Spatial-Aware Approximate Big Data Stream Processing

Isam Mashhour Al Jawarneh, Paolo Bellavista , Luca Foschini , Rebecca Montanari

Dipartimento di Informatica – Scienza e Ingegneria, University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{isam.aljawarneh3, paolo.bellavista, luca.foschini, rebecca.montanari}@unibo.it

***Abstract*— The widespread adoption of ubiquitous IoT edge devices and modern telemetry has generated an unprecedented avalanche of spatially-tagged datasets, which if could interactively be explored, would offer relevant insights into interesting natural phenomena. Online application of spatial queries is expensive, a problem that is further inflated by the fact that we, more than often, do not have access to a full dataset population in non-stationary settings. As a way of coping up, sampling stands out as a natural solution for approximating estimators such as averages and totals of some interesting correlated parameters. In any sampling design, representativeness remains the main issue upon which a method is regarded good or bad. In a loose way, in a spatial context, this means fairly sampling quantities in a way that preserves spatial characteristics so as to provide more accurate approximates for spatial query responses. Current big data management systems either do not offer over-the-counter spatial-aware online sampling solutions or, at best, rely on randomness, which causes too many imponderables for an overall estimation. We herein have designed a QoS- spatial-aware online sampling method that outperforms vanilla baselines by statically significant magnitudes. Our method sits atop Apache Spark Structured Streaming's codebase and have been tested against a benchmark that is consisting of millions-records of spatially-augmented dataset.**

***Keywords*— Spatial Sampling, Spark Streaming, Z-order curves, stratification, dimension reduction.**

I. INTRODUCTION

The exponentially increasing adoption of Internet of Things (IoT) catalyzes a fast-track advancement of a breed of big data management ecosystems, ultimately aiming at real-time deep insightful exploration for guiding strategic decision making in all aspects of our lives, including environmental and business issues. Most importantly, current efforts in the relevant state-of-art are geared toward promoting a constellation of components dubbed collectively as Stream Processing Engines (SPE) [1]. Low-latency and high-accuracy yet remain the two largely antithetical requirements that are guiding the way through which those systems are operating, where SPEs seek trading them off in a way that can satisfy prescribed Service Level Agreements (SLAs). Side by side with that in mind, with a strict throughput/latency balance requirement by several emerging scenarios, more attention is now given to a family of online computations that is known as Approximate Query Processing (AQP). Its increasingly swift adoption is due, in large part, to the fact that users, more than often, are satisfied with approximations and are willing to trade an error-bounded accuracy for even a small latency gain. This in its essence, means the dependence on sampling, which is loosely defined as selecting subsets of population census, aiming at finding approximate answers for statistical computations. However, one of the main challenges is selecting a high-quality sample with an acceptable degree of error-bounded guarantees [2]. Distributions of many real-world datasets are proved to be uniform- and Gaussian-like, which caused early online sampling methods to rely on randomness. On the downside, random sampling produces good results only when data distribution is uniform but otherwise has poor quality on skewed geospatial datasets. In reality, data streams are mostly spatially-tagged, giving that they naturally originate in IoT devices, which mainly aim at incorporating location-awareness into insightful analytics. For example, an environmental study, through data snooping, seeks at disclosing reasons behind the fact that global warming is aggressively affecting specific regions on our earth more than others. Current SPEs mainly focus on balancing the throughput/latency colliding requirements, without considering spatial characteristics of arriving data streams, thus rendering their current versions unsuitable for geospatial scenarios. The closest works from relevant literature [3-5] apply various dimensionality reduction approaches, but however are computationally expensive and inapplicable in distributed online deployments.

In this work, we have made the following contributions. We design online spatial-aware sampling strategies for real-time estimation of spatial parameters, such as totals and averages of spatial-oriented target variables. We specifically have designed a novel spatial-aware online sampling method that acts dynamically on top of a state-of-art micro-batch-based SPE representative, Specifically Spark Structured Streaming [6], (SpSS as a shorthand). Simply put, our method acts on dimensionality reduction, thus transforming the two-dimensional into a one-dimensional space by utilizing specifically a space-filling curves (SFC)-based method (precisely, Geohashing). We divide the study area into equal-sized polygons (currently square-shaped) and sample equal-fraction quantities from each polygon, thus preserving spatial co-locality and characteristics. Our method resorts to stratification. Converting GPS data into geohashes offers the ability to capture every set of units (in one borough in city administrations terms, for example) under the same geohash value, thus demystifying the process of spatial-aware online sampling. Our second contribution is that we support a wide spectrum of well-established baseline statistical approximate (for spatially-rich data) queries in a streaming fashion, thus serving results interactively and incrementally, where estimates gain more accuracy as time tick forward. Those queries include

single-value estimators (such as totals and averages) and aggregations (including order and rank statistics, such as Top-N). To the best of our knowledge, we are not aware of any system from the relevant literature that achieves these goals. We first introduce the theory behind our work. We then shortly recapitulate the implementation of the system. In what follows, we present our results with proper discussions. We finalize by summing up the effort and recommend future research directions.

II. Background and a Brief Primer on Sampling

A. The Importance of Sampling

Solutions designed for exact responses are inconvenient for fast arriving spatial data streaming loads that are fluctuating in arrival rates and skewness. For example, measuring the average trip distance travelled by taxis from each borough in NYC, the United States, can never be exactly accurate, giving that we do not have access to a total population. Instead, data is arriving in streams that normally exhibits temporal density and skewness [7]. Luckily, in geo-statistics, such measurements can be totally based, with statistically significantly acceptable error-bounds, on approximation [8]. That said, a representative sample can be potentially utilized for approximating estimators such as averages and proportions of an interesting study target variable. Another reason for sampling is the fact that observing all interesting variables in a location could be prohibitively impractical, such as an abundance of migrating birds in a specific region, which normally exhibit uneven spatial distribution [9].

B. The Elegance of Spatial Representativeness in Online Sampling Designs

Current SPEs, and their associated spatial-aware patches and glues, mainly focus on achieving a balance between low-latency and high-accuracy by either provisioning extra computing resources (scaling in/out) or dropping-off (a.k.a. sampling) part of the arriving data, thus sacrificing accuracy for latency. The former is not favourable because it tends to under-use resources in restrictive settings, and overflow in permissive settings. To achieve the latter, current SPEs employ sampling designs that chiefly embrace randomness, following trends such as those appearing in Simple Random Sampling (SRS) [10] , thus not accounting for spatial information taken from close-by locations. However, in spatial patchy distributions (where units are clumped into few patches), SRS does not interplay well in either approximating estimators or predicting unseen values. Simply put, it may unfairly select random quantities with uneven fractional rates for all autonomous regions of the study (those resembling strata in stratified sampling, explained shortly), even if it hits the absolute center of the target at times, at some other time it will not, or even much at all. Also, it is commonly agreed among statisticians that spatial data maintains spatial trends which affect the observed responses, which stems from the fact that spatially-collocated units are normally affected by the same set of surrounding environmental, ecological or locational factors, such as those related to anthropogony [4, 5, 9]. Having said that, it naturally implies that selecting spatially well-spread-out samples positively affects the accuracy of estimators. We refer to such kinds of samples as *spatially representative samples*. Further, despite the scarce abundance of some statistical spatial sampling methods, they mostly do not consider continuous populations, perhaps most importantly attributed to the fact that system's computational capacity of the time was prohibitive. Nowadays, with advancements in SPEs, specifically those based upon MapReduce (MR) frameworks [11] , it is possible to apply such statistics online.

III. Spatial Aware Approximate Data Stream Processing System Overview

A. Usage Model and Baseline System

Visualization by map rendering systems provides an insightful look at spatial data. However, those systems are typically space-limited and only capable of observing a fraction of arriving data streams. Consider an interactive continuous query (CQ) that asks to "generate a real-time heatmap showing people and objects, such as NYC taxicabs, in-move". Plotting all arriving spatial objects easily cause a clutter. Sampling thus is a natural solution. A baseline that is based on simple random probability sampling (such as the SpSS-based SRS baseline) tends to overlook regions, generating heatmaps that do not necessarily reflect reality. On the contrary, a sampling design that selects a well-representative spatial sample is favourable as it preserves spatial density in all regions, avoids selection unfairness, and generates more realistic maps. This represents one usage model of our SAOS sampling design (described shortly).

B. Design Premises

We have designed SpatialSPE (short for spatial stream processing engine) so that it operates with the following premises and presumptions, which we consider as future perspectives. There is a cost module acting as a controller that maps SLAs requirements (the contradicting response's accuracy level and latency requirements) into an adaptive sampling fraction. We leave the consideration of peculiarities of such a controller to a future work. Also, we currently consider equal-sized square polygons represented by corresponding geohashes. We consider other possibilities (such as variable-sized random shaped polygonal areas) in a future work.

C. SpatialSPE

We have designed a spatial aware approximate data stream processing system (we dub SpatialSPE hereafter for short) for online big geospatial data stream processing approximate analytics. The context diagram in Fig. 1 depicts a high-level architecture of SpatialSPE's workflow, which acts as an integral

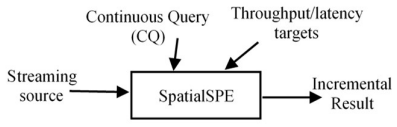

Fig. 1. SpatialSPE workflow

component of the overview tier in the system's usage model. Data is coming from heterogeneous sources (spatially-tagged), to be then served as an unbounded input table (in SpSS parlance) at regular basis (batch intervals, a.k.a. trigger intervals in SpSS jargon). As data hits this stage, SpatialSPE starts its unbounded operation. At a front-stage resides our spatial-aware stratified-based sampling method (described shortly) as a pivotal module in our design.

We currently assume that latency/throughput targets are served to the system externally by the user. Based on those figures, we sample specific fractions of the input arriving data tuples, thereby producing an output that has rigorous error bounds and serve it to the user interactively. For now, sampling fractions are the same for all constituent stratum (geohashes in our setting). In addition, we receive a CQ that will be incrementalized through our system. Particularly, SpatialSPE utilizes our spatial-aware sampling method (discussed shortly) for selecting a spatial-aware sample from the input stream that is proportionate to the latency/throughput targets.

An indispensable module of our design is a procedure that is calculating a list of geocodes (currently geohashes) covering a study area where data samples are collected. It basically receives a list of vertices that collectively form polygonal areas (known as neighborhoods, districts or boroughs in city management terms), thereafter our procedure explodes all geohashes that are covering all polygons , constructing a map that is then served to our sampling method . What remains incumbent then is applying our spatial-aware online sampling procedure. Our system then applies a user-defined CQ on the sample during that window interval and serves the result interactively to user. Results are incrementalized, meaning that they are served gradually to user, reflecting every change after each time window. Algorithm 1 shows the workflow of SpatialSPE.

Algorithm 1: SpatialSPE Workflow

```
/* latThrTargets: latency throughput targets, geoPrec: geohash
precision */
Input: stream, ContinuousQuery (CQ), latThrTargets, polygons,
geoPrec, seed
samplingMap ← ∅ //map of geohash keys and sampling fractions
coverGeo ← getCoverGeo (polygons, geoPrec) /* List of
geohashes covering study area */
//cost model computes the sampling fraction
sampFraction ← costProcedure(latThrTargets)
Foreach geohash in coverGeo do
   // construct a map, geohash: key, sampling fraction: value
   element ← map {geohash→ sampFraction}
   samplingMap.put(element)
End
Foreach time window interval do
   windowSample = ∅  // tuples sampled in current time window
   Foreach batchInterval in window interval do
        batchSample = ∅  //tuples sampled in current batch interval
        forall tuples_i in batch interval do
          /* apply SAOS on tuples of current batch interval: tuples_i */
                batchSample ← SAOS (tuples_i, samplingMap,
sampFraction, seed)
                windowSample.add(batchSample)
        End
  End
     //Compute and serve incremental output every time window
 incrementalOutput ← run (CQ, windowSample)
 return incrementalOutput with error bounds
End
```

D. Spatial Aware Online Sampling (SAOS) Algorithm

To simplify the spatial approximate real-time data processing, we have designed a novel method that we term as Spatial-Aware Online Sampling (SAOS), which then constitutes the main technological block of our SpatialSPE system. Our algorithm is efficient in the sense that it preserves spatial representativeness in addition to its streamlined injection within the layers of state-of-art micro-batch stream processing models such as SpSS. Having said that, it does not require a pre-knowledge of the streaming statistics, such as a total data population (that may arrive during a long lifespan), it otherwise depends on incrementalization offered by the underlying engine and builds on top of that.

The essence of our SAOS method is coded in the pseudocode of Algorithm 2. The workflow proceeds as follows: For each batch interval during a time window, we first compute a geohash corresponding to a geometric coordinates of an arriving tuple, then we use the resulting geohash in fetching a corresponding sampling fraction, and thereafter we use that to draw a random sample from each stratum independently, where each tuple within that stratum (geohash) receives an equal selection (aliased inclusion) probability. To take a utilitarian perspective, SAOS algorithm is analogous to the following heuristic overview. Imagining the earth flattened out, the process starts by placing a square grid over the study region, and then proceeds by randomly selecting a spatially-proportional fair number of tuples from each grid cell. Each grid cell represents a geohash in our setting. By doing so, we resort to stratified sampling, which is favorable over other designs in that it renders better statistical estimates for target variables of spatially-rich settings. We offer a fine-tuning capability through the control of geohash precision parameter at which the user can tune at different levels. This is visualized as follows; imagine that the grid starts initially as (2 x 2), where each geohash covers a cell, then changing the precision of geohash (number of bits on the orders of five, i.e. 30 bits geohash is equal to a string/digit combination of size six, five each) offers the ability to make the cells granular or coarser depending on needs. This process reduces a geographical two-dimensional space into one-dimensional space while preserving spatial locality and characteristics. We posit that the hybridization between z-order curves (from which geohash is a special case) and simple probability sampling (random in this setting, within the boundaries of each grid cell) yields a spatially well-representative sample, that, in turns, yields better estimator values for population target variables.

Algorithm 2: Spatial-Aware Online Sampling (SAOS)

```
SAOS (tuples_i, samplingMap, sampFraction, seed)
r = random(seed), S ← ∅
Foreach tuple in micro-batch-tuples do
   geohash ← geocode (tuple)
   //get sampling fraction for this geohash key = fraction_i, or zero
   fraction_i ← samplingMap.getOrElse(geohash,0.0)
   //toss a coin selecting items from each geohash in current batch
   If (P (r < fraction_i)) S.put(tuple)
   End
return S
```

E. Supported Queries

We currently support two basic groups of spatial queries, from which other complex queries (such as spatial clustering [12] and kernel density estimator (KDE) [13]) are seamlessly composable.

Group1 (G1). **Single queries** (a.k.a. **linear**). An example includes "finding an estimate of a population average for a target variable by only using a spatial sample". Since our method recovers stratified sampling in its core, then the theory of stratified sampling applies [10] , and hence, such an estimation problem is better formulated as follows: say we have in total K geohashes (each geohash is a stratum), y_{kj} represents a value of a j_{th} tuple in geohash k. t (pronounced tau) is the population total of stratum k. Then the population total of the target variable y can be estimated using our method by applying $\hat{t}_{SAOS} = \sum_{k=1}^{K} t_k = \sum_{k=1}^{K} N_k \bar{y}_k$. The average is thus estimated using SAOS by applying $\bar{Y}_{SAOS} = \hat{t}_{SAOS}/N = \sum_{i=1}^{I} (N_i/N)\bar{y}_i$. Since SpSS does not provide over-the-shelf solutions for such estimators, we have incorporated a patch for accomplishing that.

Group2 (G2). **Stateful aggregation queries** (a.k.a. **ensembles**). We specifically focus on Top-N ensembles, where we first apply SAOS to the arriving tuples, then we group arriving tuples by geohash (or neighborhood, borough, districts etc., at a coarser level) keys, and thereafter count the number of tuples in each geohash incrementally and order the resulting list in descending fashion. An example query in this category : "where do people tend to order taxi pickups in NY City in US".

F. Quantifying Uncertainty

Estimators through sampling are naturally bounded to a degree of uncertainty and quantifying that is essential.

I) For **single queries**, As our method resorts to stratified sampling, we rely on the Theory of Stratified Sampling [10] for estimating the accuracy of G1 queries approximations obtained through SAOS. We first use $\hat{v}(\hat{t}_{SAOS}) = \sum_{k=1}^{K} (1 - (n_k/N_k)) N_k^2 (s_K^2/n_k)$ to calculate the estimated variance of the estimated total. Then we carry that over to estimate the variance of the estimated average using $\hat{v}(\bar{Y}_{SAOS}) = \hat{v}(\hat{t}_{SAOS})/N^2$. Then, we calculate the standard error (SE) depending on $SE(\bar{Y}_{SAOS}) = \sqrt{\hat{v}(\bar{Y}_{SAOS})}$. Then we depend on $\bar{Y}_{SAOS} \mp z_{\alpha/2} SE(\bar{Y}_{SAOS})$ to approximate $100(1-\alpha)\%$ confidence interval (CI) for the population mean $\bar{Y}_{pop}$, where $z_{\alpha/2}$ is the upper $\alpha/2$ point of the normal distribution. Then we define $RE = z_{\alpha/2}(SE(\bar{Y}_{SAOS})/\bar{Y}_{SAOS})$ to calculate a relative error (RE). We also define the accuracy loss as accLoss = |estimatedMean – trueMean| / trueMean. In addition to those, we calculate the gain of applying SAOS (instead of the baseline), for which we define $gain_{SAOS} = \hat{v}(\bar{Y}_{SAOS}) / \hat{v}(\bar{Y}_{SRS})$, where $\hat{v}(\bar{Y}_{SAOS})$ is estimated variance from applying SAOS, whereas $\hat{v}(\bar{Y}_{SRS})$ is the estimated variance from applying SpSS based SRS.

II) For **stateful aggregations** (specifically Top-N) queries. We aim at measuring each method's ability in preserving the aggregation's original ranking. For this, we rely on Spearman's rank correlation coefficient [14] (abbreviated Spearman's rho hereafter). We have adopted this with a tiny retrofitting. Spearman's rho is a measure for statistical dependency between the ranking of two variables in a dataset. In brevity, our application proceeds as follows: we accumulate the ranks, and once the CQ stop (intentionally by user) we take the accumulated ranking of the original aggregations (w/o sampling) and the one calculated through SAOS (and comparatively speaking, through SpSS-based SRS baseline). and we serve to Spearman's rho, and thereby we apply $\rho_{rg} = cov(rank_{nos}, rank_{samp}) / (\sigma_{rank_{nos}} . \sigma_{rank_{samp}})$, where ρ_{rg} (pronounced rho) is spearman's correlation coefficient applied for ranking statistics , $cov(rank_{nos}, rank_{samp})$ is the covariance of the rank variables, $\sigma_{rank_{nos}}$ and $\sigma_{rank_{samp}}$ are the standard deviations of the rank variables, w/o and with sampling, respectively.

IV. SPATIALSPE REALIZATION

To show the excellence of SpatialSPE[1], we have built a standard-compliant prototype on top of SpSS, following the trending layered-up software stack. Since, as of this writing, SRS is not currently implemented for SpSS, and for a fair comparison, we have implemented a version of SRS that is able to operate in streaming settings and injected it as a glue to the SpSS for approximate estimations depending on SRS (we refer to this as SpSS-based SRS hereafter for short). This was only possible because we rely on the micro-batching mode of streaming operation, where streaming tuples are accumulated in blocks before being dispatched for processing in a Spark underlying job. Thus, by implanting a frontstage after the formation of a block and just before partitioning, an SRS can work as if it was operating in a batch mode (the essence of micro-batch stream processing).

V. PERFORMANCE EVALUATION AND RESULTS

A. Deployment Settings and Benchmarking

Dataset. For benchmarking, we use the NY City taxicab trips datasets [2], from which we choose a cohort of six months dataset (around nine million units) representing data captured through taxi rides for the first half of 2016. We choose the green taxi trip records, which include interesting fields capturing, most importantly, pick-up/drop-off locations and trip distances.

Deployment and experimental settings. We run our system, SpatialSPE, on a Microsoft Azure HDInsight Cluster hosting Apache Spark version 2.2.1. It consists of 6 NODES (2 Head + 4 Worker) with 24 cores. Head (2 x D12 v2) nodes , and Worker (4 x D13 v2) nodes. Each head node operates on 4 cores with 28 GB RAM and 200 GB Local SSD memory, and quantities are double those figures for worker nodes.

Throughput. We simply define throughput as the count of processed rows per second. For calculating the throughput for all systems, including the SpSS-based retrofitted SRS sampling

[1] The source code of SpatialSPE (including SAOS) is available at: https://github.com/IsamAljawarneh/SpatialSPE

[2] https://www1.nyc.gov

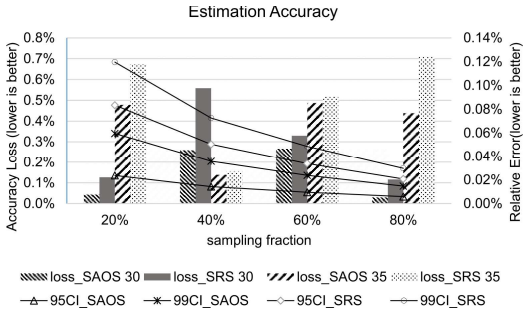

Fig. 2. Estimation accuracy of SAOS vs. SpSS-based SRS, for G1 queries

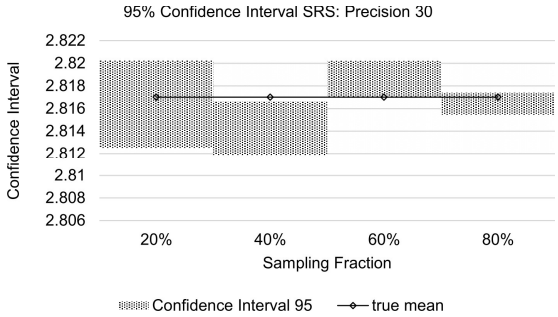

Fig. 4. confidence interval 95% of SpSS- based SRS on average estimator.

method that we have designed and our SAOS method, we utilize the StreamingQueryListener (from SpSS) to capture start and end timestamps and number of processed tuples, then we simply divide the latter by the total time elapsed during a CQ streaming session. Throughput is an interesting measurement because it captures the adaptability of system against fluctuating arrival rates of oscillating data streams.

B. *Results*

We report results we have obtained by measuring the metrics mentioned in § II. We depend on varying four parameters. Those are sampling fraction, arrival rate, geohash precision and computing power. As we have two categories of queries (refer to § II), we measure variations of metrics for each as follows. **Query G1** (linear). We have measured the performance based on the following linear query: "find the average trip distance of a NYC taxicab itinerary trip during the first six months of the year 2016". **Query G2** (Top-N). We use the following query to measure the performance of ensembles: "group NYC taxi trips count by specific region (region could be geohash at a granular level and neighbourhood at a coarser level, e.g. zoom-out) with descending order". We have applied the following combinations of parameter settings; each parameter setting's group is specialized in measuring a base category. In parameter setting 1, we focus on accuracy for both query groups, whereas in the parameter setting 2, we focus more on throughput.

Parameter settings 1. Varying geohash precision and sampling fraction. We vary geohash precisions from 30 to 35, and the sampling fraction from 20% to 80% (with a scale slide of 20% each). For this combination specifically, we aim at measuring the accuracy of both query groups G1 and G2.

The combinational graph in Fig. 2 shows the comparison between SAOS and SpSS-based SRS methods in the language of accuracy for the estimator of query in G1.

We notice from Fig. 2 that SAOS outperforms SpSS-based SRS for all geohash precision settings (30 and 35), for both measures, accuracy loss and relative error. Comparing SAOS to itself with differing geohash precision, generally speaking, we note that SAOS have bigger accuracy loss for geohash precision 35 (loss_SAOS 35 in Fig. 2), compared to SAOS accuracy loss at geohash precision 30 (loss_SAOS 30 in Fig. 2). This is attributed to that a wider geohash means smaller polygons, hence smaller number of keys per stratum and less precise stratification per batch interval that carries over a negative effect to the estimators and, consequently reduces the accuracy. Fig. 3 shows that under SAOS , for 95% of the possible samples of all fractions (20% to 80% from the total population size) the corresponding confidence intervals cover the true value of the population mean (a.k.a. average), whereas as shown in Fig. 4, SRS confidence intervals are susceptible to missing the true value, look at the case of sampling fraction 40%, where the average true value does not fall within the corresponding confidence interval and is also marginal on fraction 60%. The same trend occurs for the 68 CI (though not shown here for the lack of space). To better understand how SAOS is adept more than SRS in geo-statistics, we show in Fig. 5 the gain obtained by applying our method to G1 queries (calculated by applying $gain_{SAOS}$, refer to § III). Fig. 6 shows a comparison between our method SAOS and baselines in terms of accuracy for G2 queries. Ranking precision of SAOS outperforms those for SRS

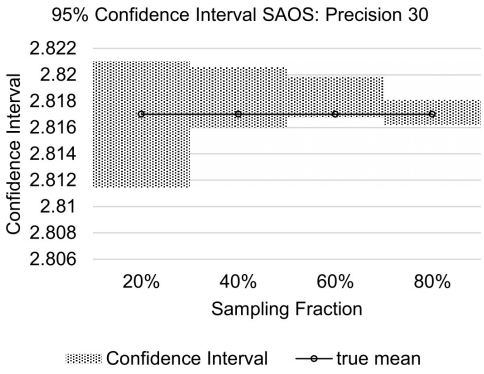

Fig. 3. confidence interval 95% of SAOS on average estimator.

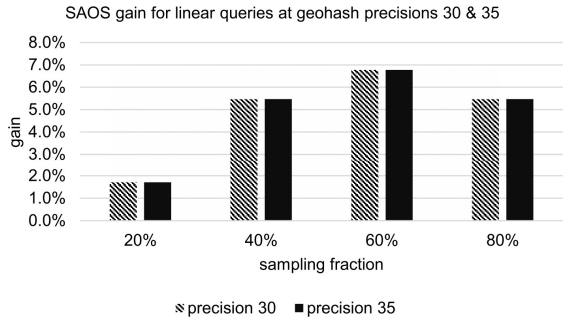

Fig. 5. Gain by applying SAOS instead of and SpSS-based SRS method

(though almost at par with SRS for fractions 40% and 80% with geohash precision 30).

Parameter settings 2. Fixing geohash precision at 30, and varying arrival rate from 1000K to 2000K tuples/second and sampling fractions between 20% to 80% (20% each step), including also 1% and 5% to account for strict latency targets. The number of tuples in each window increases proportionally. By this parameter setting, we measure the throughput and latency of Query group 2 (G2), as it is the one that is consisting of costly stateful aggregations. Throughput is calculated as the median (i.e., 50th percentile) of ten runs. Fig. 7 shows that SAOS slightly outperforms SpSS-based SRS. As shown in the secondary axis to the right of the figure, we rationale this to the fact that, on average, SpSS-based SRS needs to manage more geohash key states (for stateful aggregations, trans-trigger boundaries) than those keys that need to be managed by SAOS. Both, SpSS-based SRS and SAOS show similar trends for arrival rate of 2000K tuples/second with SpSS-based SRS slightly underperforming, though not shown here for the lack of space.

VI. Summary and Future Research Frontiers

In this work, we have designed a novel system that targets spatial approximate processing in fast arriving data streaming and dynamic scenarios, such as smart cities and urban computing [15] . We specifically focus on Spark Structured Streaming (SpSS) as the first in its category which introduces a fully declarative API for interactive stream processing [6]. However, Spark in its native version does not offer off-the-shelf solutions for spatial approximate query processing(SAQP), such as spatial sampling. Aiming at closing this void, we have designed SpatialSPE and incorporated it within the layers of SpSS, which then ensures its interoperability with the Spark ecosystem (specifically in SQL mode), taking full advantage of the underlying optimizations provided by Spark SQL, while operating in a more specialized mode (geospatial) and ensuring, at the same time, that query plans are optimal. To address the design premises presumed in § II, we are currently working on designing a controller that first predicts the fluctuation in the data arrival rate and skewness, thereby utilizes those in calculating a proportionate sampling fraction, which ensures preventing the congestion of operators comprising the DAG graph of the stream processing pipeline, thus meeting the latency/throughput targets prescribed by the user through SLAs.

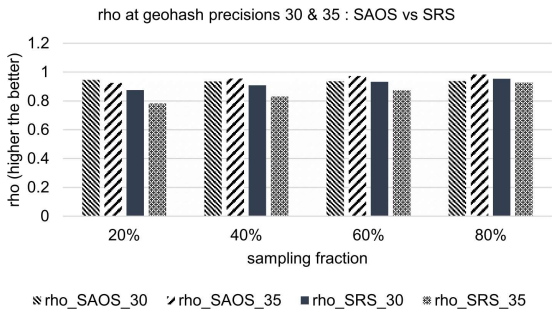

Fig. 6. rho by applying SAOS against SpSS SRS-based method

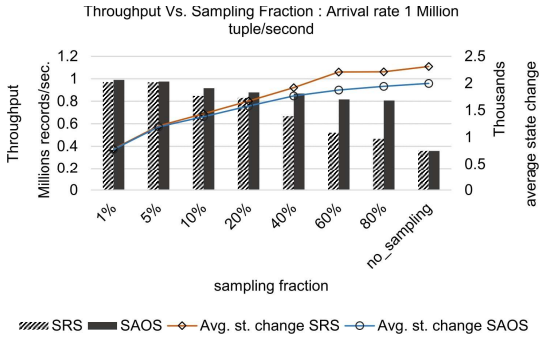

Fig. 7. Throughput comparison SAOS against SpSS-based SRS

Acknowledgment

This research was supported by the SACHER (Smart Architecture for Cultural Heritage in Emilia Romagna) project funded by the POR-FESR 2014-20 (no. J32I16000120009) through CIRI.

References

[1] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles,* 2013, pp. 423-438.

[2] K. Li and G. Li, "Approximate query processing: what is new and where to go?" *Data Science and Engineering,* vol. 3, *(4),* pp. 379-397,2018.

[3] A. J. Lister and C. T. Scott, "Use of space-filling curves to select sample locations in natural resource monitoring studies," *Environ. Monit. Assess.,* vol. 149, *(1-4),* pp. 71-80, 2009.

[4] D. L. Stevens Jr and A. R. Olsen, "Spatially balanced sampling of natural resources," *Journal of the American Statistical Association,* vol. 99, *(465),* pp. 262-278, 2004.

[5] A. Grafström, N. L. Lundström and L. Schelin, "Spatially balanced sampling through the pivotal method," *Biometrics,* vol. 68, *(2),* pp. 514-520, 2012.

[6] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica and M. Zaharia, "Structured streaming: A declarative API for real-time applications in apache spark," in *Proceedings of the 2018 International Conference on Management of Data,* 2018, pp. 601-613.

[7] I. M. Al Jawarneh, P. Bellavista, F. Casimiro, A. Corradi and L. Foschini, "Cost-effective strategies for provisioning NoSQL storage services in support for industry 4.0," in *2018 IEEE Symposium on Computers and Communications (ISCC),* 2018, pp. 1227.

[8] L. Wang, R. Christensen, F. Li and K. Yi, "Spatial online sampling and aggregation," *Proceedings of the VLDB Endowment,* vol. 9, (3), pp. 84-95, 2015.

[9] S. K. Thompson, "Spatial sampling," *Precision Agriculture: Spatial and Temporal Variability of Environmental Quality, (210),* pp. 161, 1997.

[10] S. L. Lohr, *Sampling: Design and Analysis*. Nelson Education, 2009.

[11] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun ACM,* vol. 51, *(1),* pp. 107-113, 2008.

[12] I. M. Al Jawarneh, P. Bellavista, A. Corradi, L. Foschini, R. Montanari and A. Zanotti, "In-memory spatial-aware framework for processing proximity-alike queries in big spatial data," in *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD),* 2018, pp. 1-6.

[13] C. H. Fleming and J. M. Calabrese, "A new kernel density estimator for accurate home-range and species-range area estimation," *Methods in Ecology and Evolution,* vol. 8, *(5),* pp. 571-579, 2017.

[14] A. Lehman, N. O'Rourke, L. Hatcher and E. Stepanski, *JMP for Basic Univariate and Multivariate Statistics: Methods for Researchers and Social Scientists*. Sas Institute, 2013.

[15] Y. Zheng, "Urban computing: Tackling urban challenges using big data," in *2016 IEEE 24th International Requirements Engineering Conference (RE),* 2016, pp. 3.