A Synopses Data Engine for Interactive Extreme-Scale Analytics

Antonis Kontaxakis
Athena Research Center
Technical University of Crete
akontaxakis@athenarc.gr
akontaxakis@softnet.tuc.gr

Nikos Giatrakos
Athena Research Center
Technical University of Crete
ngiatrakos@athenarc.gr
ngiatrakos@softnet.tuc.gr

Antonios Deligiannakis
Athena Research Center
Technical University of Crete
adeli@athenarc.gr
adeli@softnet.tuc.gr

ABSTRACT

We detail the novel architecture of a Synopses Data Engine (SDE) which combines the virtues of parallel processing and stream summarization towards interactive analytics at scale. Our SDE, built on top of Apache Flink, has a unique design that supports a very wide variety of synopses, allows for dynamically adding new functionality to it at runtime, and introduces a synopsis-as-a-service paradigm to enable various types of scalability.

CCS CONCEPTS

• **Information systems** → **Stream management**; • **Theory of computation** → **Parallel algorithms**; • **Computer systems organization** → *Distributed architectures.*

KEYWORDS

data summarization; data streams; Big Data analytics

ACM Reference Format:
Antonis Kontaxakis, Nikos Giatrakos, and Antonios Deligiannakis. 2020. A Synopses Data Engine for Interactive Extreme-Scale Analytics. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3412154

1 INTRODUCTION

Interactive extreme-scale analytics over massive, high speed data streams are essential in a wide variety of modern applications. For instance, in the financial domain, NYSE alone generates several terrabytes of data a day, including trades of thousands of stocks [4]. Stakeholders such as authorities and investors need to analyze these data in an interactive, online fashion for timely market surveillance or investment risk/opportunity identification purposes.

To enable interactive analytics at extreme-scale, stream processing platforms and systems need to provide three types of scalability:
• Horizontal scalability, i.e., scaling out (parallelizing) the computation to a number of machines and processing units in a computer cluster or cloud, to handle massive data volumes and arrival rates.
• Vertical scalability, i.e., the ability to scale the computation with the number of processed streams. For instance, the problem of identifying the highly correlated pairs of stock data streams under various statistical measures, such as Pearson's correlation, over N distinct data streams requires tracking a $\Theta(N^2)$ correlation matrix.
• Federated scalability, i.e., to scale the computation in settings where data arrive at multiple, geographically dispersed sites.

Big Data platforms, including Apache Flink [1] and Spark [3] among others, support stream processing focusing on horizontal scalability, but they are not sufficient by themselves to provide vertical and federated scalability. On the other hand, there is a wide consensus in stream processing [9, 10, 12] that approximate but rapid answers to analytics tasks, more often than not, suffice. Data summaries such as samples, sketches or histograms [9] preserve data properties important for providing approximate answers, with tunable accuracy guarantees, to a wide range of analytic queries. Such queries include, but are not limited to, cardinality, frequency moment, correlation, set membership or quantile estimation [9].

In this work, we detail the design and structure of a Synopses Data Engine (SDE) built on top of Apache Flink [1]. Our SDE combines the virtues of parallel processing and stream summarization towards delivering interactive analytics at extreme scale by enabling enhanced horizontal, vertical and federated scalability. It achieves horizontal scalability by parallelizing the computation of data synopses to a number of processing units and by enabling interactive analytics over such compact data summaries. It exploits properties of data synopses to achieve vertical scalability in ways that are not possible otherwise. Indicatively, the coefficients of Discrete Fourier Transform(DFT)-based synopses [12] have been used for correlation-aware hashing of streams to respective processing units. Highly uncorrelated streams are hashed to different processing units and comparisons are pruned for streams that are not hashed nearby. Finally, federated scalability is ensured by the fact that its design allows for data processing and query answering over distributed sites, exploiting the mergeability property [8] of many synopses techniques. What is more, our design supports a large number of commonly used synopses and implements a Synopsis-as-a-Service (termed SDEaaS) paradigm. Our novel SDEaaS approach allows one constantly running SDE job to: (i) accept on-the-fly requests for maintaining new synopsis, (ii) dynamically enhance its functionality by plugging-in external, new synopsis definitions customizing the SDE to application field needs at runtime, with zero downtime and (iii) reusing each available synopsis within multiple, concurrent application workflows, instead of duplicating respective streams and synopses for each. To our knowledge, our approach is the first to support such a large number of data synopses, to serve all discussed types of scalability and combine such SDEaaS facilities. Therefore, the design of the few prior efforts in the area,

This work has received funding from the EU Horizon 2020 research and innovation program INFORE under grant agreement No 825070.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412154

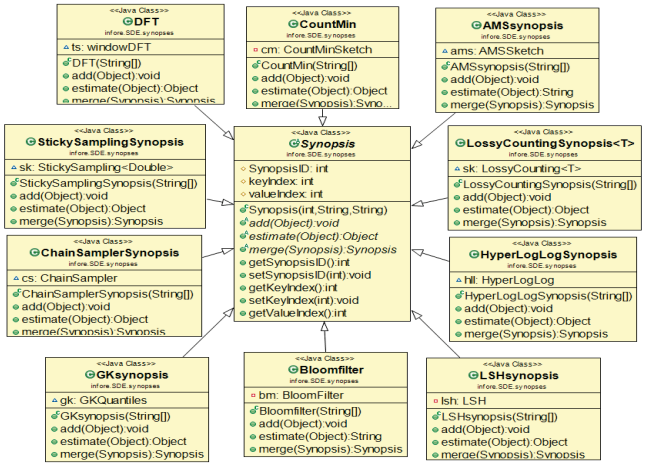


Figure 1: Structure of the Synopses Library (partial view).

Synopsis	Estimation	Parameters
CountMin	Count/Frequency Estimation	ϵ,δ
BloomFliter	Set Membership	#elements, False Positive Rate
FM Sketch	Distinct Count	Bitmap size, ϵ, δ
HyperLogLog	Distinct Count	Relative Standard Error
AMS Sketch	L_2-Norm, Inner Product	ϵ, δ
Discrete Fourier Transform (DFT)	Correlation, BucketID	Similarity Threshold, Number of Coefficients
Random Hyperplane Projection (RHP)	Correlation, BucketID	Bitmap Size, Similarity Threshold Number of Buckets
Lossy Counting	Count, Frequent Items	ϵ
Sticky Sampling	Count, Frequent Items	support, ϵ, δ
Chain Sampler	Sample	Sample Size
GKQuantiles	Quantiles	ϵ
CoreSetTree	CoreSets	Bucket size, dimensionality

Table 1: Supported synopses [9, 10, 12]. ϵ is the approximation error bound achieved with $1-\delta$ probability. For synopses over a window, window parameters are added.

such as [3, 5–7, 11] (more details in [13]), is not comparable with our novel architecture and SDEaaS approach, since these efforts do not deal with all types of required scalability and they need to run a separate cluster job, reserving virtual machines (VM) instead of new VM tasks, as we do, for new synopses.

2 EXTENSIBLE SDE LIBRARY AND API

A synopsis is typically either (a) a **single-stream synopsis**, i.e., a synopsis (e.g. sample) maintained on the trades of a single stock, or (b) a **data source synopsis**, i.e., maintained on all trades irrespectively of the stock. Our SDE supports both types of synopses.

The structure of the extensible synopses library is illustrated in Figure 1. Table 1 provides a full list of currently supported synopses, their parameters and estimated quantities. The development of the SDE Library exploits subtype polymorphism in Java to ensure the desired level of pluggability for new synopses definitions.

The higher level class `Synopsis` contains as member variables a unique identifier and 2 strings. The first string holds the details of the request with respect to how the synopsis should be physically implemented, i.e., index of the key field in an incoming data tuple (for single stream synopsis), the respective index of the value field which the summary is built on, whether the synopsis is a federated one and, in this case, which cluster should synthesize the overall estimation, and so on. The second string holds information regarding synopsis parameters, cited in Table 1. The `Synopsis` class also includes methods for `add`, `estimate` and `merge`. These are the typical operations involved in updating a synopsis with a new data tuple (add), in providing an estimate to a query (estimate), or in merging different synopses i.e., in distributed or federated scenarios.

Every synopsis algorithm is implemented in a separate class, as shown in Figure 1, that extends `Synopsis` and overrides the `add`, `estimate` and `merge` methods with its own algorithmic details [9]. New synopses can be easily added, **even dynamically at runtime**, through the use of inheritance and polymorphism to our SDE Library. The functionality our SDEaaS API provides is:

`Build/Stop Synopsis`. A synopsis can be created or ceased on-the-fly, as the SDE is up and running. In that, the execution of other running workflows that utilize synopsis operators, is not hindered.

`Dynamically Added Functionality`. Because shutting down our continuously running SDEaaS every time we want to upgrade its functionality (add a new supported synopsis) is not desired, our SDEaaS supports the pluggability of the code of additional (not included in the SDE Library) synopses, their dynamic loading and maintenance at runtime. The structure of the SDE Library, utilizing inheritance and polymorphism, is key for this task.

`Ad-hoc Query`. The SDE accepts one-shot, ad-hoc queries on a certain synopsis and provides estimations, based on its current status.

`Continuous Querying`. In this case, estimations, such as counts or correlations, are provided every time a synopsis is updated, for instance, due to the reception of a new tuple.

Estimations across streams (e.g., on joins) can often be computed by synopses on each stream [9, 10].

3 SDE ARCHITECTURE

Our architecture is built on top of Apache Flink [1] and Kafka [2]. Kafka is used as a fast, scalable and fault-tolerant messaging system enabling connectivity between the SDE and upstream, downstream operators in the workflows served by our SDE. Kafka together with the employed JSON format of requests and responses allows SDEaaS interoperability with upstream or downstream operators run on different Big Data platforms. The architecture engages a `Map` operator which produces a transformed tuple for every tuple it receives, a `FlatMap` operator which produces zero, one, or more tuples for each tuple it receives, while a `CoFlatMap` operator hosts two `FlatMap` that have access to common variables among streams. Finally, a `Union` operator receives two or more streams and creates a new one containing all their tuples, while a `Split` operator splits the stream into two or more streams given some criteria.

Employed Parallelization Scheme(s). The parallelization scheme that is employed in the design of the SDE is partition-based parallelization. That is, every data tuple that streams in the SDE architecture and is destined to be included in a maintained synopsis, does so based on the partition key it is assigned to it. **When a synopsis is maintained for a particular stream** (i.e., per stock) the key that is assigned to the respective update (newly arrived data tuple) is the identifier of that particular stream for which the synopsis is maintained. In this case, within Flink, that stream is processed

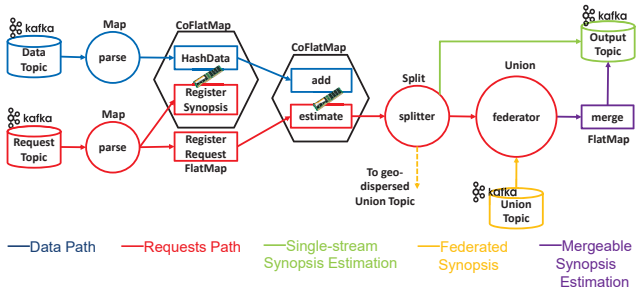


Figure 2: SDE Architecture – Condensed View.

by a task of the same worker and parallelization is achieved by distributing the number of streams for which a synopsis is built, to the available workers in the cluster hosting the SDE. On the other hand, **when a synopsis involves a data source** (i.e., financial data source for all monitored stock streams), the desired degree of parallelism is included as a parameter in the respective request to build/start maintaining the synopsis. In the latter case, one dataset is partitioned to the available workers in a round-robin fashion and the respective keys are created by the SDE (details on that follow shortly) each of which points (is hashed) to a particular worker.

3.1 SDE Architectural Components

Data and request (JSON snippet) streams arrive at a particular Kafka topic each. In the case of the `DataTopic` of Figure 2, a parser component is used in order to extract the key and value field(s) on which a currently running synopsis is maintained. The respective parser of the `RequestTopic` topic of Figure 2 reads the JSON snippet of the request and processes it. When an incoming request involves the maintenance of a new synopsis, the parser component extracts information about the parameters of the synopsis (see Table 1) and its nature, i.e. whether it is on a single stream, on a data source, or a synopsis that is also maintained in SDE instances in other geo-dispersed clusters. In case the request is an ad-hoc query, the parser component extracts the corresponding synopsis identifier(s).

Requesting New Synopsis Maintenance (red path). When a request is issued for maintaining a new synopsis, it follows the red-colored paths of the SDE architecture in Figure 2. The corresponding parser sends the request to a `FlatMap` operator (termed `RegisterRequest` at the bottom of Figure 2) and to another `FlatMap` operator (`RegisterSynopsis`) which is part of a `CoFlatMap` one. `RegisterRequest` and `RegisterSynopsis` produce the keys (as analyzed in the description of the supported parallelization schemes) for the maintained synopsis, but provide different functionality. The `RegisterRequest` operator uses these keys in order to later decide which worker(s) an ah-hoc query, which also follows the red-colored path, as explained shortly, should reach. The operator `RegisterSynopsis` uses the same keys to decide to which worker(s) a data tuple destined to update one or more synopses, which follows the blue-colored path in Figure 2, should be directed to.

Updating the Synopsis (blue path). When a data tuple destined to update one or more synopses is ingested via the `DataTopic` of Kafka, it follows the blue-colored path of the SDE architecture in Figure 2. The tuple is directed to the `HashData` `FlatMap` of the corresponding `CoFlatMap` where the keys (stream identifier for single stream synopsis and/or worker identifier for data source synopsis) are looked up based on what `RegisterSynopsis` has created. Following the blue-colored path, the tuple is directed to a `add` `FlatMap` operator which is part of another `CoFlatMap`. The `add` operator updates the maintained synopsis as prescribed by the algorithm of the corresponding technique. For instance, in case a FM sketch [9] is maintained, the `add` operation hashes the incoming tuple to a position of the maintained bitmap and turns the corresponding bit to 1 if it is not already set.

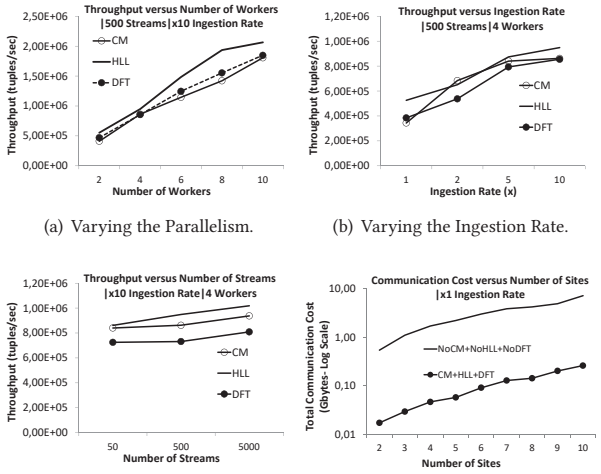
Ad-hoc Query Answering (red path). An ad-hoc query, via the `RequestTopic` of Kafka, is directed to the `RegisterRequest` operator. The operator which has produced the keys using the same code as `RegisterSynopsis` does, looks up the key(s) of the queried synopsis and directs the corresponding request to the `estimate` `FlatMap` operator of the respective `CoFlatMap`. The `estimate` operator reads, via the shared state, the current status of the maintained synopsis and extracts the estimation of the corresponding quantity the synopsis is destined to provide (Table 1).

Continuous Query Answering. In case continuous queries are to be executed on the maintained synopses, a new estimation needs to be provided every time the estimation of the synopsis is updated, either via an `add` operation or because a window on the data expires. In this particular occasion `estimate` needs to be invoked by `add`.

Both in ad-hoc and continuous querying, the result of `estimate`, following the red path in Figure 2, is directed to a `Split` operator, termed `splitter`. If necessary, the `splitter` forwards estimations to a `Union` operator, termed `federator` which reads from a `Union` Kafka topic (yellow path in Figure 2). The `Union` Kafka topic and the `federator` involve our provisions for maintaining federated synopses, i.e., synopses that are kept at a number of potentially geo-dispersed clusters. The `splitter` distinguishes between three cases. **Case 1** – `estimate` involves a single-stream synopsis maintained only locally at a cluster. Then, `Split` directs the output to downstream operators of the executed workflow via Kafka, by following the green-colored path in Figure 2. **Case 2**– a federated synopsis is queried but the request has identified another cluster responsible for extracting the overall estimation. Then, `Split` acts as the producer (writes) to the geo-dispersed `Union` Kafka topic of another cluster (declared by the dotted, yellow arrow coming out of `splitter` in Figure 2). **Case 3** – For non-federated synopses defined on entire data sources (e.g., a sample over all stock data), a number of workers of the current cluster participate in the employed parallelization scheme. Each such worker provides its local synopsis/estimation. Because something similar holds when individual clusters maintain federated synopses and the current cluster is set as responsible for synthesizing the overall estimation, in both cases the output of the `Split` operator is directed via `Union` to a `merge` `FlatMap` following the purple-colored path. The `merge` operator merges [8] the partial results of the various workers and/or clusters and streams the final estimation to downstream operators, again via an `Output` Kafka topic.

4 EXPERIMENTAL EVALUATION

We utilize a Kafka cluster with 4 Dell PowerEdge R320 Intel Xeon E5-2430 v2 2.50GHz machines with 32GB RAM each. Our Flink



(a) Varying the Parallelism. (b) Varying the Ingestion Rate.

(c) Varying the Number of Streams. (d) Varying the Number of Sites.

Figure 3: SDEaaS Scalability Study.

cluster has 10 Dell PowerEdge R300 Quad Core Xeon X3323 2.5GHz machines with 8GB RAM each. We use a real dataset of 5000 stocks with a total of ~10 TB of stock trade data provided by Spring Techno GmbH & Co. KG partner in the http://infore-project.eu/ project. The full version of this paper [13] expands on experimental findings.

We test the performance of our SDEaaS approach, measuring throughput, expressed as the number of tuples being processed per time unit (second) and communication cost (Gbytes) among workers, while varying a number of parameters involving horizontal ((i),(ii)), vertical (iii) and federated (iv) scalability, respectively: (i) the parallelization degree [2-4-6-8-10], (ii) the update ingestion rate [1-2-5-10] times the Kafka ingestion rate (i.e., each tuple read from Kafka is cloned [1-2-5-10] times in memory to further increase the tuples to process), (iii) the number of summarized stocks (streams) [50-500-5000] and (iv) the Gbytes communicated among workers for maintaining each examined synopsis as a federated one. This also represents the communication cost that would incur among equivalent number of sites (computer clusters), instead of workers, each of which maintains its own synopses. We build and maintain Discrete Fourier Transform (DFT–8 coefficients, 0.9 threshold), HyperLogLog (HLL – 64 bits, $m = 3$), CountMin (CM – $\epsilon = 0.002, \delta = 0.01$) synopses which are destined to support analytics related to correlation, distinct count and frequency estimation, respectively (Table 1). We use a time window of 5 minutes. The above parameters were set after discussions with the data provider.

Figures 3(a) and 3(b) show that increasing either the number of Flink workers or the ingestion rate causes a proportional increase in achieved throughput. This is a key sign of horizontal scalability, i.e., the data rates the SDEaaS can serve, are equivalent to the increasing rates at which data arrive to it. Figure 3(c) shows something similar as the throughput increases upon increasing the number of processed streams from 50 to 5000. This validates our claim regarding the vertical scalability aspects the SDEaaS can bring in the workflows it participates. Finally, Figure 3(d) illustrates the communication performance of SDEaaS upon maintaining federated

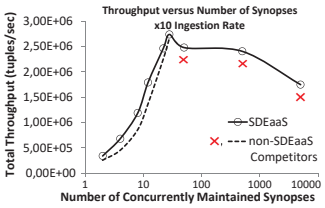


Figure 4: SDEaaS vs non-SDEaaS. ✘ denote that non-SDEaaS cannot maintain more than 40 synopses simultaneously.

synopses and communicating the results to a responsible site so as to derive the final estimations (see yellow arrows in Figure 2 and Section 3.1). For this experiment, we divide the streams among workers and each worker represents a site which analyzes its own stocks by computing CM, HLL, DFT synopses. A random site is set responsible for merging partial, local summaries and for providing the overall estimation. Note that the sites do not communicate all the time, but upon an `Ad-hoc Query` request every 5 minutes.

The important factor to judge federated scalability is the communication when we use the synopses, "CM+HLL+DFT" line in Figure 3(d), compared to when we do not, "NoCM+NoHLL+NoDFT" line in Figure 3(d). As Figure 3(d) illustrates, using synopses decreases communication cost by more than an order of magnitude.

SDEaaS vs non-SDEaaS. We now provide a comparison to a non-SDEaaS approach, such as Proteus [5], that extends Flink with data summarization utilities, but lacks the SDEaaS paradigm. We design an experiment where we start with 2 CM sketches (Table 1) for frequency estimations on the volume, price pairs of stocks. Then, we express demands for maintaining more CM sketches for up to 5000 stocks, without stopping the already running synopses. In Figure 4, we measure the sum of throughputs of all running jobs for the non-SDEaaS approach and the throughput of our SDEaaS. non-SDEaaS cannot maintain more than 40 synopses simultaneously, since it assigns entire task slots of VMs to new synopses and the 40 task slots in our cluster are depleted (see ✘ signs). SDEaaS has no such limit since it assigns new tasks for new synopses. SDEaaS also exhibits higher throughput than non-SDEaaS for up to 40 synopses.

REFERENCES

[1] [n.d.]. Apache Flink v. 1.9. https://flink.apache.org/.
[2] [n.d.]. Apache Kafka v. 2.3. https://kafka.apache.org/.
[3] [n.d.]. Apache Spark v. 2.4.4. https://spark.apache.org/.
[4] [n.d.]. Forbes. https://www.forbes.com/sites/tomgroenfeldt/2013/02/14/at-nyse-the-data-deluge-overwhelms-traditional-databases/#362df2415aab.
[5] [n.d.]. Proteus Project. https://github.com/proteus-h2020.
[6] [n.d.]. Stream-lib. https://github.com/addthis/stream-lib.
[7] [n.d.]. Yahoo DataSketch. https://datasketches.github.io/.
[8] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi. 2012. Mergeable Summaries. In *PODS*.
[9] G. Cormode, M. Garofalakis, P. Haas, and C. Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases* 4, 1-3 (2012), 1–294.
[10] M. Garofalakis, J. Gehrke, and R. Rastogi. 2016. Data Stream Management: A Brave New World. In *Data Stream Management - Processing High-Speed Data Streams*. Springer.
[11] B. Mozafari. 2019. SnappyData. In *Encyclopedia of Big Data Technologies*.
[12] Y. Zhu and D. E. Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *VLDB*.
[13] A. Kontaxakis, N. Giatrakos, and A. Deligiannakis. 2020, https://arxiv.org/abs/2003.09541. A Synopses Data Engine for Interactive Extreme-Scale Analytics.