STORM: Spatio-Temporal Online Reasoning and Management of Large Spatio-Temporal Data

Robert Christensen[1], Lu Wang[2], Feifei Li[1], Ke Yi[2], Jun Tang[1], Natalee Villa[1]
[1]University of Utah [2]Hong Kong University of Science and Technology
{robertc, lifeifei, jtang, villa}@cs.utah.edu {luwang, yike}@cse.ust.hk

ABSTRACT

We present the STORM system to enable spatio-temporal online reasoning and management of large spatio-temporal data. STORM supports *interactive* spatio-temporal analytics through novel spatial online sampling techniques. Online spatio-temporal aggregation and analytics are then derived based on the online samples, where approximate answers with approximation quality guarantees can be provided immediately from the start of query execution. The quality of these online approximations improve over time. This demonstration proposal describes key ideas in the design of the STORM system, and presents the demonstration plan.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management – *Systems*

Keywords

Spatial online sampling, spatial online analytics, STORM

1. INTRODUCTION

The increasing presence of smart phones and various sensing devices has led to humongous amounts of spatio-temporal data, and the imperative needs for rich data anlytics over such data. Many data from a measurement network and social media data sources are inherently spatial and temporal. As a result, numerous analytical tasks based on such data have a spatial and/or temporal extent.

Even though various forms of spatial and spatio-temporal analytics have been extensively studied, the ever-increasing size of spatio-temporal data introduces new challenges. In particular, when the underlying data set is large, reporting all points that satisfy a query condition can be expensive, since there could be simply too many points that satisfy a query. The CPU cost of performing an analytical task or computing an aggregation using all these points adds additional overhead, and may not scale well with increasing number of points. Hence, waiting for the exact analytical or aggregation results may take a long time.

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2735373.

An important observation is that approximate results are often good enough, especially when approximation guarantees are provided. It is even more attractive if the quality of an approximation improves continuously over time until the exact result is obtained in the end. We dub such an approach *online aggregation and analytics*. Online aggregation and analytics enables *interactive analytics and exploration* over large scale spatio-temporal data. A user may terminate a query whenever s/he is satisfied with the approximation quality provided by the system. The system can also be asked to terminate a query whenever the approximation quality for a query has met a query-specific (user specified) quality requirement. Alternatively, the system can also operate in the "best-effort" mode where user specifies the amount of time s/he is willing to spend on a given task, and the system will provide a result with the best possible approximation quality within the amount of time given.

Consider the following example. A user wants to understand the electricity usage in NYC over the first quarter. But s/he wants to explore different area and time range combinations. So s/he could zoom in to a particular area from NYC on a map and specify between January 5 to March 5, and ask for the average electricity usage per unit for units in this area and measurements in this time period. In interactive exploration, or formally interactive analytics, user can change his/her query condition without the need of waiting for the current query to complete. In other words, in the above example, user may change to a different area in NYC and/or adjust the time range to between January 15 and March 12, while the first query is still being executed.

On big spatial and spatio-temporal data sets, waiting for exact results may take a while. The user faces a *dilemma*: *either* waits for the current query to complete *or* terminates the current query and issue the new query. And the number of possible combinations a user wants to investigate in order to find interesting patterns, even for a small region like NYC and first quarter, can be daunting.

The STORM system solves this dilemma. STORM uses spatio-temporal online reasoning and management to achieve online aggregation and analytics on large spatio-temporal data. In the above example, assume that after 1 second into the execution of the first query, system reports that the average electricity usage is 973 kWh with a standard deviation of 25 kWh and 95% confidence, if the user is happy with the quality of this estimation, s/he can immediately change the query condition to stop the first query and start the second query. S/he could also wait a bit longer for better quality, say, using 1.5 seconds, system now reports the average electricity usage for the 1st query as 982 kWh with a standard deviation of 5 kWh and 98% confidence.

STORM uses *spatial online sampling* to achieve its objectives. In particular, spatial online sampling continuously returns randomly sampled points from a user specified spatio-temporal query region, until user terminates the process or enough samples have been obtained to meet an accuracy requirement. An unbiased estimator, tailored towards a given analytical query, is built using the spatial online samples, and its approximation quality improves in an online fashion while more samples are being returned.

To make it easy for users and different applications to enjoy the benefit of spatio-temporal online analytics and aggregation, STORM also implements a data connector, so that it can easily import data in different formats and schemas, and enable spatio-temporal online analytics over such data without much efforts. Lastly, it features a number of built-in analytical modules so that a set of common analytical tasks can be executed without further engineering efforts once data have been imported. More complex and other analytical tasks can be built in a customized fashion.

Demonstration proposal. This demonstration proposal describes the design of STORM, and explains its key technical ideas. It also presents a detailed demonstration plan, and some evaluations to illustrate the advantage of STORM.

- We formalize spatial online sampling and online analytics in Section 2.
- We describe the design of the STORM system in details, and explain its key technical ideas in Section 3
- We present a small set of performance evaluations to illustrate the superiority of the STORM design over a few baselines, and a detailed plan for the demonstration of the STORM system in Section 4.

Lastly, we review related works in Section 5.

2. OVERVIEW

Random sampling is a fundamental and effective approach for dealing with large data sets, with a strong theoretical foundation in statistics supporting its wide usage in a variety of applications that do not require completely accurate answers. The use of random sampling for approximate query processing in the database community also has a long history, notably with line of work on *online aggregation* [7].

In online aggregation, instead of evaluating a potentially expensive query until the very end, we repeatedly take samples from all tuples that satisfy the query condition, and continuously compute the required aggregate based on the sampled tuples returned so far. The accuracy of the computed aggregate gradually improves as we get more and more samples, which is measured by *confidence intervals*, and the user may stop the query processing as soon as the accuracy has reached a satisfying level. Recently, online aggregation has received revived attention [11,18], as an effective tool for answering "big queries" that touch a huge number of tuples but the user can often be satisfied with just an accurate enough estimate.

However, past work on online aggregation has focused on relational aggregates, group-by, and join queries [5,7,11,18], on relational data. Motivated by the needs for *interactive spatio-temporal exploration and analytics* as explained in Section 1, we build the STORM system to achieve spatial online analytics and aggregation over spatial and spatio-temporal data. Since the statistical side of online aggregation is relatively well understood [5,7,11,13,18], which we will discuss briefly in Section 3, the key challenge essentially reduces to that of *spatial online sampling*, i.e., how to repeatedly sample a tuple from a spatio-temporal query until the user says "stop". This is formally defined as follows.

Definition 1 (Spatial online sampling) Given a set of N points P in a d-dimensional space, store them in an index such that, for a given range query Q, return sampled points from $Q \cap P$ (with or without replacement) until the user terminates the query.

Spatial online aggregation is a direct product of spatial online sampling, where online estimators for different types of spatial and spatio-temporal aggregates, like sum or average, are built using spatial online samples. A spatial online estimator ideally should be an *unbiased estimator*, and its estimation quality, characterized by confidence intervals, improves over time in an online fashion while more spatial samples are obtained. This concept can be further generalized beyond simple aggregates to *spatio-temporal online analytics* that covers a wide range of analytical tasks, like spatial clustering, spatial kernel density estimate (KDE). More details on this topic are provided in Section 3.

The proposed STORM system (spatio-temporal online reasoning and management) uses spatial online sampling to build spatio-temporal online estimators. STORM builds on a cluster of commodity machines to achieve its scalability. It uses a DFS (distributed file system) as its storage engine. As a result, it integrates with many existing distributed data management systems seamlessly, such as Hadoop and Spark. In particular, we have based the development of STORM on top of a distributed installation of MongoDB, which uses a DFS and the JSON format for its record structures.

The STORM system provides a *query interface* that supports a number of commonly encountered analytical queries on spatio-temporal data sets, such as basic spatio-temporal aggregations. It also includes a few more complex, advanced analytical queries such as kernel density estimate, trajectory reconstruction, semantics analysis (on short-text data) to illustrate the wide applicability of spatio-temporal online analytics by building a customized online estimator.

It also uses a *data connector* to connect to different data sources in order to import or index data from different storage engines such as excel spreadsheets, relational databases, a key value store such as Cassandra or HBase.

Lastly, it exposes a set of library and APIs, and a query language to enable users to build customized spatio-temporal online analytical tasks. An overview of the STORM system is shown in Figure 1.

Demonstration overview. The demonstration consists of four different components, namely, basic analytics, data import, updates, and customized analytics.

In *basic analytics*, we will showcase spatio-temporal online analytics in STORM using a number of data sets that are already imported and indexed by STORM. Our data sets include a massive national atmospheric measurement network data from nearly 40,000 weather stations from the MesoWest project, and data from various social media data sources (in particular, a growing subset of twitter data from July, 2013 to present). Users are able to interactive with the system by issuing basic analytical queries in a map-based query and analytical interface such as spatio-temporal aggregations on MesoWest data (e.g, the average temperature reading from

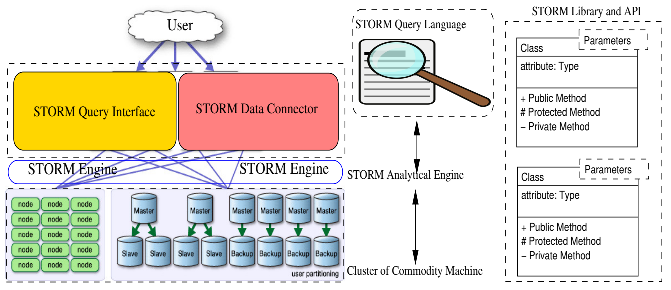


Figure 1: The STORM system overview.

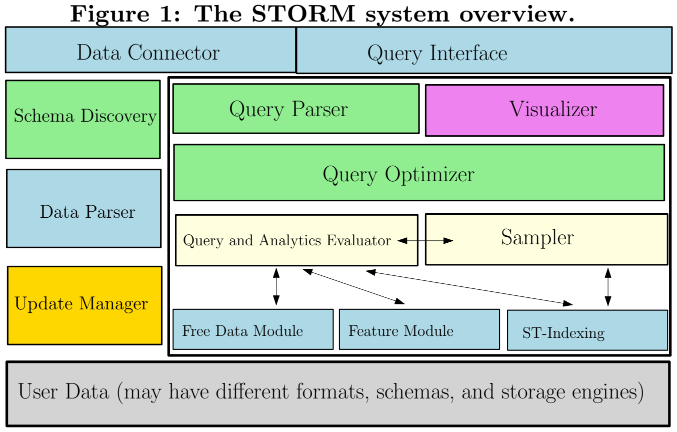


Figure 2: The STORM engine architecture.

a spatio-temporal region), population density estimate for a spatio-temporal region using KDEs over twitter data.

In *data import*, a user may import a data set from an external data source during the demo. Through its data connector module, STORM supports the import and indexing of data from a number of different storage engines, such as excel spreadsheets, MySQL, Cassandra, MongoDB. Users have the option of either importing the data into the STORM storage engine which is based on JSON format in a distributed MongoDB installation, or simply indexing the data through the data connector (without importing the data into the STORM storage engine).

In *updates*, we will demonstrate how STORM supports adhoc data updates efficiently. Its novel spatial online sampling module is able to update its indexing structure to reflect the latest state of the underlying data sets, so that a correct set of online spatio-temporal samples can always be returned with respect to the latest records in a data set.

Finally, in *customized analytics*, we will show how to construct a customized online analytical estimator for a spatio-temporal analytical query.

3. THE STORM SYSTEM

The design of the STORM engine is shown in Figure 2. In what follows, we explain the key technical ideas behind the *sampler* and the *ST-indexing* in details, and only briefly introduce the other modules.

3.1 The Sampler and ST-Indexing

The key objective of the *sampler* is to return spatial online samples as defined in Definition 1. Specifically, under the request of query and analytics evaluator, the *sampler* continuously returns independent random samples, one at a time, from a user specified spatio-temporal query region. In most

P	The raw data set in $\mathbb{R}^d$.
k	The number of samples to report.
N	$\lvert P\rvert$, the size of the raw data set.
Q	A range query in $\mathbb{R}^d$.
P_Q	$P \cap Q$, elements in the query range.
q	$\lvert P_Q\rvert$, the number of elements in the query range.
$u, v, \cdots$	Tree nodes.
$T(u)$	The subtree rooted at node u.
$P(u)$	The set of all data points covered by $T(u)$.
R_Q	The canonical set for Q.
$r(N)$	The size of a canonical set in a R-tree of size N.
B	The size of a disk block.

Table 1: Notation used in the paper.

cases, we need a relatively small sample whose size increases over time, i.e., k samples where $k \ll N$. Queries are *online and continuous*. The evaluator may keep asking for samples until satisfied with those returned (to meet an accuracy or time requirement). This means that k is *unknown* until the query is terminated by the system, somewhat like data stream algorithms. In fact k is never given to the sampler as a parameter, and we will design methods that efficiently extract *one* sample at a time. Hence, this problem can also be interpreted as asking the *sampler* to return k samples for an *arbitrary* (integer) value of k, from the set of $P \cap Q$.

The notation listed in table 1 will be used to describe the sampling procedures. The two most straightforward methods for this problem would be QueryFirst and SampleFirst:

QueryFirst Calculate $P \cap Q$ first, then repeatedly extract a sample from the pre-calculated set upon request.

SampleFirst Upon request, pick a point randomly from P and test if it is within Q. Return the sample if so, otherwise dispose it and repeat.

The running time of QueryFirst is $O(r(N) + q)$, the same as a full range reporting query. For SampleFirst, because a randomly picked point falls inside Q with probability q/N, we expect to draw $O(N/q)$ samples in order to see one inside. Thus, the expected cost of SampleFirst is $O(kN/q)$. This could be good for very large q, say, a query that covers a large constant fraction of P. However, for most queries, this cost can be extremely large. If $q = 0$, it never terminates.

A better solution is to adapt the random sampling method of Olken [15] to R-trees. His method takes a sample from P_Q by walking along a random path from the root down to the leaf level. When deciding which branch to take, the subtree sizes $|P(u)|$ are considered so that the probabilities can be set appropriately. This way, a sample can be obtained in $O(\log N)$ time. Over k samples, the total time is $O(k \log N)$. We call this method RandomPath. It is reasonably good, but only in internal memory. When the R-tree resides on disk, each random path may involve a traversal in a completely different part of the R-tree, resulting in at least $\Omega(k)$ I/Os in total, which is very expensive.

To further improve the efficiency and achieve better scalability, STORM uses a *ST-indexing* module (spatio-temporal indexing) to facilitate the *sampler* to retrieve spatial online samples. Two different indexing schemes are introduced.

The first index structure, LS-tree, is based on the "level sampling" idea. We independently sample elements from P_i with probability $1/2$ to create P_{i+1}, and stop when the last P_ℓ is small enough, in expectation $\ell = O(\log N)$. Then we build R-tree T_i for each P_i. We set $P_0 = P$. Since their sizes form a geometric series, the total size is still $O(N)$.

Upon a query Q, we simply execute an ordinary range reporting query on the R-trees in turn $T_\ell, T_{\ell-1}, \ldots, T_0$. Note that from T_i, each reported point is sampled with probability $1/2^i$ independently, and all must fall inside Q. Thus, they form a probability-$(1/2^i)$ coin-flip sample of P_Q. To turn this into a sample without replacement, we perform a random permutation, and start to report the points to the user one by one, until the user terminates the query, or all samples are exhausted. In the latter case, we move on to the next R-tree T_{i-1}. Since $P_j \subseteq P_i$ if $j > i$, we need to make sure no sample is reported twice, by maintaining a set of all the samples that have been reported for the running query Q.

Suppose the user terminates the query after receiving k samples. Then in expectation, we have reached tree T_j such that $q/2^j \approx k$, i.e., $j = \log(q/k)$. Thus, the total query cost (in expectation) is $O(k) + \sum_{j=\log(q/k)}^{\ell} r\left(\frac{N}{2^j}\right)$. This solution works well in external memory, since the query on each R-tree is just a normal R-tree range query. As a result, the term $O(k)$ does not lead to $O(k)$ IOs for disk-based data sets; rather we expect $O(k/B)$ IOs where B is the block size.

Note that distributed R-trees are used when applying the above idea in a distributed cluster setting.

But LS-tree needs to maintain multiple trees, which can be a challenge especially in distributed settings and/or with many updates. We can further improve this method by maintaining only one R-tree. The key idea is to associate a set of samples within each R-tree node, and visit as few nodes as possible. To ensure correctness and scalability, a number of effective ideas are employed:

Sample Buffering: We associate a set $S(u)$ of samples for each R-tree node u. $S(u)$ is obtained from the canonical cover of u. The size of $S(u)$ is properly calculated.

Lazy Exploration: We also maintain a count for each node u which is the number of data elements covered by u from the leaf level. Then, we can use a carefully constructed weighted sampling technique to save unnecessary exploration of nodes.

Acceptance/Rejection Sampling: Subtrees rooted at nodes in R_Q (the canonical set of Q) vary in size. The acceptance/rejection sampling is used to quickly locate large subtrees in R_Q. Observe that if we take a small set of random samples from $P \cap Q$, the larger subtree it is, the more likely we will take samples from it. The smaller subtree it is, the more time is necessary to locate it. So we want to avoid exploring small subtrees in R_Q which are expensive yet relatively useless.

Integrating the above ideas in a single R-tree leads to the design of the second indexing structure in the ST-indexing module, namely, the RS-tree. In particular, we develop RS-tree based on a single Hilbert R-tree over P. A distributed Hilbert R-tree is used to work with the underlying distributed cluster. For brevity, we omit the technical details for the construction and analysis of the RS-tree.

Lastly, since both LS-tree and RS-tree leverage on R-tree as its main data structure, supporting ad-hoc updates is easy, as long as we properly update the associated samples in the process. These technical details are also omitted for brevity.

3.2 Other Modules

The other modules in STORM are more or less similar to common modules found in a data management system. Its query interface supports a keyword based query language with a query parser, where predefined keywords are used to specify an aggregation or an analytical task that are already supported in the system. A temporal range and a spatial region (on a map) are used to define a spatio-temporal query range. A set of online estimators for common spatio-temporal aggregations and analytics are included in the *feature module*, which builds these estimators using spatial online samples.

Note that the statistical side is relatively well understood [5, 7, 11, 13, 18]. Essentially, any aggregate of the whole population can be estimated from a sample set, and the accuracy improves as more samples are obtained and the sample size increases. Suppose each point e in our data set is associated with an attribute $e.x$ of interest. Then for example, it is well known that the sample mean is an unbiased estimator of the real mean, i.e., letting S be the set of k samples returned and P_Q the set of all points in the query range, we have $E[\bar{X}] = E\left[\frac{1}{k}\sum_{e\in S} e.x\right] = \mu = \frac{1}{q}\sum_{e\in P_Q} e.x$.

Furthermore, by the central limit theorem, $X - \mu$ approaches $Normal(0, \sigma^2/k)$, where σ is the population standard deviation. This means sample variance is inversely proportional to sample size, and we expect to have a quality estimate with even when k is small. We can also estimate σ^2 from the sample, and further compute the confidence intervals, as in standard online aggregation [7].

In the spatial setting, there are more complicated statistics than simple aggregates like sum or mean. A widely used one is the *kernel density estimation (KDE)*, which construct a continuous spatial distribution from discrete points. Specifically, the distribution density at some point p is computed as $f(p) = \frac{1}{q}\sum_{e\in P_Q} \kappa(d(e,p))$, where $d(\cdot,\cdot)$ is the distance between two points, and $\kappa(\cdot)$ is the *kernel function* that models the "influence" of e at p, usually a decreasing function of distance. Then we can compute $f(p)$ at regularly spaced points (say, all grid points), and construct a density map of the underlying spatial distribution. We observe the distribution density at each point, $f(p)$, is still an average, so we can compute an approximated density map by drawing a sample from P_Q, and derive the confidence interval (for each point p).

Other spatial analytics tasks, such as clustering, can also be performed on a sample of points. Intuitively, the clustering quality also improves as the sample size increases. The STORM APIs allow a user to access the sampler and feature module directly to build complex, advanced, customized online estimators, with user-derived, operator-specific guarantees for confidence interval and approximation quality.

The data connector uses schema discovery and data parser for a number of data sources that are supported in STORM in order to import and index a data source from a specified storage engine. Additional storage engines can be added by extending the code-base for the data connector.

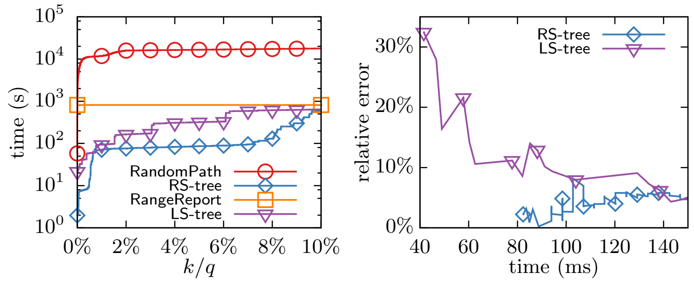
The query optimizer implements a set of basic query optimization rules for deciding which method (as we have discussed in Section 3.1) the sampler should use when generating spatial online samples for a given query. The visualizer implements a number of basic visualization tools to enable visualizing the results from an online estimator, such as visualizing density estimate from KDE. The update manager handles data updates for data sets currently indexed by STORM. Lastly, the free data module is used to convert between different record formats and JSON format, as used by the storage engine of STORM.

4. EVALUATION AND DEMONSTRATION

4.1 The performance of STORM

We carried out extensive experiments to evaluate the performance of STORM. In particular, the performance of STORM is compared against competing baselines for generating spatial online samples and executing spatio-temporal online analytics. A detailed report of these results is beyond the scope of this demonstration proposal.

We only report two results, concerning the query efficiency and the estimation accuracy respectively, both in an online fashion, using the full open street map (OSM) data set.



(a) query efficiency: vary k. (b) query accuracy: relative error on avg(altitude).

Figure 3: Query performance in STORM.

Figure 3(a) shows the time taken for different methods (as discussed in Section 3.1) to produce spatial online samples of increasing size, where we fixed a spatio-temporal range query Q with $q = 1$ billion ($q = |P_Q|$). Clearly, LS-tree and RS-tree perform much better than competing baselines.

Figure 3(b) shows how the relative error improves with respect to the increase in query execution time for a spatio-temporal aggregate query where it estimates the avg(altitude) for all points in a user-specified spatio-temporal query range. It clearly indicates that STORM is able to produce online estimations whose approximation quality improves over time.

4.2 The demonstration of STORM

The STORM system is available at `http://www.estorm.org` with username *guest* and password *guest@storm*.

Figure 4 shows the user interface of STORM. As introduced in Section 2, the demonstration of STORM consists of four components. In *basic analytics*, users may select different built-in analytical queries and data sources that are already indexed by STORM, to experience the benefits of spatio-temporal online analytics. For example, using the twitter data set and the online KDE estimator, we can estimate population density over an arbitrary spatio-temporal regions based on the location and timestamp of the underlying tweets *interactively in real time in an online fashion*, as shown in Figure 5 when user zooms out from Salt Lake City to the entire United States for tweets in last 30 days. The density estimate improves its accuracy with better visualization results as query time increases.

The user is also able to interact with the MesoWest data to issue spatio-temporal online aggregations over the MesoWest data (`http://mesowest.utah.edu/`).

In *data import*, we allow a user to import data from an external data source. Currently, STORM supports importing and indexing data from excel spreadsheets, text files, Cassandra, MySQL, and MongoDB. In particular, we will walk through the steps for importing a new data source from a plain text file and a MySQL database respectively. Once

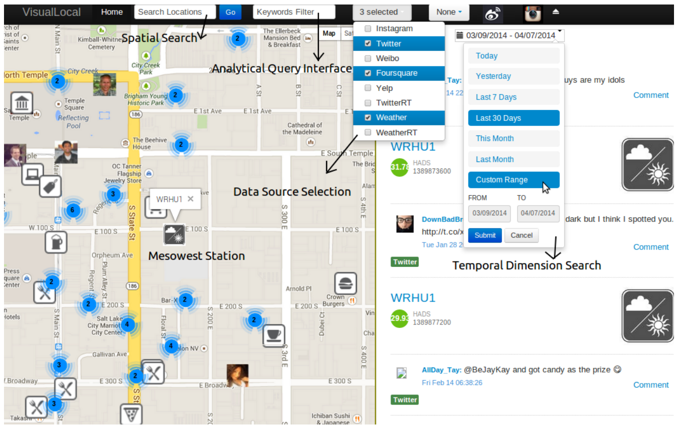
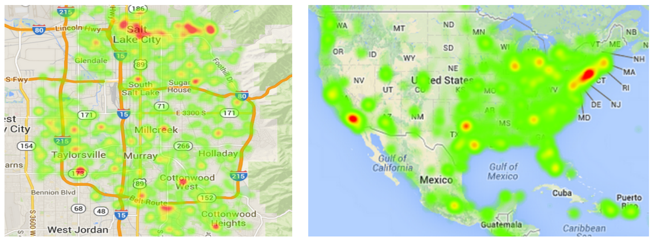


Figure 4: Overall query interface in STORM.



(a) SLC KDE. (b) USA KDE.

Figure 5: Interactive, online analytics.

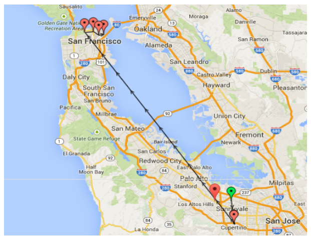
imported, users are able to interact with the new data sets with the basic analytical queries supported in the system.

In *update*, we will make updates to an existing data set (e.g., the twitter data set in STORM is constantly updated with new tweets using the twitter API), and illustrate that STORM has successfully incorporated their impacts to analytical results by issuing analytical queries with time range that narrows down to the most recent time history.

Lastly, in *customized analytics*, we will showcase how to build advanced and more complex online analytics in STORM by accessing its *feature module* and *sampler* directly. We will use two examples for this purpose.

In the first example, we show how to build an online, approximate trajectory using spatial online samples for a given twitter user for a specified time range, using location and timestamp information from his/her tweets. The end result is shown in Figure 6(a). In the second example, we show how to perform online short-text understanding using online samples of tweets for an arbitrary spatio-temporal query range. There was a highly anomalous heavy snow in the Atlanta area in the days between February 10 and February 13, 2014. To see how the citizens of Atlanta reacted, we used a spatio-temporal window on downtown Atlanta during that period, and used our short-text understanding online estimator for twitter data in STORM; shown in Figure 6(b). We can quickly observe that the population was quite unhappy and frustrated, particularly considering the highlighted terms snow, ice, outage, shit, hell, why. Another interesting observation from this example is that STORM enables integrated online data analytics from multiple data sources; in this case, user can interactively explore both MesoWest data (to confirm the heavy snow) and the twitter data.

In both examples, we will show how to program a customized analytical task using the built-in feature module and spatial online samples returned from the sampler. Users


(a) online approximate trajectory construction.

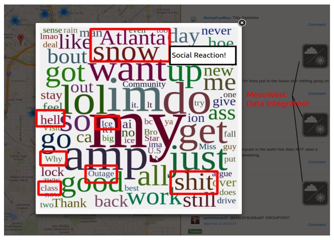

(b) spatio-temporal short-text understanding.

Figure 6: Advanced, customized online analytics.

may also import third party libraries to facilitate the implementation of a customized analytical task.

5. RELATED WORK

The concept of online aggregation was first proposed by Hellerstein et al. in [7], and has been revisited for different operators (e.g., join [5], group-by [19]) for relational data models, and computation models (e.g., MapReduce [18]). The standard approach is to produce online samples and build estimators that improve accuracy gradually over time using more and more samples [5–7, 18]. The connection from query accuracy (especially for standard aggregations) and estimation confidence to sample size is mostly well understood, see [5–7, 13, 18, 20] and many other work in the literature on building various kinds of estimators using random samples. Nevertheless, to the best of our knowledge, a comprehensive system such as STORM that supports spatio-temporal online analytics has not been investigated before.

Our work is closely related to online sampling and sampling from a database in general. Olken proposed the idea of taking a random sample from P_Q by walking along a random path from the root down to the leaf level in his PhD thesis [15]. This idea works for both B-tree in one dimension and R-tree in higher dimensions [15–17]. However, as explained in Section 3 and confirmed by experiments in Section 4.1, this method is too expensive for generating online samples in large spatial databases.

Hu et al. [8] recently showed how to produce samples for range queries with a new constraint that samples must be independent with respect to both intra-query and inter-queries. Their result is purely theoretical, and is too complicated to be implemented or used in practice. It holds only for one-dimensional data and their external memory data structure is static and does not support dynamic updates.

There is an increasing interest in integrating sampling as an operator in a database management system; see recent efforts in [1, 2, 9, 10, 12, 14, 21, 22]. Nevertheless, none have investigated spatial and spatio-temporal databases.

Finding samples from a set of geographic points for better map visualizations is described in [3]. Samples are taken from the data set such that the samples will be evenly distributed when the sampled data is drawn on a map. This differs from STORM, as their definition of *spatial sampling* has a different objective, which is to produce better visual representation of the underlying data.

The most closely related work is SpatialHadoop [4], which is a comprehensive system for building spatial and spatio-temporal analytical tasks on large spatio-temporal data, using the MapReduce computation framework. However, a fundamental difference between STORM and SpatialHadoop is *online* versus *batched offline* analytics. As a result, STORM and SpatialHadoop nicely complement each other and satisfy different application scenarios and user needs.

6. ACKNOWLEDGMENT

Robert Christensen, Feifei Li, Jun Tang, and Natalee Villa are supported in part by NSF grants 1443046 and 1251019. Feifei Li is also supported in part by NSFC grant 61428204 and a Google research award. Lu Wang and Ke Yi are supported by HKRGC under grants GRF-621413 and GRF-16211614, and by a Microsoft grant MRA14EG05.

7. REFERENCES

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.
[2] S. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. In *PVLDB*, 2012.
[3] A. Das Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy. Efficient spatial sampling of large geographical tables. In *SIGMOD*, 2012.
[4] A. Eldawy and M. F. Mokbel. A demonstration of SpatialHadoop: An efficient mapreduce framework for spatial data. *PVLDB*, 2013.
[5] P. Haas and J. Hellerstein. Ripple joins for online aggregation. In *SIGMOD*, pages 287–298, 1999.
[6] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM*, 1997.
[7] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *SIGMOD*, 1997.
[8] X. Hu, M. Qiao, and Y. Tao. Independent range sampling. In *PODS*, 2014.
[9] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. The monte carlo database system: Stochastic analysis close to the data. *ACM TODS*, 36(3):18, 2011.
[10] P. Jayachandran, K. Tunga, N. Kamat, and A. Nandi. Combining user interaction, speculative query execution and sampling in the DICE system. *PVLDB*, 2014.
[11] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *ACM Transactions on Database Systems*, 33(4), Article 23, 2008.
[12] A. Klein, R. Gemulla, P. Rösch, and W. Lehner. Derby/s: a DBMS for sample-based query answering. In *SIGMOD*, 2006.
[13] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
[14] S. Nirkhiwale, A. Dobra, and C. M. Jermaine. A sampling algebra for aggregate estimation. *PVLDB*, 2013.
[15] F. Olken. *Random Sampling from Databases.* PhD thesis, University of California at Berkeley, 1993.
[16] F. Olken and D. Rotem. Random sampling from B+ trees. In *VLDB*, 1989.
[17] F. Olken and D. Rotem. Sampling from spatial databases. In *ICDE*, 1993.
[18] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. In *PVLDB*, 2011.
[19] F. Xu, C. M. Jermaine, and A. Dobra. Confidence bounds for sampling-based group by estimates. *ACM TODS*, 33(3), 2008.
[20] Y. Yan, L. J. Chen, and Z. Zhang. Error-bounded sampling for analytics on big sparse data. *PVLDB*, 7(13), 2014.
[21] K. Zeng, S. Gao, J. Gu, B. Mozafari, and C. Zaniolo. ABS: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, 2014.
[22] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.