GeoStreaming in Cloud

Seyed Jalal Kazemitabar
kazemita@usc.edu

Farnoush Banaei-Kashani
banaeika@usc.edu

Dennis McLeod
mcleod@usc.edu

Computer Science Department
University of Southern California
Los Angeles, CA 90089

ABSTRACT

In recent years, geospatial databases have been commercialized and widely exposed to mass users. Current exponential growth in data generation and querying rates for these data highlights the importance of efficient techniques for streaming. Traditional database technology, which operates on persistent and less dynamic data objects does not meet the requirements for efficient geospatial data streaming. Geostreaming, the intersection of data stream processing and geospatial querying, is an ongoing research focus in this area. In this paper, we describe why cloud is the most appropriate infrastructure in which to support geospatial stream data processing. First, we argue that cloud best fits the requirements of a large-scale geostreaming application. Second, we propose *ElaStream*, a general cloud-based streaming infrastructure that enables huge parallelism by means of the divide, conquer, and combine paradigm. Third, we examine key related work in the data streaming and (geo)spatial database fields, and describe the challenges ahead to build scalable cloud-based geostreaming applications.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications—*Spatial databases and GIS*; H.2.4 [**DATABASE MANAGEMENT**]: Systems—*Parallel databases*

General Terms

DESIGN

Keywords

Geostreaming, Cloud computing, Data stream processing, Spatial databases

1. INTRODUCTION

Geostreaming, or processing and analyzing stream data with geographic and spatial attributes, has recently gained attention in academia and industry [19, 8]. The real-time geo-tagged data received in such applications need to be processed in short time to meet the low latency requirements of continuous queries. Among all geostreaming applications, large-scale ones are growing both in quantity and scale due to recent advancements in sensing technology and popularity of social media and smartphones. For example, consider the following two applications:

Example 1: **Locatoin-Based Services (LBS):** Online advertisement and dating applications are among many LBSs that receive real-time location updates and continuously run geospatial queries. As an example scenario, a user might be interested to receive special deals from stores within a small range. The value of such information deteriorates as he passes by the stores. In a similar vein, a Foursquare application, which is hosted by millions of users just in the U.S., can run a kNN query to find nearest friends who have just checked-in to restaurants around a client. The market share of these applications is growing fast for various reasons including current appeal for social media such as Facebook with its 350 million active mobile users.

Example 2: **Intelligent Transportation Systems (ITS):** Valuable real-time geo-tagged data are produced by loop detectors and GPS devices in cars, taxies, trucks, and by government agencies in big cities around the world [19, 11]. For example, millions of drivers in LA county would like to know the fastest route or time to destination while driving and expect this information to be promptly updated right after an accident happens in the road ahead. Once again, the scale is growing every day as manufacturers inject new cars to road networks, more sensors are installed throughout roadways, and most important, more cars are equipped with advanced data transmission systems.

Large-scale geostreaming applications often share the following features:

1. *Parallelizable*: Geostreaming applications include independent modules that can run in parallel. For instance, every Foursquare client looking for his friends would submit a separate query that can be processed in parallel with others.

2. *Continuously growing and resource intensive*: Due to recent changes, namely cost reduction in sensor technology, availability of GPS-enabled devices, and popularity of social networking, geostreaming applications encounter continuous rise in load. There are numerous success stories on companies offering LBS that faced

IWGS '11 November 1, 2011, Chicago, IL, USA.

thousand percent load growth within months. Moreover, large quantity of records received in real-time from one side, and meeting the low-latency requirements on the other side demand huge computational resources and bandwidth. A shopping or driving individual interacting with geostreaming applications expects seamless response to his queries over real-time data.

3. *Changing load*: Geostreaming applications frequently confront change in their load. In fact, they encounter fluctuating load interrupted by (predictable) peaks. An LBS has to serve the largest number of queries during mid-day hours. Similarly, an ITS receives many queries during the rush hour. This is while both applications are almost idle during early hours after midnight.

We claim that there is no existing framework for geospatial and stream data processing that can keep up with all the noted features. A single machine has too limited resources. A parallel data stream management system (DSMS) parallelizes the tasks. However, it has barriers towards satisfying the other two features. The market is limited to big companies as it requires high initial down payment for specialized software solutions and high-end servers. Specialists need to continually keep an eye on it to improve current configuration and hardware set up to accommodate increased load. This would also include undesirable system downtime which affects both revenue and clients. Moreover, the costly hardware remains idle most of the times as peak load rarely happens.

The cloud infrastructure on the other hand, has properties that can properly accommodate the features of a large-scale geostreaming application. First, multiple nodes available at a time would enable parallel geospatial query processing. Second, it can provide virtually unlimited resources upon request within a short time. Thus, it potentially enables *automatic scale out*. Third, it is *elastic*: if correctly programmed, it reacts flexibly to fluctuations in load by managing the available resources. Based on the *pay-as-you-go* cost model of cloud, one pays only for the amount of resources in use and not those required during maximum peak times. This would enable all start-ups to be as much in the game as are the top players in IT industry since no upfront resource provisioning or maintenance costs are incorporated.

We believe that the large-scale geostreaming applications can be best served on cloud compared to any other infrastructure. Extensive research is required to adapt such applications to obtain all the automation, flexibility, parallelism, and cost efficiency that cloud offers.

In what follows, we first review existing work on geospatial and streaming systems in section 2. Next in section 3 we explain our Elastic Stream processing framework (*ElaStream* for short) and show how it takes full advantage of what cloud offers in order to handle large-scale geostreaming applications. We use the simple (and thus scalable) divide, conquer, and combine paradigm to gain two main benefits. First, it helps avoid bottlenecks and thus guarantees true scalability as load grows. Second, it increases parallelization thus conforming with another feature of the target applications. We also exemplify the workflow in our framework with basic streaming operators. Finally, in section 4 we highlight the challenges ahead of streaming and geospatial communities for building scalable geostreaming applications in cloud.

2. RELATED WORK

In this section, we review the related work on and off the cloud infrastructure.

2.1 On-premises Systems

Similar to cloud, parallel Database Management Systems (DBMS) and DSMS can address the online processing and storage constraints of continuous query processing over data. Teradata is a parallel DBMS running on proprietary hardware and has recently started using solid state devices as storage [16]. Vertica [6] is a parallel analytic database running on commodity servers. Its column-oriented design makes it a great choice for data-warehouses. System S [15] is a large-scale stream processing middleware which supports both structured and unstructured data stream processing.

These systems have common drawbacks though. The hardware and software components that enable dynamic addition and configuration of plug and play processors are not implemented in such systems. This is mainly because they were not designed to be used by many users with varying workloads. Today's evolving real-time data generation rates require human intervention to upgrade the fixed hardware and software structure of these systems. This would impose recurring complexity and cost specially during the upgrade period and affects customer satisfaction.

2.2 Cloud-based Systems

2.2.1 Data Management

NoSQL (Not only SQL) databases have recently emerged to handle a wide range of large-scale web applications. They most often differ from a SQL database in that they are not relational, sacrifice ACID properties in favor of performance, horizontally scale in a cluster, and replicate the data for availability and fault tolerance. NoSQL databases can be categorized based on their data model, read/write performance for random/batch access, etc. Neo4j [5] is an open source object-oriented graph database which supports ACID properties and is highly scalable. MongoDB [4], mainly used for web data, is an open-source schema-free document-store database without support for transactions. Dynamo [14] is Amazon's reliable proprietary key/value database which acts similar to a distributed hash table. To be highly available , it does not provide any isolation and supports a weaker level of consistency called eventual consistency. HBase [2] is a column-store key/value database with emphasis on high write throughput. Cassandra [21], a similar column-store database, has the advantage that it can be tuned for synchronous or asynchronous I/O and replication. YCSB [13] is a benchmark for comparing NoSQL databases. Key/value and column-store databases with high random access read and write performance (e.g. HBase and Cassandra) are most relevant to streaming applications. However, their current indexing and querying facilities are limited and do no keep up with the requirements of large-scale geostreaming applications targeted in this paper.

Relational databases have been offered as utility services over cloud by large companies. For example, Microsfot SQL Azure provides a subset of SQL Server features and its size is currently limited to 50 GB per database [3]. Amazon Re-

lational Database Service (RDS) provides full features of a MySQL or Oracle database upon request and thus follows their technical specifications (e.g. consistency levels) [1]. Little technical details are available about these competing services. The performance of these systems have been compared under different workloads in [20]. To the best of our knowledge, scalable spatial index structures are not implemented in cloud databases. Also, they target a wide range of OLTP applications (e.g. banking) and are thus costly.

Continuous Bulk Processing (CBP) falls between DBMS and data stream processing and focuses on continuous processing of disk-resident data in a batch mode. [23] manipulates MapReduce to support incremental and stateful batch processing. This can be widely used to prevent reprocessing the whole data set after new chunks of daily data are added. Nova [27] is a workflow manager built at Yahoo! to push continually arriving new data to Hadoop, an open-source implementation of MapReduce. Comet [17] is a similar solution proposed by Microsoft over Dryad, their MapReduce-like processing architecture. These technologies fail to satisfy the requirements of large-scale geostreaming applications due to the large latency caused by disk-oriented structures and also batch processing of records instead of adopting an event driven approach in geostreaming.

2.2.2 Data Streaming

Streaming solutions for cloud have just started to emerge. S4 is Yahoo!'s newborn streaming system for cloud [25]. Every processing node in S4's symmetric design contains several processing elements (PEs) each configured to handle key/value based events. Communication among nodes and automatic failover is managed by ZooKeeper. The current implementation has several drawbacks. The number of processing nodes and the distribution of PEs on each one of them is fixed and has to be hard-coded by application programmer. This would avoid dynamic load balancing and also prevents utilizing the scalable architecture of cloud.

Esc [29] distributes a DAG of user-defined functions among nodes and has a module for scaling out upon increased input rates. The system is implemented as a distributed Erlang application with each PE being a light-weight Erlang process. Thus, failover is handled by Erlang. Esc is tested against a CPU intensive function (i.e. finding correlation among sequences of stock ticks) with a peak input rate of 2.5 requests/second. Limited available experimental results make it hard to have a comprehensive understanding of its performance on general streaming applications (e.g. higher event rates, larger number of nodes, communication performance for larger workflows). However, existing results show that the system reacts slowly to load increase. None of the above systems address general streaming requirements [9].

ElasticStream [18] proposes a cost efficient way to transfer incoming stream to cloud when local nodes cannot accommodate it. The prototype system periodically predicts future data rates. Then, the minimum number of required cloud virtual machines is calculated according to latency limits in SLA (Service Level Agreement). Finally the remaining processes are assigned to cloud. The method costs less than reserving a fixed number of cloud nodes. However, it does not provide true elasticity as nodes are added based on periodic predictions rather than real-time load.

There are also some related work that are manipulating the architecture of MapReduce. [22] targets incremental one-pass analytics by replacing the sort-merge structure of MapReduce with a dynamic incremental hash technique. [12] reduces the high I/O cost between execution of map and reduce phases. These deformations of MapReduce, if in the right path to a streaming system, do not yet meet the needs of a general streaming system either because they have not completely taken off from persistent storage of data or they still do not meet the ultra-low latency required in streaming applications.

2.2.3 Geospatial Query Processing

Little study has been done on processing geospatial data over cloud. [31] parallelizes geospatial data processing over a single machine by using hundreds of GPU cores that are currently available on personal computers. This solution is not scalable though as the processing power is bound by the number of GPUs that a manufacturer embeds in one machine.

[7] provides an end to end geospatial query processing solution with MapReduce. First, a voronoi-based index structure is built in offline mode. Spatial queries such as RNN and kNN are then formalized as Map and Reduce functions. Finally, the system handles incoming queries as MapRedue tasks. This method has some drawbacks. First, the index structure only supports stationary data objects. The method is not applicable to the more common case of moving objects since the index structure needs to be totally recomputed per any object movement. Second, due to the inherent disk-dependence of MapReduce, the method's application is limited to batch processing of snapshot queries. As a result, continuous queries are not supported.

[26] applies two spatial index structures to a NoSQL database by mapping indexed partitions of local data to buckets in the database. K-d tree and quad-tree are used to index incoming geospatial data. A linearization technique (i.e. z-ordering) is used to define the order of partitions in database. Even in presence of a reasonable number of compute and storage nodes, queries over stored data take seconds [26]. Meanwhile, the process-after-store structure of this method binds its performance to the throughput for prompt insertion which is still no more than tens of thousands records per second for current data stores [26, 13].

[30] proposes a scalable architecture for storing and querying multi-dimensional data. Every node stores an R-tree of its local data. Multiple nodes are connected using CAN routing protocol [28]. Upon receiving a query, the compute nodes including the answer are located within a few hops. The method has a fair performance on kNN queries as it initially considers a small range and enlarges the search range when top-k closest data are not yet found. R-tree index also does not perform well in a dynamic environment which is the target of this paper.

3. PROPOSED APPROACH

ElaStream is a framework on cloud infrastructure which accommodates the three common features of large-scale geostreaming applications. In section 3.1 we first start by providing a big picture of the framework. Then we explain how this framework conforms to Babu and Widom's streaming architecture [10, 9]. The framework is then explained in detail. In section 3.2, we illustrate how streaming operators can be efficiently implemented in our framework.

3.1 ElaStream Framework

The asymmetric organization of nodes in *ElaStream* includes a *supervisor node* and virtually unlimited number of *processing nodes*. Each processing node includes *worker* threads that consume available resources on that node in a time-shared manner. An operator in the query plan is considered as a task. A task is divided into more granular *subtasks* to utilize the parallelization that cloud offers. A worker receives an input stream, runs an assigned subtask, and publishes results. To increase efficiency, the same subtask might be assigned to other *similar workers* running the same subtask in parallel on different processing nodes. A result published by a worker can be consumed by other workers or considered as the final query result. A cloud database is used for logging.

Babu and Widom proposed a streaming architecture at the query abstraction level and verified its compatibility against general streaming scenarios [10, 9]. Their abstract architecture includes *stream*, *store*, and *scratch* components for each query. Tuples in stream are considered as query results. Store includes potential answers. Based on following input tuples, tuples in store might later be forwarded to stream (as part of the answer), moved to scratch (as derived data for future reference), or discarded. Saved data in scratch might later be moved to store to help compute query results.

Similar to their work, we consider these components for each query as well as all its operators on cloud infrastructure. However, due to our emphasis on enabling maximum parallelism through distributing queries at a high granularity, we further utilize this abstraction for every worker that together with similar or different workers represents an operator. The modular behaviour of an operator in our cloud-based framework matches that of Babu and Widom's as the components of end-point worker(s) of an operator can be mapped to those in their architecture. Materializing these abstract components in cloud is straightforward. Each worker on a processing node pushes events in stream to subscribing workers through network. Store is locally saved in memory. A NoSQL database is used to store scratch. It is important to note that storing scratch on local disk is not a proper decision as intermediate results of long running queries would be lost in case of a node failure (which is not an exception in cloud). Instead, a NoSQL database can provide the substituting processing node with valuable stored data.

ElaStream addresses the three features of large-scale geostreaming applications. Figure 1 shows two cases that can be parallelized by workers in *ElaStream*. A dotted line delimits the centralized design on the left and their cloud-based counterpart on the right side. Data streams are shown as arrows. A rectangle tagged with number i represents a task i which can run on a worker. Independence is the key to parallel execution of tasks. As shown in Figure 1a, independent workers over a stream can run in parallel over cloud compared to sequential execution over a centralized architecture. Figure 1b shows multiple streams being processed independently by similar workers running in parallel on separate processing nodes.

Cloud provides virtually unlimited resources upon request. Continuously growing geostreaming applications requiring low-latency response time need to avoid bottlenecks in order to utilize the abundant resources in scaling out. Within our framework, we follow the divide-and-conquer paradigm to increase performance on many intricate operators (Figure 2a). An input stream, is first divided into sub-streams by *dispatcher* workers. Built-in and user-defined operators are also divided into subtasks as hard-coded optimized functions and implemented interfaces in user program respectively. Parallelization schemes in Figure 1 can help avoid bottleneck in this phase. Next, sub-streams are forwarded to *conqueror* workers that conquer the assigned subtasks in parallel. Finally, partial results are combined by a *combiner* worker to produce the output stream. Based on the amount of load, the supervisor node might decide to extend the three level structure of dispatcher, combiner, and conqueror workers to a hierarchy with more layers (i.e. more than one layer for each type of worker). As we will note in section 4, adjusting the granularity of subtasks and hierarchy of workers is crucial to cost and performance.

The divide, conquer, and combine paradigm in our framework resembles the two layer structure of Map and Reduce. However, our framework is dedicated to streaming and is different from MapReduce, first because it does not depend on persistent storage, second, it is based on a push (rather than pull) model, and third, it envisions the components of a streaming application (e.g. dealing with streaming operators) which are all missing in the MapReduce framework. Unlike the recent trend in some papers, we prefer not to affiliate our framework to MapReduce solely because they commonly utilize the parallel hardware by distributing subtasks.

There might be cases where an operator is indivisible or it is a user-defined operator for which the programmer has not bothered to break it into subtasks. Chances are high for such an operator to become a bottleneck especially if it is fed by a large input stream, or if it is a compute intensive function. Fortunately, all cloud providers offer a diverse range of instance types with high computation and memory capabilities and thus, this potential bottleneck can be avoided (Figure 2b). It is good to note that little before cloud evolved, one had to use parallel database or streaming systems and pay a lot to buy a powerful server to accommodate rare peaks. Now with cloud's pay-as-you-go model, small start-ups can rent even more powerful servers and spring towards success.

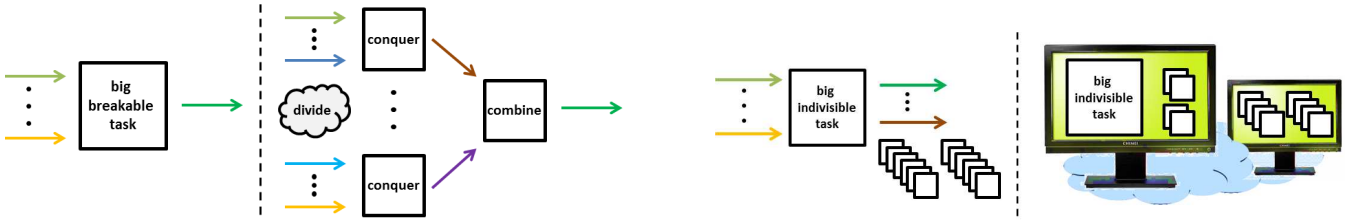
Our framework reacts flexibly to load fluctuations in geostreaming applications. Towards enabling this feature, the supervisor selects a less loaded processing node and extends the existing layout of workers by adding new workers to the selected node. The supervisor node is responsible for global monitoring and scheduling tasks such as health monitoring, analyzing statistics, and handling request messages from processing nodes. When the supervisor is asked for help on a potential lack of resources in a processing node, it reviews the fetched statistics from all nodes to make a choice from the following options: ordering a new processing node (thus scaling out), elastically loading existing nodes, or ignoring the request based on its level of severity. Adversely, shrinking policies may be applied upon severe load reduction notifications. The supervisor continually broadcasts heartbeat messages to all processing nodes to express its presence.

Each processing node includes a pool of workers, their configuration information, statistics unit, and a scheduler for managing resources. The configuration information of a worker includes initial and currently assigned share of machine resources and a hash function for stream routing pur-

(a) Independent tasks over an event

(b) Processing events independently

Figure 1: Parallelization can make event processing faster and accommodate more events.



(a) Divide, conquer, and combine.

(b) Allocating more resources to large indivisible tasks

Figure 2: Avoiding bottlenecks to achieve scalability.

poses. Statistics unit continually sends statistics messages to the supervisor. Such messages also serve as the heartbeat message communicating the availability of processing nodes. The scheduler works closely with the statistics unit and dynamically allocates resources among workers based on their current load. It may send a request message to the supervisor if future load might potentially exceed available resources (e.g. if current CPU usage has consistently surpassed a threshold value or if limited amount of free memory is available).

Both the supervisor and processing nodes periodically checkpoint their status to the NoSQL database. Upon a node failure, a replacing node can load the latest configuration and statistics information. Workers on a substituting processing node that are assigned to a long running query can also load their scratch to recover query state from failure. The framework can be configured to select from a number of choices upon a supervisor node failure. Example choices in increasing order of performance and cost include requesting a new node from the cloud provider, nominating a current processing node, and using a running back-up node. The automatic failover duration ranges from minutes to less than a second depending on the choice.

3.2 Case-study of Operators

Many traditional and/or modern user-defined streaming operators inherit common design characteristics. To show the extensibility of our framework, we studied the feasibility of projection, select, group by followed by aggregate, and sort (e.g. top-k) operators within our framework. As we will note in section 4, it is important to consider the trade-off between increasing parallelization and network bandwidth usage. More specifically, moving huge chunks of data in favor of maximally parallelizing a tiny task can adversely affect the application. For the remainder of this section however, we focus on illustrating the parallelization and scalability of our framework.

Pipelined operators (e.g. projection and select) maximally benefit the parallel processing paradigm of a shared-nothing architecture -as the most scalable parallel architecture [24] - in cloud as the results provided by separate workers on different processing nodes can be independently streamed to the next worker or reported to the output. Conforming to Widom's architecture, results are directly forwarded to stream or store. Distributing the job among k similar workers on different nodes however, would make it roughly k times faster (aka linear speedup).

Blocking operators (e.g. aggregate or top-k) on the other hand, would have to undergo a divide-and-conquer process due to the dependency of the final result on stores and streams collected by each parallel worker. The general idea is that the conqueror workers separately preprocess the divided stream in parallel and the combiner worker calculates the final results (Figure 2a). top-k operator, as an instance of blocking operators, can be implemented by first running an online sort algorithm (e.g. insertion sort) in parallel (divide and conquer phases) and then finalizing by a merge operation on a single worker (combine phase). Workers on different nodes insert new events in their store, and flush it completely to stream once a punctuation event is received (exact query result) or upon approaching the memory limits for store (in case the system is configured for approximate query results). Next, streams of various workers of an operator are pushed to input buffers of the merging worker.

Group by, as another blocking operator, can be built in three layers. Dispatchers in the first layer linearly speed-up the task as the classification of each data element is totally independent from others. Events are directly forwarded to the proper stream based on a routing function (as the running subtask). Store component would remain empty as these workers only dispatch the incoming events. Conqueror workers run aggregate functions in the second layer. Partial results generated in this layer are combined by workers in the third layer. Parallelization and breaking down the tasks within our cloud-based framework avoid potential bottlenecks caused by large input size or complexity of grouping or aggregate functions.

The design is extensible to user-defined operators and ag-

gregates. A user can select from the hierarchical or indivisible structures (figures 2a and 2b) and configure and implement the required workers. For example, a user can implement conqueror and combiner interfaces. Similar to the case of built-in operators, dispatcher, conqueror, and combiner workers may be elastically added/ dropped at run-time to handle load fluctuations.

Parallelizing the execution of operators in our framework has a number of benefits compared to centralized data stream processing. First, the need for a sampling technique would be observed k times less than the centralized solution, as the online data structure (located in store) would encounter k times smaller input rates. Pipelined and blocking operators share this benefit together. Second, thanks to partial data processing in the conquer phase, batch processing techniques would remain an option only for much higher input rates as final computation per data element would be kept small on the combiner worker.

4. CHALLENGES

Scalable stream processing on cloud: Although some prototype or specialized streaming systems have recently been proposed for cloud, in most cases their performance has not been tested under medium or large input rates, nor have they addressed all challenges towards an elastic streaming system:

1. Timely reaction to load increase is crucial to bounded latency and demands:
   - Prompt detection of an affecting input source
   - Little start-up time for running workers of congested operators on a new or less loaded machine
2. The correctness of results need to be ensured during and after the *transition phase*, i.e. the period in which workers are added/deleted to/from an operator for elasticity or scale out purposes. For example, consistency of route functions on different nodes , setting correct initial state for newly added workers for stateful operators, combining current and new results, etc.)
3. While decomposing tasks into highly granular subtasks and growing the divide, conquer, and combine hierarchy is beneficial, it can also adversely affect performance. Excessively transferring data among processing nodes that only "touch" the data would slow down the system. Moreover, it would increase costs as both network bandwidth and (delayed) computation affect charges in cloud[1]. Reducing costs while maximizing parallelization in presence of dynamically changing input streams is a challenge that calls for extensive research.
4. Failure is no more an exception as cloud is built on commodity machines and this is all regardless of what application is hosted on it. Relying on current cloud maintenance services (e.g. ZooKeeper as in [25]) can be helpful as long as they are not specialized to meet fundamentally different requirements. Research needs to be done to find and modify a matching service.

[1]Many cloud providers no more charge for internal data transfer among instances within the same region. However, I/O performance is constrained by type of reserved instance. Wasting the bandwidth would urge renting more expensive machines to avoid unpredicted latencies imposed by other instances residing on the same physical host.

Scalable spatial algorithms for cloud data stores: The rich set of spatial database algorithms might require trimming or reconsideration in order to fit the architecture of cloud. More specifically, we see three challenges towards persistent geospatial data storage and querying on cloud:

1. Each current data store supports a limited number of (if not only one) index structure(s). This would urge spatial indexes and queries to overlay the existing indexes. The incurred processing and I/O overhead slows down the insert and querying throughput. Finding alternative spatial algorithms that fit the underlying storage architecture and 1) ease data insertion and 2) querying at the same time might be challenging. Distinct range of available data stores can be deemed as another opportunity or challenge.
2. Current data stores, although focusing on various data access patterns and consistency levels, are not basically optimized for multi-dimensional data access. Emerging need for scalable geospatial applications calls for customized cloud data stores. Fortunately, many existing data stores are open-source; so one does not have to re-invent the whole wheel.
3. The emerging track of relational databases on cloud are still so costly and come with limitations e.g. in size [20]. However, this should not halt the spatial community from taking their part in formatting these systems to accommodate multi-dimensional applications.

Geostreaming, the rendesvous of geospatial and streaming techniques: Geostreaming applications have characteristics that can be helpful in addressing the above challenges. For example, the amount of input stream for a LBS is roughly predictable per hour. Mining current applications and gaining such knowledge can replace a sophisticated adaptive control mechanism and help scaling out or shrinking resources with a good level of precision and thus reduce costs.

5. CONCLUSIONS

Many recent geospatial applications are emerging exponentially and deal with spatial queries over real-time data. In this paper, we have proposed *ElaStream*, a general streaming framework for cloud which can accommodate continuous spatial queries. The independent structure of most geospatial data and queries helps gaining the most out of the divide, conquer, and combine paradigm of our framework. Finally, we characterised the challenges ahead of the two related communities, namely data streaming and spatial databases, in order to frame current young cloud solutions to accommodate the needs of geostreaming applications.

6. REFERENCES

[1] Amazon Relational Database Service. http://aws.amazon.com/rds/.

[2] Apache HBase. http://hbase.apache.org/.

[3] Microsoft SQL Azure. http://www.microsoft.com/windowsazure/sqlazure/.

[4] MongoDB. http://www.mongodb.org/.

[5] Neo4j. http://neo4j.org/.

[6] Vertica. http://www.vertica.com/.

[7] A. Akdogan, U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. Voronoi-based geospatial query processing with mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 9 –16, 30 2010-dec. 3 2010.

[8] M. Ali, B. Chandramouli, J. Fay, C. Wong, S. Drucker, and B. S. Raman. Online visualization of geospatial stream data using the worldwide telescope. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, September 2011.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[10] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–120, 2001.

[11] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. Ibm infosphere streams for scalable, real-ti intelligent transportation services. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 1093–1104, New York, NY, USA, 2010. ACM.

[12] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[15] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. SPADE: the system S declarative stream processing engine. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1123–1134, New York, NY, USA, 2008. ACM.

[16] T. Groenfeldt. Are SSDs Ready for the Enterprise? http://www.ciozone.com/index.php/Server-Technology-Zone/Are-SSDs-Ready-for-the-Enterpriseu.html.

[17] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou. Comet: batched stream processing for data intensive distributed computing. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 63–74, New York, NY, USA, 2010. ACM.

[18] A. Ishii and T. Suzumura. Elastic stream computing with clouds. *IEEE International Conference on Cloud Computing*, pages 195–202, 2011.

[19] S. J. Kazemitabar, U. Demiryurek, M. Ali, A. Akdogan, and C. Shahabi. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *Proceedings of the VLDB Endowment*, 3:1537–1540, September 2010.

[20] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD Conference*, pages 579–590, 2010.

[21] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010.

[22] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. J. Shenoy. A platform for scalable one-pass analytics using mapreduce. In *SIGMOD Conference*, pages 985–996, 2011.

[23] D. Logothetis, C. Olston, B. Reed, K. C. Webb, and K. Yocum. Stateful bulk processing for incremental analytics. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 51–62, New York, NY, USA, 2010. ACM.

[24] S. Madden, D. DeWitt, and M. Stonebraker. Database parallelism choices greatly impact scalability. http://www.vertica.com/2007/10/30/database-parallelism-choices-greatly-impact-scalability.

[25] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *ICDM Workshops*, pages 170–177, 2010.

[26] S. Nishimura, S. Das, D. Agrawal, and A. E. Abbadi. MD-HBase: A scalable multi-dimensional data infrastructure for location aware services. In *12th International Conference on Mobile Data Management (MDM)*, pages 7–16, 2011.

[27] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. N. Rao, V. Sankarasubramanian, S. Seth, C. Tian, T. ZiCornell, and X. Wang. Nova: continuous Pig/Hadoop workflows. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 1081–1090, New York, NY, USA, 2011. ACM.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM.

[29] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar. Esc: Towards an elastic stream computing platform for the cloud. *Cloud Computing, IEEE International Conference on*, pages 348–355, 2011.

[30] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi. Indexing multi-dimensional data in a cloud system. In *SIGMOD Conference*, pages 591–602, 2010.

[31] J. Zhang. Towards personal high-performance geospatial computing (HPC-G): perspectives and a case study. In *Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems*, HPDGIS '10, pages 3–10, New York, NY, USA, 2010. ACM.