing added filtering functionality on the basic publish/subscribe. While publish/subscribe and simple event processing focused on individual events, DSMS and CEP processed multiple data streams.

There have been many publish/subscribe systems and stream processing systems. For example, Birman et al. (1987), Powell (1996), Skeen (1998), TIBCO (1999), Siena (Carzaniga et al. 2001), JEDI (Cugola et al. 2001), and Hermes (Pietzuch 2004) are the existing works on publish/subscribe system. NiagaraCQ (Chen et al. 2000), TelegraphCQ (Madden and Franklin 2002; Chandrasekaran et al. 2003), COUGAR (Bonnet et al. 2001), PLACE (Mokbel et al. 2005), Tapestry (Terry et al. 1992), Cayuga (Brenna et al. 2007), StreamBase (2011), Oracle CEP (2009), Esper (2012), IBM System S (Gedik et al. 2008), and Microsoft StreamInsight (2012) are the researches related to data stream processing system.

At the current stage, publish/subscribe systems and DSMS are similar in terms of their high level architectures and functionalities. They both allow users to register queries, and allow data providers to push data to the system through an input adaptor. Then they both have a continuous query engine to match the new coming data with the predefined queries. Finally, they both have an output adaptor to disseminate notifications to users.

However, as mentioned before, one of the major reasons that publish/subscribe system is difficult to be applied for sensor webs is that most of current sensor web services only support pull-based communication model. In other words, there is no suitable sensor data source for a traditional publish/subscribe system as they require data to be pushed from data sources.

Therefore, in order to address this issue, this paper proposes a solution that modifies the input adaptor module in a publish/subscribe system. Instead of only accept pushed data, the input adaptor generates requests that pull data from data sources, and then it pushes the new data to the next module in the publish/subscribe system. Hence, we this proposed solution as a *hybrid pull-push* system.

2. PROPOSED SOLUTION

This paper focuses on the input adaptor module in a publish/subscribe system to retrieve sensor data from pull-based sensor web services in a timely manner. Other modules of a publish/subscribe system, e.g., continuous query engine, are out of the scope of this paper.

2.1 System Architecture

The proposed input adaptor has three major components, namely (1) query aggregator, (2) adaptive feeder, and (3) sensor data cache. With the queries users submit, the query aggregator first aggregates queries in order to avoid redundant requests. Then the adaptive feeder tries to get new data with the aggregated queries in a timely manner. Finally, the sensor data cache is where the system keeps the sensor data according to the query criteria. The workflow and architecture is shown in Figure 1.

In this paper, we use OGC SOS as the sensor data source as it is one of the most popular open standards to share sensor data online. In addition, OGC SOS also only supports pull-based interaction model, which matches the major issue we mentioned earlier.

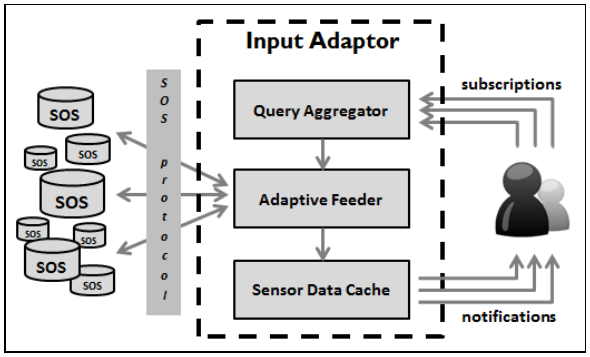

Figure 1. System architecture and workflow

In the following two subsections, we present the details of the query aggregator and the adaptive feeder.

2.2 Query Aggregator

Before presenting the functionality of the query aggregator, we need to discuss what a query is in the sensor web context. Since sensors measure a specific physical phenomenon (e.g., wind speed) at a certain geographical location and time point, each sensor reading contains at least the following five elements, namely, a physical phenomenon identifier, a measurement value, a unit of measurement, a geographical location, and a

[1] RSS 2.0 Specification (http://www.rssboard.org/rss-specification)
[2] Atom wiki (http://www.intertwingly.net/wiki/pie/FrontPage)

time point. Moreover, since sensor readings are pushed to a sensor web service for users to retrieve, some additional parameters are required to locate the sensor readings, such as service location on the Internet (*i.e.*, service URL) and the observation offering ID in the OGC context.

Therefore, when users want to register a query for sensor data in OGC SOS, they need to specify the service location, an observation offering ID, a observed property URI (which is the identifier for the physical phenomenon), a geographical coverage (*i.e.*, a bounding box), and a temporal coverage (*i.e.*, a time period). In addition, since the objective of this proposed system is to retrieve "new" data in a timely manner, the temporal coverage could move forward as time goes by, which is called the *sliding window*. Besides the sliding window, there are two other types of temporal window, namely, *fixed window* (the temporal coverage will not change) and *landmark window* (the start time point is fixed while the end time point is moving). Therefore, in our system, users need to specify the type of temporal window they want to use.

After defining what a query is in the senor web context, we now present the functionality of the query aggregator. Since most sensor web services are based on pulling interaction model, the input adaptor needs to proactively requests data from services. However, since queries from users could have different but overlapped geographical and temporal coverage, if we pull data from sensor web services based on each query, the overlapped spatio-temporal coverage will be transmitted redundantly. These redundant transmissions could cause huge and unnecessary burden on both service-side and client-side as the amount of sensor data growing rapidly. Therefore, we propose the query aggregator to aggregate and filter out unnecessary requests to pull data from sensor web service efficiently. We consider this query aggregator as one of the major contributions of this paper.

In the query aggregator, we utilize the LOading Spatio-Temporal Indexing Tree (LOST-Tree) (Huang et al. 2011) as data loading management component to aggregate user queries and avoid redundant data transmission. LOST-Tree uses two key ideas to aggregate requests and specify the loaded portions. First, LOST-Tree applies predefined hierarchical spatial and temporal frameworks, so that both the spatial and temporal extents of requests can be indexed for loading management. Since the frameworks are predefined, LOST-Tree can simply compare spatial and temporal indices between requests to filter out redundant transmission. Also, because the frameworks are hierarchical, LOST-Tree can aggregate several indices to attain a smaller tree size, which consequently results in a smaller memory footprint and query latency. In this paper, we use quadtree as the spatial framework and Gregorian calendar as the temporal framework. Second, LOST-Tree uses only the spatio-temporal extent of requests to specify the loaded portions. Since LOST-Tree only manages the spatio-temporal extent of requests, LOST-Tree does not grow with the sensor data volume, which also allows LOST-Tree to attain a small memory footprint and query latency.

2.3 Adaptive Feeder

After the query aggregator aggregates and filters out unnecessary requests, the aggregated requests are forwarded to the adaptive feeder. The major problem to retrieve sensor data from a pull-based data source is that we do not know when a new data will be available in the service. A naïve solution is to frequently and periodically send requests to the SOS servers. However, this approach could generate many unnecessary requests with empty-hit response (*i.e.*, no data contains in the response).

Therefore, in order to address this issue, the adaptive feeder attempts to predict when new data will be available in SOS servers. By detecting the sensor sampling frequency (*i.e.*, the frequency that a sensor measure a phenomenon), the adaptive feeder modifies the requesting frequency accordingly. Although the sampling time (the time that the data was measured) and valid time (the time that the data is available online) are different, a client can only speculate the valid time from the sampling time, as the valid time is not available for the client.

In our current adaptive feeder design, the best scenario is that the new sensor reading becomes available right after it is measured (*i.e.*, small difference between sampling time and valid time). The adaptive feeder will be able to retrieve the data in a timely manner as the prediction is close to reality. However, sensor readings sometimes need to be buffered or calibrated before being inserted into web service. In this case, even though the valid time could be very different from the prediction, the adaptive feeder can still retrieve data no later than the sampling frequency as soon as the data becomes available online.

3. EXPERIMENTAL RESULTS

In this section, we present the preliminary experimental results of the proposed system. We tested the proposed solution on two existing sensor web services (here we name them as service A and service B). While both services have the same sampling frequency (around 15 minutes), these two services have different data update behaviour. Service A makes the sensor data available as soon as it receives data from sensors, which could be our best scenario. Service B first buffers or calibrates sensor data before making them available online, in which the sampling time is far from the valid time.

It is worth to note that in addition to the aforementioned prediction time, we also add a buffer time (*i.e.*, 30 seconds) to accommodate the possible delay when services make data available online. In this case, our results would be 30 seconds worse than the best scenario. This buffer time will be adjusted to a shorter setting after we get more testing results.

We record the difference between the time point that we get the new data and the time point that the latest reading was measured. This time difference evaluates how "real time" the proposed system can achieve. Table 1 shows the preliminary experimental results including the average and standard deviation of time difference, the number of unnecessary requests (*i.e.*, request that does not retrieve any new data), and the total number of feedings performed in this experiment.

As we can see in the column of service A (*i.e.*, the best scenario), we can retrieve new data in the time slightly larger than 30 seconds, which is the buffer time. In addition, all 21 feedings are able to retrieve new data, which means there is no unnecessary request in the case of service A.

On the other hand, as we can see in the column of service B, since service B does not make data available online as soon as it is measured, the adaptive feeder will send requests every detected sampling frequency, which consequently causes many unnecessary requests. As we can see from Table 1, there is a 90

minutes delay before service B makes sensor data available online. In this case, the updating behaviour of service B may not be suitable to be data sources as near real-time applications.

	Service A	Service B
Average time difference (millisecond)	30,679	5,433,863
Number of unnecessary requests	0	24
Total number of feedings	21	26

Table 1. Experimental results

4. CONSLUSIONS AND FUTURE WORK

We have presented a hybrid push-pull system to retrieve sensor data in a near real-time manner. The proposed system first uses the query aggregator to aggregate user queries and filter out unnecessary requests. Then the adaptive feeder component detects the updating frequency of OGC sensor web services and retrieves sensor data with the aggregated requests in a timely manner. As shown in the experimental results, our proposed system can retrieve sensor data in a timely manner if the service makes data available online as soon as it is measured. On the other hand, if the service buffers or calibrates sensor data before making them available online, the proposed system will periodically request data with the detected sampling frequency with the trade-off of redundant requests.

As we can see from the experimental results, the performance of the proposed system is highly related to the updating behaviour of sensor web service. Therefore, one of our future works is to simulate sensor web services with different data updating behaviours. The other future direction is the integration of sensor data from different sensor web service. The current sensor web services are heterogeneous in terms of protocol, syntactic, and semantic. Users need to first find the services that host the data they are interested in. However, with the growing number of sensor web services, this discovery process becomes a challenging task. Therefore, how to integrate sensor data to provide a coherent view of sensor web is also one of our future works.

REFERENCES

Birman, K. and Joseph, T., 1987. Exploiting virtual synchrony in distributed systems, In: *the 11th ACM Symposium on Operating Systems Principles*, Vol. 21, Part 5, pp. 123-138.

Bonnet, P., Gehrke, J., and Seshadri, P., 2001. Towards Sensor Database Systems, In: *International Conference on Mobile Data Management*, pp. 3-14.

Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., and White, W., 2007. Cayuga: A High-Performance Event Processing Engine, In: *the 2007 ACM SIGMOD*, New York, USA, pp. 1100-1102.

Carzaniga, A., Rosenblum, D., Wolf, A., 2001. Design and Evaluation of a Wide-Area Event Notification Service, *ACM Transactions on Computer Systems*, Vol. 19, Part 3, pp. 332-383.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S.R., Reiss, F., and Shah, M.A., 2003. TelegraphCQ: Continuous Dataflow Processing, In: *ACM SIGMOD*, New York, USA.

Chen, J., DeWitt, D.J., Tian, F., and Wang, Y., 2000. NiagraCQ: A Scalable Continuous Query System for Internet Databases, In: *the 2000 ACM SIGMOD*, pp. 379-390.

Cugola, G., Nitto, E.D., and Fugetta, A., 2001. The JEDI Event-based Infrastructure and Its Application to the Development of the OPSS WFMS, *IEEE Transaction on Software Engineering*, Vol. 27, Part 9, pp. 827-850.

Esper, 2012. http://esper.codehaus.org/ (05 January 2012).

Gedik, B., Andrade, H., Wu, K.L., Yu, P.S., and Doo, M., 2008. SPADE: The System S Declarative Stream Processing Engine, In: *the 2008 ACM SIGMOD*, New York, USA, pp. 1123-1134.

Hart, J.K. and Martinez, K., 2006. Environmental Sensor Networks: A revolution in the earth system science? *Earth Science Reviews*, Vol. 78, pp. 177-191.

Hsieh, T.T., 2004. Using Sensor Networks for Highway and Traffic Applications. *IEEE Potentials*, Vol. 23, Part 2, pp. 13-16.

Kassab, A., Liang, S., and Gao, Y., 2010. Real-Time Notification and Improved Situational Awareness in Fire Emergencies using Geospatial-based Publish/Subscribe, *International Journal of Applied Earth Observation and Geoinformation*, Vol. 12, Part 6, pp. 431-438.

Liang, S.H.L., Croitoru, A., and Tao, C.V., 2005. A Distributed Geospatial Infrastructure for Sensor Web. *Computers and Geosciences*, Vol. 31, Part 2, pp. 221-231.

Madden, S. and Franklin, M.J., 2002. Fjording the Stream: An Architecture for Queries over Streaming Sensor Data, In: *the 2002 International Conference on Data Engineering*, pp. 555.

Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and Anderson, J., 2002. Wireless Sensor Networks for Habitat Monitoring. In: *the 2002 ACM International Workshop on Wireless Sensor Networks and Applications*. Atlanta, USA.

Microsoft, 2012. "Microsoft StreamInsight 2.0" http://msdn.microsoft.com/en-us/library/hh750619(v=SQL.10).aspx (12 January 2012).

Mokbel, M.F., Xiong, X., and Aref, W.G., 2005. Continuous Query Processing of Spatio-Temporal Data Streams in PLACE, *GeoInformatica*, Vol. 9, Part 4, pp. 343-365.

Open Geospatial Consortium, 2007. "Sensor Observation Service" http://www.opengeospatial.org/standards/sos (05 January 2012).

Oracle, 2009. "Oracle Complex Event Processing: Lightweight Modular Application Event Stream Processing in the Real World
http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/oracle-37.pdf (05 January 2012).

Pietzuch, P.R., 2004. Hermes: A Scalable Event-based Middleware. *Queens' College, University of Cambridge*, Cambridge, UK.

Powell, D., 1996. Group Communication. *Communications of the ACM*, Vol. 39, Part. 4, pp. 50-97.

Skeen, D., 1998. "Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview" http://www.vitria.com (05 January 2012).

StreamBase, 2011. "StreamSQL Guide" http://www.streambase.com/developers/docs/latest/streamsql/index.html (05 January 2012).

Terry, D., Goldberg, D., Nichols, D., and Oki, B., 1992. Continuous Queries over Append-Only Databases, In: *the 1992 ACM SIGMOD*, pp. 321-330.

TIBCO, 1999. "TIB/Rendezvous" TIBCO, Palo Alto, CA. http://www.tibco.com/multimedia/ds-rendezvous_tcm8-826.pdf (05 January 2012).

Xu, N., 2002. "A Survey of Sensor Network Applications", IEEE Communications Magazine, http://enl.usc.edu/~ningxu/papers/survey.pdf.

ACKNOWLEDGEMENTS

The authors would like to thank CANARIE, Cybera, Alberta Innovates Technology Futures, and Microsoft Research for their supports on this project.