2017 IEEE 33rd International Conference on Data Engineering

Efficient Exploration of Telco Big Data with Compression and Decaying

Constantinos Costa, Georgios Chatzimilioudis
*Dept. of Computer Science
University of Cyprus
1678 Nicosia, Cyprus
{costa.c, gchatzim, dzeina}@cs.ucy.ac.cy

Demetrios Zeinalipour-Yazti‡*
‡Max Planck Institute for Informatics
Saarland Informatics Campus
66123 Saarbrücken, Germany
dzeinali@mpi-inf.mpg.de

Mohamed F. Mokbel
Dept. of Computer Sc. & Engr.
University of Minnesota
Minneapolis, MN 55455, USA
mokbel@cs.umn.edu

***Abstract*—In the realm of smart cities, telecommunication companies (telcos) are expected to play a protagonistic role as these can capture a variety of natural phenomena on an ongoing basis, e.g., traffic in a city, mobility patterns for emergency response or city planning. The key challenges for telcos in this era is to ingest in the most compact manner huge amounts of network logs, perform big data exploration and analytics on the generated data within a tolerable elapsed time. This paper introduces *SPATE*, an innovative telco big data exploration framework whose objectives are two-fold: (i) minimizing the storage space needed to incrementally retain data over time; and (ii) minimizing the response time for spatiotemporal data exploration queries over recent data. The storage layer of our framework uses lossless data *compression* to ingest recent streams of telco big data in the most compact manner retaining full resolution for data exploration tasks. The indexing layer of our system then takes care of the progressive loss of detail in information, coined *decaying*, as data ages with time. The exploration layer provides visual means to explore the generated spatio-temporal information space. We measure the efficiency of the proposed framework using a 5GB anonymized real telco network trace and a variety of telco-specific tasks, such as OLAP and OLTP querying, privacy-aware data sharing, multivariate statistics, clustering and regression. We show that out framework can achieve comparable response times to the state-of-the-art using an order of magnitude less storage space.**

I. Introduction

Unprecedented amounts and variety of spatiotemporal *big* data are generated every few minutes by the infrastructure of a *telecommunication company (telco)*. The rapid expansion of broadband mobile networks, the pervasiveness of smartphones, and the introduction of dedicated Narrow Band connections for smart devices and Internet of Things (NB-IoT) [1] have contributed to this explosion. An early example of the data volume and velocity of telco big data is described in [2], where a telco collects 5TBs per day, i.e., almost 2PBs per year, in a single city of 10M cell phone customers.

Data exploration queries over big telco data are of great interest to both the telco operators and the smart city enablers (e.g., municipalities, public services, startups, authorities, and companies), as these allow for interactive analysis at various granularities, narrowing it down for a variety of tasks including: network plan optimization and user experience evaluation, precise marketing, emergency response, urban planning and new urban services [2], [3], [4], [5], [6]. Data exploration and visualization might be the most important tools in the big data era [7], [8], [9], where decision support makers, ranging from CEOs to front-line support engineers, aim to draw valuable insights and conclusions visually.

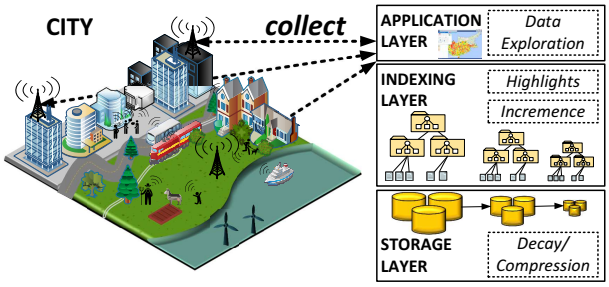

Fig. 1. *SPATE* is an efficient telco big data exploration framework that deploys compression, decaying and exploration of the collected data.

One key challenge in this new era of telco big data is to *minimize the storage costs* associated with the data exploration tasks, as big data traces and computed indexes can have a tremendous storage and I/O footprint on the data centers of telcos. Although the volume of electronically stored data doubles every year, storage capacity costs decline only at a rate of less than 1/5 per year [10]. Storing big data locally, due to the sensitive nature of data that cannot reside on public cloud storage, adds great challenges and costs that reach beyond the simplistic capacity cost calculated per GB [11]. From a telco's perspective, the requirement is to: (i) *incrementally store big data in the most compact manner*, and (ii) *improve the response time for data exploration queries*. These two objectives are naturally conflicting, as conjectured in [12].

In this paper we present *SPATE*[1], a SPAtio-TEmporal framework that uses both lossless data *compression* and lossy data *decaying* to ingest large quantities of telco big data in the most compact manner. *Compression* refers to the encoding of data using fewer bits than the original representation and is important as it shifts the resource bottlenecks from storage- and network-I/O to CPU, whose cycles are increasing at a much faster pace [13], [14], [15]. It also enables data explo-

[1]SPATE: "*a large number of things that appear or happen in a short period of time" (Merriam-Webster dictionary)*

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.175

IEEE computer society

ration tasks to retain full resolution over the most important collected data. *Decaying* on the other hand, as suggested in [16], refers to the progressive loss of detail in information as data ages with time until it has completely disappeared (the schema of the database does not decay [17]). This enables data exploration tasks to retain high-level data exploration capabilities for predefined aggregates over long time windows, without consuming enormous amounts of storage.

SPATE can be regarded as a domain-specific streaming data warehouse, which is divided into the following layers: (i) the *Storage layer*, which passes newly arrived network streams (coined *snapshots*), arriving with a periodic clock, through a lossless compression process that stores the results on a replicated big data file system for availability and performance; (ii) the *Indexing layer*, which provides the structures for efficient data exploration but also invokes the decaying process. Particularly, it is responsible for storing the upcoming snapshots on disk with the incremence module, for identifying interesting event summaries with the highlights module and for decaying the oldest leaf nodes of the index, i.e., by pruning-off parts of the tree index using a provided decaying function (i.e., *data fungus*). *Indexing* is the standard mechanism to speed up queries in incremental spatio-temporal querying and visualization systems [9], [5], [2], [6]; and (iii) the *Application Layer*, which holds components responsible for processing user queries and presenting results through a spatio-temporal visual user interface and a declarative SQL user interface.

We evaluate the performance of the *SPATE* framework using a 5GB anonymized real telco big data trace, whose structure is explained in the next section. To show the utility of *SPATE*, we carry out a variety of telco-specific querying tasks, such as OLAP and OLTP querying, clustering, regression and privacy sanitization. Our results indicate that *SPATE* requires sub-linear storage space with respect to the amount of data ingested, an update time of only a few seconds without affecting the online data, and a data exploration response time that is independent of the queried temporal window. *SPATE* achieves similar query response times to state-of-the-art solutions [9], but using only a fraction of the data storage space.

There is no prior work that studies *data decay* and efficient *data exploration* for telco big data in combination. In previous work, custom data management systems have been designed with the objectives to save storage space using compression, and speed up temporal range queries using indices [18], [19], [20], [21]. None of these considers the notion of "decay" as expressed in [16], which suggests sacrificing either accuracy or read efficiency for less frequently accessed data to save space. Furthermore, these solutions are tailored specifically for managing scientific (floating point) data.

In contrast, we focus on compressing and decaying incremental telco big data, which mostly contains string and integer values, on spatiotemporal data exploration queries in a telco setting, and also develop our solution on top of off-the-shelf open source systems (e.g., Hadoop, Spark) that are widely used in industry, with low installation, administration and maintenance costs.

Our contributions can be summarized as follows:

- We apply efficient compression algorithms in order to reduce the storage space and minimally affect query response time to exploratory search queries in a telco big data setting.
- We propose a multi-resolution spatio-temporal index that supports the notion of data decaying.
- We provide an extensive experimental evaluation using a variety of telco-specific tasks to show the benefits of our approach.

II. Preliminaries

In this section we describe the special characteristics of telco big data and the telco network that produces them.

A. The Anatomy of a Telco Network

A telco network consists of the Radio and the Core Network, as shown in Figure 2. A Radio Network operates in three different modes, namely GSM/GPRS, UMTS and LTE. The Global System for Mobile Communications (GSM) is the standard developed for voice communications in a digital circuit-switched cellular network (2G). To allow for data communication, the GSM standard expanded over time to become a packet-switched network via General Packet Radio Service (GPRS). The Universal Mobile Telecommunications System (UMTS) extends GPRS to deal with increased data needs and constitutes the third generation (3G) mobile cellular system that moves toward an all-IP network. Finally, the Long-Term Evolution (LTE) standard was developed to increase the capacity and speed using a different radio interface together with Core Network improvements. It is sometimes called the fourth generation (4G) mobile cellular system. Overall, all three modes interleave one another to offer the best possible coverage to the users. In the future, fifth generation (5G) networks are expected to improve the radio coverage, capacity, data rate and latency with technologies like femtocells, millimeter waves, massive MIMO, beamforming and full duplex [22], i.e., mainly advancing the Radio Network.

The GSM operation is supported by base stations called Base Transceiver Stations (BTS), which are controlled by Base Station Controllers (BSC). Each BTS is equipped with transceivers for transmitting and receiving radio signals, antennas and components for encrypting and decrypting the messages using the BSC. The BSC controls hundreds of BTSs and is responsible for the allocation of radio channels, receives measurements from the mobile phones, and controls handovers from BTS to BTS. It ultimately combines the multiple low-capacity connections of its BTSs into combined "virtual" connections that are sent over to the Mobile Switching Center (MSC) in the Core Network. Finally, it provides data to the *Operation Support Subsystem (OSS)*, whose data will be described extensively in the next subsection.

The UMTS operation is supported by base stations, called Node B, which are controlled by Radio Network Controllers (RNC). The RNC provides similar functions to that of BSC,

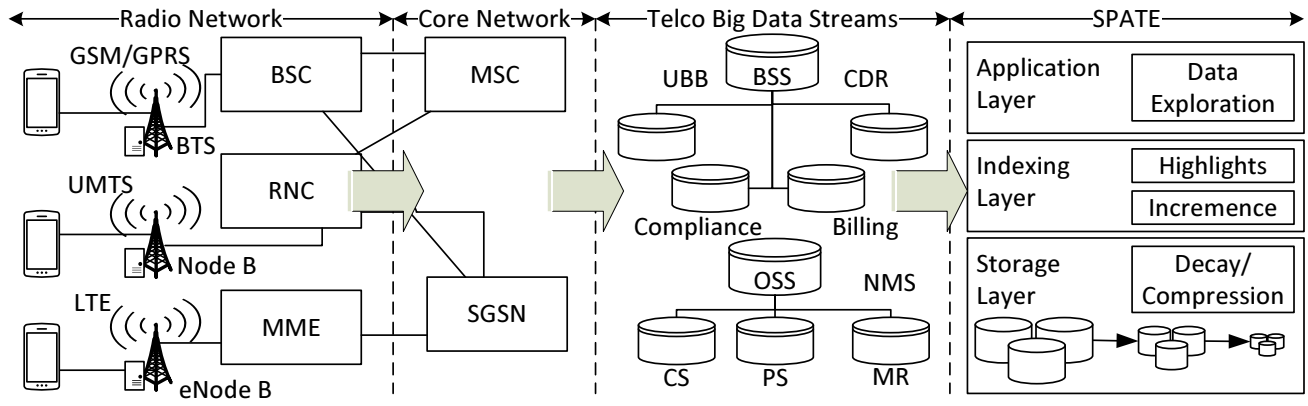

Fig. 2. The anatomy of a typical telco network architecture that generates telco big data streams consumed by *SPATE*.

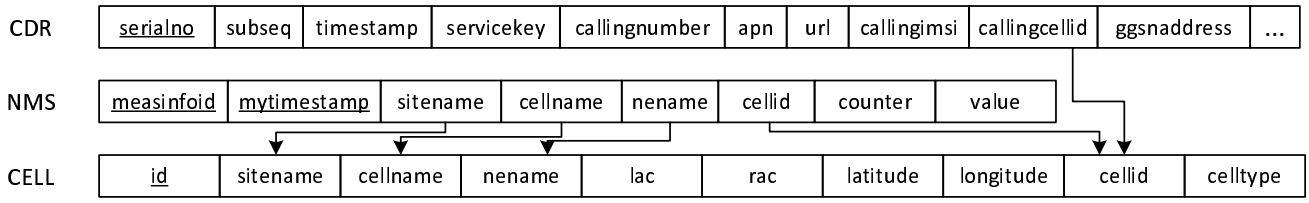

Fig. 3. The relational schema of the telco's data shows the first 10 out of ∼200 attributes of the CDR data and all the attributes of the NMS and CELL data.

only for the UMTS network. RNC connects to the circuit-switched Core Network through the Serving GPRS Support Node (SGSN). The RNC also provides data to the OSS. The LTE operation is supported by base stations, called eNode B, which can directly connect to the core network. The Mobility Management Entity (MME) authenticates the wireless devices connected to eNode Bs and is involved in hand-offs between LTE and previous standards.

The data generated by the network can be considered as *data streams*, which are aggregated continuously on the telco's data center for operational purposes but also replicated to an exploration and visualization system like SPATE, for efficient indexing, querying, visual exploration and analysis.

B. The Structure of Telco Big Data

Typical *telco big data* streams [4] consist of: (i) *Business Supporting Systems (BSS)* data, which are used to run the telco business operations towards customers (e.g., orders, payment issues, revenues). BSS are associated with the following specific types of data and stored in conventional SQL databases: User Base Behavior (UBB) records, compliance records, billing records, as well as voice/message *Call Detailed Records (CDR). BSS data has only a limited volume of around 24GB per day (for a 2M+ clientele of a China telco) [4]* and were widespread within telco operational and analytical IT infrastructure even before the big data era; and (ii) *Operation Supporting Systems (OSS)* data, which is generated by the telco's computer systems used to manage its networks and stored in NoSQL big data stores. OSS comprises of data in the following parts: Circuit Switch (CS), Packet Switch (PS) - often referred as *Mobile BroadBand (MBB)* data - and Measurement Report (MR). CS data describe call connection quality, PS data describe users' web behavior (e.g., web speed, connection success rate) and MR includes a variety of measurement reports (e.g., for estimating user location [23]). By analyzing OSS one can observe important call connection quality statistics and user experience indicators. As a representative source of OSS, we use *Network Measurement System (NMS)* reports that contain counters for call drop rates, call duration measurements, antenna throughput. *OSS data has a volume of around 2.2TB per day, occupying over 97% data volume of the entire dataset [4].*

A CDR contains only metadata with fields that describe a specific telecommunication session (i.e., transaction), but does not contain the actual content of that transaction. For example, a CDR describing a phone call may contain the phone numbers of the calling and receiving party, the start time, end time, call type, and duration of that call. CDR is the primitive data source for customer billing purposes. On the other hand, the OSS logs contain aggregated performance counters for the three different network types. In general, the data is highly structured and comprises of relational records based on a predetermined schema with a large number (~ 200) of attributes that take mostly nominal text and interval-scaled discrete numerical values (see Figure 3).

We analyzed a real anonymized dataset in an effort to understand what compression ratios can be achieved. The time duration of the dataset was 1 week and it consisted of about 1.7M CDR and 21M NMS records coming from approximately 300K users. The total size of the dataset was 5GB. Based on

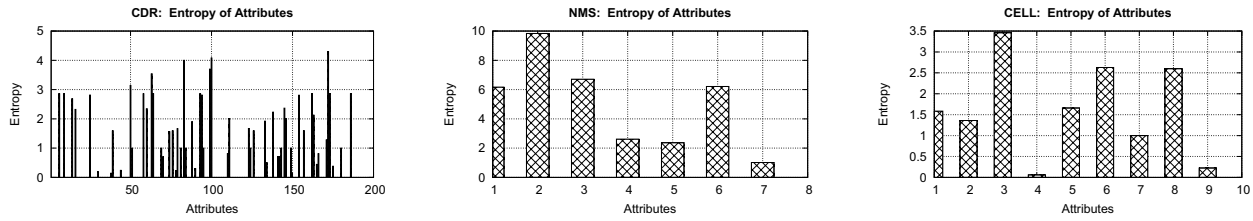

Fig. 4. The entropy of each attribute in (Left) CDR data, (Center) NMS data, and (Right) CELL data.

Shannon's source coding theorem [24], the minimum number of bits needed to express a symbol given a set of possible symbols S and their probabilities P is $-\sum_{p_i \in P} \log_2 p_i$, and the maximum compression ratio possible is inversely proportional to the entropy $H = -\sum_{p_i \in P} p_i \log_2 p_i$ of the data. In Figure 4, we plot the entropy of the attributes included in three different file types that arrive at a telco data center. Looking at the first plot that corresponds to CDR data files, it immediately stands out that most attributes have an entropy smaller than 1 and some even have an entropy of 0. This confirms that high compression ratios can be achieved. The attributes that have 0 entropy are optional attributes that usually are left blank.

The data arrives at the data center in batches, called henceforth *snapshot* data noted by d_i, in the form of horizontally segmented files every 30 minutes, a period we call *ingestion cycle* or *epoch*. A snapshot d_i contains records of user activity (e.g., phone call completion) or network activity that took place at some timepoint within the corresponding ingestion cycle, and records that express aggregate values over the same ingestion cycle (e.g., call drops, throughput). Each snapshot d_i can be seen as a table of records (rows) with a predefined set of attributes (columns).

Regarding the spatial information inherent within the telco network data, every record is linked to a specific cell ID c. Each cell ID corresponds to a cell that covers a specific area a_c and is attached to a base station that has a known location. Therefore, we can not talk about spatial data in the traditional sense, as each record is only associated with a specific geographical cellular area a_c that usually spans hundreds of meters and estimating user location, as in [23], is outside the scope of this work.

III. SPATE: Overview

In this section we provide an overview of the *SPATE* architecture, which consists of three layers (see Figure 1): the storage, the indexing and the application layer. We start out with a problem formulation and then outline our solution.

A. Problem Formulation

Given a telco setting, where telco big data arrives periodically in batches, we want to: (i) minimize the space needed to store/archive data; and at the same time (ii) minimize response time for spatiotemporal data exploration queries and tasks.

Given storage space S needed for storing data before any compression is performed, storage space S_c needed for the data after compression and storage space S_i needed for storing the access method information (e.g., index), we can measure the ratio of contribution towards the first objective as $O_1 = S/S'$, where $S' = S_c + S_i$. Similarly, given the query response time T over the uncompressed data and the query response time T_{ci} over compressed and indexed data, we can measure the ratio of contribution towards the second objective of this work as $O_2 = T/T_{ci}$.

B. Our Solution

We express our solution in three layers overviewed next and described in detail in the following sections:

Storage Layer: implements the compression logic in *SPATE*. The *Storage layer* passes newly arrived network snapshots through a lossless compression process storing the results on a replicated big data file system for availability and performance. This component is responsible for minimizing the required storage space with minimal overhead on the query response time. The intuition is to use compression techniques that yield high compression ratios but at the same time guarantee small decompression times. We particularly use GZIP compression that offers high compression/decompression speeds, with a high compression ratio and has maximum compatibility with I/O stream libraries in the big data ecosystem we use. The storage layer is basically only responsible for the leaf pages of the *SPATE* index described in the next layer.

Indexing Layer: is responsible for minimizing the query response time for data exploration queries. It maintains and uses a multi-resolution spatiotemporal index that consists of the *Incremence* module and the *Highlights* module. The Incremence module receives the newly arriving snapshot d_i and incorporates it into the index by incrementing it on the right-most path. The Highlights module combines data from the stored snapshots to create efficient representations of interesting events, called "highlights". These highlights are constructed for each day, month and year and the end of each such period, respectively. The highlights of a month are constructed from the daily highlights, and the highlights of a year are constructed from the monthly highlights. Finally, this layer is also responsible for the gradual decay of the data. It does so by pruning-off parts of the tree index by using the notion of data fungus we will explain.

Application Layer: implements the querying module and the user interface. In our case, we present the *Data Exploration* module, which receives a data exploration query $Q(a, b, w)$

and based on a, b, and w uses the index to combine the needed highlights to answer the query. Finally, *SPATE* is equipped with an easy-to-use map-based web interface that hides the complexity of the system and accesses all *SPATE* functionality. Details of the web interface are described in Section VI.

IV. SPATE: Storage (Compression) Layer

This section describes the lowest layer of the *SPATE* framework that relates to storage. The *SPATE* storage layer takes care of compressing snapshots of telco streams as they arrive periodically. In our setting, exploratory queries need to be able to perform exact queries over recent data, therefore our compression mechanisms have to be *lossless*. In the following subsections we provide basic compression terminology and the desiderata of our approach. We then provide a qualitative description of various lossless compression libraries that are compared against in a microbenchmark that follows.

A. Terminology and Desiderata

We start out with some basic terminology and then provide our objectives. Given a lossless compression codec c and a dataset d that occupies space S, the codec can compress d into S_c space in *compression time* T_{c1}. The *compression ratio*, which quantifies the reduction in data size produced by a data compression algorithm, is defined by $r_c = S/S_c$ and depends on both c and d. Finally, c can decompress d back to its original state in *decompression time* T_{c2}.

The first objective of *SPATE* relates to saving space. Particularly, the compression mechanism needs to achieve a high *compression ratio*. On the other hand, the query response time needs to be kept low (second objective), therefore the compression mechanism needs to have a small *decompression time*, since this overhead will be paid for every query. It is important to notice that our approach is not particularly concerned with the *compression time*, as the compression cost is only paid for a single snapshot in each round. One final argument of concern is compatibility with existing stream readers. For example, GZIP [25] is widely supported by various environments and its usage provides maximum portability.

B. Lossless Compression Libraries

In this section we overview some traditional and some emerging big data lossless compression *libraries*, which can be linked directly to existing big data processing software like *SPATE*. These libraries also have respective standalone command-line tools.

- **GZIP [25]:** is a traditional file format and a software library used for file compression and decompression. It is based on the DEFLATE algorithm that uses a combination of Lempel-Ziv coding (LZ77) [26] and Huffman coding. In LZ77, repeated occurrences (in a look-ahead buffer) are replaced with pointers to a recently encoded sequence (sliding window buffer). In this sense, it is a sequential data compression technique with a dictionary that is constructed during the encoding process. On the contrary, in Huffman coding, a one-to-one symbol-to-code map is constructed based on occurrence probabilities of symbols in a learning corpus (i.e., entropy-based). The DEFLATE algorithm, defined through RFC1951, is then a hybrid algorithm that consists of a series of blocks encoded in either LZ77 or Huffman and preceded by a respective header.
- **7z [27]:** is another traditional dictionary-based compression tool and library based on the LZMA and LZMA2 algorithms. Like GZIP, it aims to build a good statistical model or "dictionary" for the input data upon which bit sequences of frequently encountered data can be compacted more densely.
- **SNAPPY [28]:** is a modern open-source compression and decompression library by Google that aims for maximum compression speed as opposed to maximum compression ratios. In SNAPPY, the compressed files are reported to be 20% to 100% bigger than other compression tools but compression and decompression speeds are reported to be faster due to an optimized implementation. The library has been used extensively by Google in its BigTable, MapReduce and internal RPC systems.
- **ZSTD [29]:** is another modern open-source lossless compression/decompression library developed by Facebook, which targets real-time compression scenarios. ZSTD (aka Zstandard) uses new generation entropy coders HUFF0 (Huffman) and FSE (Fine State Entropy), which are designed to perform well on modern CPUs and belong to the Asymmetric Numeral Systems (ANS) family of entropy algorithms. Compared to prior tools, ZSTD allows building domain-specific training dictionaries.

TABLE I
Lossless Compression with Different Libraries in SPATE
(Average results per 30-min snapshot)

Objectives \ Libraries	GZIP	7z	SNAPPY	ZSTD
Compress. Ratio (r_c)	9.06	11.75	4.94	9.72
Compress. Time (T_{c1}) in sec	21.37	20.99	21.39	21.07
Decompress. Time (T_{c2}) in sec	0.11	0.12	0.13	0.11

C. Microbenchmark

Our objective in this subsection is to empirically assess the presented compression libraries as part of the *SPATE* storage layer, which stores snapshots in a directory hierarchy. Our dataset includes 200 snapshots from the 5GB anonymized and uncompressed telco dataset that comprises of 1.7M CDR and 21M NMS records. Our microbenchmark is performed on top of an HDFS v2.5.2 filesystem (more details about the testbed are provided in Section VII). We particularly focus on the three common metrics: *compression ratio* r_c, *compression time* T_{c1} and *decompression time* T_{c2}.

Table I shows the results of our evaluation. Our first observation is that r_c is similarly satisfactory for GZIP, 7z and ZSTD. On the other hand, SNAPPY shows that its r_c

is only half as good as the rest libraries so it might not be a good alternative for *SPATE*. The T_{c1} and T_{c2} results relate to compression and decompression time, respectively, for a single snapshot and are measured in seconds. Looking at these numbers we clearly observe that T_{c1} takes always more time than T_{c2}, which is very typical for compression algorithms. Looking at the costs more carefully, we observe that the T_{c1}/T_{c2} ratio is about 200 instead of the more typical 2. This is attributed to the fact that *SPATE* performs many additional CPU-bound functions in each compression round, such as parsing. Finally, SNAPPY does not expose any speed benefits overall, as the slow I/O is hiding its benefits.

As a conclusion, we denote that the *SPATE* storage layer can operate with a variety of libraries, each of them coming with different performance trade-offs. In our implementation and evaluation, we chose the GZIP library, which was readily available from within the `java.util.zip` core libraries and was also supported readily by certain parts of the application layer described in Section VI.

V. SPATE: Indexing (Decaying) Layer

The *Indexing layer* provides the structures for efficient data exploration but also invokes the decaying process. Particularly, it is responsible for augmenting the upcoming snapshots on disk with the incremence module, identifying interesting event summaries with the highlights module and decaying the oldest leaf nodes of the index, i.e., by pruning-off parts of the tree index using the so called data fungus. In the remainder of this section, we outline the three modules of the *SPATE* indexing layer.

A. Incremence Module

This module is responsible for the incremental construction of a multi-resolution spatio-temporal index as data snapshots are ingested by *SPATE*. Our index has 4 levels of temporal resolutions (i.e., epoch (30 minutes), day, month, year), with each leaf level containing 2 spatial dimensions (x,y) and N additional domain-specific dimensions (e.g., CDR and NMS).

Figure 5 shows an example index in SPATE. As we can see, the root node points to year-nodes, each representing a single year. Each year node points to 12 month-nodes, each representing a single month. Similarly, the month nodes point to their corresponding day-nodes, and each day node points to its corresponding 48 snapshot leaves.

Every time a new snapshot arrives, it is compressed by the storage layer and then the temporal index is incremented on its right-most path. If the new snapshot belongs to an incomplete day, it is just added as a leaf under the existing right-most day-node. Else, we first need to add a new dummy day-node. If this new day is the first day of a new month, we also need to add a new dummy month-node. Similarly, if the new month is the first month of a new year, we first need to add a new dummy year-node.

Each leaf node could store an additional spatial index (e.g., R-tree or quad-tree variant) to speed up data exploration queries within a snapshot. For example, a query like: *"find the aggregates regarding some object of interest in a given spatial bounding rectangle for a specific time range"* could benefit from such an embedded spatial index after reaching the leaf level of the index. Such an index would allow to quickly scan the attributes stored per snapshot. However, snapshots are usually not very large (i.e., have a 30 min timespan), thus an additional index would only provide modest additional query response time benefits at the price of additional storage space that we aim to minimize.

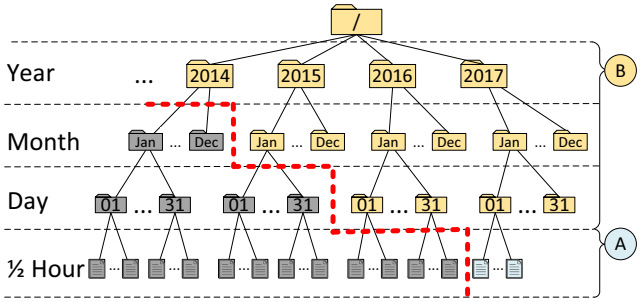

Fig. 5. The multi-resolution spatio-temporal index in *SPATE*. Our index has 4 levels of temporal resolutions with each leaf level containing 2 spatial dimensions (x,y) and N additional domain-specific dimensions (e.g., related to CDR and NMS). The red line denotes the decaying data fungus that evicts progressively the oldest leaf and non-leaf nodes of the tree.

B. Highlights Module

To enable interactive data exploration we compute "highlights" from the underlying raw data for each internal node of the temporal index structure. Such highlights are effectively materialized views to long-standing queries of users (e.g., the drop-call counters, bandwidth statistics), which are executed in a periodic manner as the snapshots stream to *SPATE*. In this sense, the highlights can be perceived as an OLAP cube whose construction cost is amortized over time. Building the highlights cube enables the application layer, we will describe next, to swiftly go over the generated statistics for visualization purposes. Consequently, users can drill down or roll up to the desired aggregates without additional delays.

Below we outline the operation of the highlights module: At the end of each day, when all the snapshots of that day have been added as leaves, the highlights of that day are calculated from the compressed data of its snapshots, and are stored in the day-node. They are also forwarded to the parent month-node, which increments its own monthly highlights. Similarly, at the end of each month/year, the highlights of that month/year are calculated from the highlights of its days/months and are stored in the month/year-node and forwarded to its parent node, which in-turn increments its own highlights. This way the root will store the highlights of all the completed years.

The computation is based on the frequency of occurrence of a value in the data. Frequent values with an occurrence frequency above threshold θ are treated as no-highlights, whereas values with an occurrence frequency below threshold θ are considered highlights. A highlight is described by its type (in case of categorical data) or its peaking point (in case

of continuous numerical values) and its duration. Its important to observe that for each level of resolution (day, month, year) a separate frequency threshold θ_i can be used, e.g., lower thresholds for higher levels resolution.

C. Decaying Module

The last module of the indexing layer deals with decaying of compressed snapshot data and aggregated highlights. *Decaying* refers to the progressive loss of detail in information as data ages with time until it has completely disappeared. Kersten refers to the existence of data fungus in [16], e.g., *"Evict Grouped Individuals"*, which helps in the decaying processing. In our work, we chose a data fungus we coin *"Evict Oldest Individuals"* as it helps us to deal more pragmatically with telco network signals, where more recent signals contain more important operational value that needs to be retained fully.

We particularly devise a scheme where operators chose the rate at which the temporal decaying policy becomes effective. The red line in Figure 5, denotes one hypothetical such policy that aims to retain up to one year of data exploration with full resolution along with yearly progressive decay. This policy is translated into a continuous decaying process where leaf and non-leaf entries of the spatio-temporal index are purged from replicated storage in a sliding window manner.

The result of the decaying process is that the data exploration warehouse can retain the highest possible data exploration resolution for predefined aggregate queries over extremely long time windows without consuming enormous amounts of storage. Otherwise, the storage overheads would soon enforce administrators to delete large quantities of telco big data traces, purging at the same time the hope to learn valuable insights from big data at the macroscopic scale.

VI. SPATE: Application Layer

In this section we present the Application layer that holds components responsible for receiving user queries and presenting the results. These are implemented in the query exploration interfaces, according to a query evaluation process we outline.

A. Query Evaluation and Processing

This module receives a data exploration query and accesses the index maintained by the Indexing layer of *SPATE*.

In our setup, a data exploration query $Q(a, b, w)$ consists of an attribute selection a, a spatial bounding box b, and a temporal window of interest w. A b can cover an area a few hundreds of square meters up to several hundreds of square kilometers. Similarly, a temporal window can span a few hours to several months or even years. A query $Q(a, b, w)$ can be expressed as *"Explore the values of a within the spatial box b and temporal window w"*. Such queries have direct applicability to interactive visualization tools used for data exploration that present data overlaying a geographical map. The users zoom into the map (thus defining the b) and select the attributes (a) and the time period (w), for which they would like to observe the query results as snapshots or as a video (i.e., "playback highlights in fast-forward").

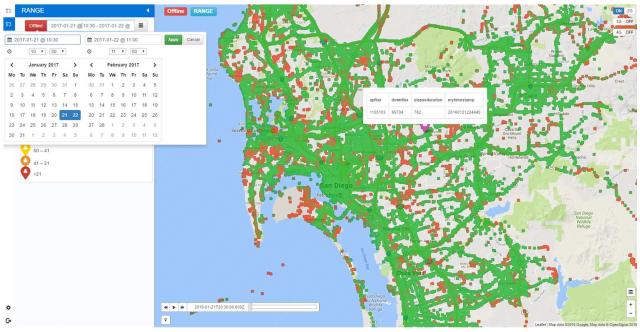

Fig. 6. SPATE-UI: A spatio-temporal telco data exploration user interface we developed on top of Google Maps, which enables combining network models (e.g., coverage heatmaps) with real network measurements (e.g., CDR, NMS, CELL) encapsulated in the compressed *SPATE* structure.

Given a data exploration query $Q(a, b, w)$ the index is accessed to find the temporal node whose period completely covers w. For example, if the temporal window is from 15 September 2016 until 15 October 2016, the index is accessed up to the year level and the highlights of year-node 2016 are retrieved. Once the correct temporal node is found, all highlight summaries or actual available data whose spatial bounds completely cover b are then retrieved.

SPATE might retrieve records for a larger period than the one requested. However, this is not a problem given that users very often like to have a quick glance to the period before and after the chosen window. In this case, our decision to retrieve a larger period serves as an implicit prefetching mechanism. When users decide to focus on a smaller window within w, it is considered as a data exploration query $Q(a, b, w')$ with $|w'| < |w|$, which can be served directly from the cache of the user interface that is explained in the next subsection.

B. Query Exploration Interfaces

We have realized two separate interfaces for the *SPATE* framework: (i) *SPATE-UI*, which is a visual spatio-temporal data exploration interface developed on top of Google Maps; and (ii) *SPATE-SQL*, which is a declarative data exploration interface in Apache Hue (Hadoop User Experience).

The **SPATE-UI** interface allows the user to interactively navigate in space and time (see Figure 6). Particularly, the user can set a temporal and spatial predicate and observe the behavior of vital network statistics and how these compare to precomputed network models. For instance, the network coverage is a precomputed heatmap model that is overlayed and can be visually compared against the real measurements that are loaded from storage to memory through the *SPATE* structure. The above effectively translates to the execution of a variety of spatial range queries on top of the *SPATE* structure. We will show in our experimental section that both range queries as well as other OLTP and OLAP queries retain desirable retrieval properties (response time) at enormous storage savings. The *SPATE-UI* finally also provides a search box that enables a user to narrow the spatial bounding box

based on well-known Points-of-Interests (POIs) organized by Google or some other provider (e.g., Openstreetmap). The *SPATE-UI* also contains a query bar that enables the execution of template queries for drop calls and downflux/upflux, heatmap statistics (e.g., showing the RSSi signal intensity around antennas), satellite/terrain map layers (e.g., for Cellular signal propagation faults due to the terrain) and others.

The **SPATE-SQL** interface allows expert users and data scientists to explore the collected data through declarative SQL. The current configuration currently allows all basic SELECT-FROM-WHERE block queries, nested queries, joins, aggregates, etc. directly through the compressed Hadoop Distributed File System (HDFS) representation of the *SPATE* structure, which aims to provide means for ad-hoc query execution with the same storage savings with SPATE-UI.

VII. Experimental Testbed and Methodology

To validate our proposed ideas and evaluate *SPATE*, we have implemented a trace-driven experimental testbed on which we conducted a comprehensive set of experiments. Particularly, we compare *SPATE* against two competing approaches for three different metrics and eight different usage scenarios.

A. Compared Frameworks

Our aim in this experimental evaluation is to compare the following three frameworks:

- **RAW:** This is the default solution that stores the telco snapshots as data files on the HDFS file system without any compression, indexing or decaying.
- **SHAHED:** This is a MapReduce-based system for querying and visualizing spatio-temporal satellite data proposed in [9]. *SHAHED* is appropriate for online querying and visualization, but does not deploy compression or decaying in its internal structures. To allow fair comparison, we isolated the spatio-temporal aggregate index of SHAHED, part of SpatialHadoop 2.4 [30], and executed it along with the other frameworks in Spark [31].
- **SPATE:** This is the framework proposed in this work. *SPATE* aims to minimize the usage of disk space by retaining fast spatio-temporal querying means.

B. Experimental Testbed

Our experimental testbed is implemented on top of a *Hadoop Distributed File System (HDFS)* [32] along with Apache Hive [32] for *online* querying and Apache Spark [31] for *offline* (i.e., data-intensive distributed in-memory) data processing. Our testbed stores data in either text format (for *RAW* and *SHAHED*) or compressed (for *SPATE*). We use an HDFS file system with 64MB block size and default replication 3. In order to streamline the trace-driven experimental evaluation process, we have formulated the evaluation tasks as individual Scala programs submitted directly to the Spark computation master. In this way, we managed to circumvent additional latencies and overheads introduced by the query exploration interfaces introduced in Section VI.

Our evaluation is carried out on the DMSL-UCY laboratory VCenter IaaS datacenter, a private cloud, which encompasses 5 IBM System x3550 M3 and HP Proliant DL 360 G7 rackables featuring single socket (8 cores) or dual socket (16 cores) Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, respectively. The datacenter is managed through a VMWare vCenter Server 5.1 that connects to the respective VMWare ESXi 5.0.0 hosts. The computing cluster, deployed over our VCenter IaaS, comprises of 4 Ubuntu 14.04 server images, each featuring 8GB of RAM with 2 virtual CPUs (@ 2.40GHz). The images utilize slow 7.2K RPM RAID-5 SAS 6 Gbps disks, available through a IBM storage system DS3512, formatted with VMFS 5.54 (1MB block size). Each node features the following frameworks, i.e., Hadoop v2.5.2, Spark 1.6.0 and Hive 2.0.

C. Datasets

We utilize an anonymized dataset of telco traces comprising of 1.7M call detail records (CDR), 21M network measurements records (NMS) and 3660 cells (CELL) coming from 1192 2G, 3G and LTE antennas distributed in an area of about 6000 km^2. The data traffic is created from about 300K users and has a total size of $\sim$5GB. To evaluate the ingestion and querying time of our propositions for various day periods, we have generated the following four large datasets based on the arrival time of the snapshots:

- **Morning Dataset:** was generated by extracting the data that have time arrival between 5 am to 12 pm (noon).
- **Afternoon Dataset:** was generated by extracting the data that have time arrival between 12 pm to 5 pm.
- **Evening Dataset:** was generated by extracting the data that have time arrival between 5 pm to 9 pm.
- **Night Dataset:** was generated by extracting the data that have time arrival between 9 pm to 5 am.

In order to better understand the behavior of our real dataset, we have additionally partitioned the dataset into seven zones at the granularity of week days (i.e., Monday to Sunday).

D. Metrics

We utilize three metrics, the first two targeting the storage and indexing process and the third one the data querying and exploration process, as follows:

- **Ingestion Time:** this measures the cost incurred for storing each arriving snapshot and incrementing the index. Given a compression library c and a data snapshot d, the ingestion time includes the compression time T_{c1} needed to compress d and the time needed to run the Incremence module that adds the data into our spatiotemporal index.
- **Space:** this measures the total space S' that data and index occupy throughout the whole distributed system in Megabytes (MB). Space S' is calculated after all incoming snapshots have been compressed and ingested.
- **Response Time:** this measures the response time for answering a query in seconds (sec). The time is being calculated from the query submission until the answer is calculated. This includes the accessing of the spatiotemporal index, the decompression of the retrieved data and

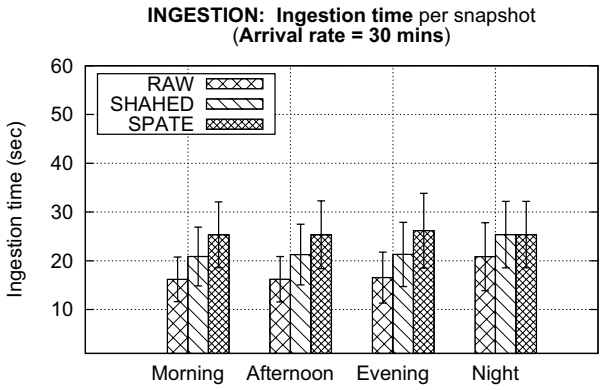

Fig. 7. **Ingestion time:** We compare *SPATE* against *RAW* and *SHAHED* on real data partitioned by day period.

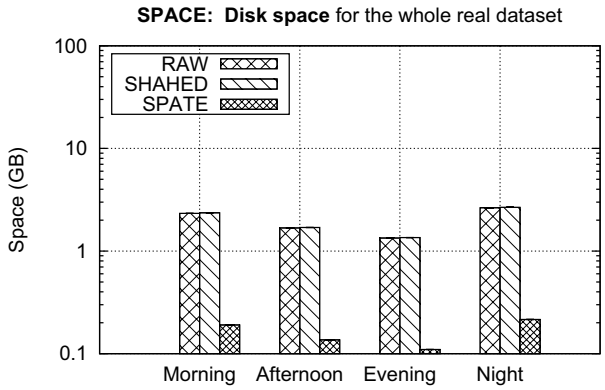

Fig. 8. **Disk space:** We compare *SPATE* against *RAW* and *SHAHED* on real data partitioned by day period.

the presentation of the results. Times are averaged over five iterations measured in seconds.

E. Data Exploration Tasks

Our experimental evaluation has been conducted based on an a diverse mix of OLTP, OLAP, privacy sanitization, statistics, data mining, and Machine Learning (ML) workloads. All aforementioned workloads are driven by a telco-specific domain task. We particularly formulated the following eight tasks segmented into two groups: (i) T1-T5 represent basic operational and analytical queries as well as privacy sanitization tasks that were executed without Spark parallelization; and (ii) T6-T8 represent heavier computational tasks that were executed with Spark parallelization.

T1. Equality: This task aims to retrieve the download and upload bytes for a requested snapshot, e.g., `SELECT upflux, downflux FROM CDR WHERE ts=''201601221530'';`

T2. Range: this aims to retrieve the download and upload bytes for a requested time window, e.g., `SELECT upflux, downflux FROM CDR WHERE ts>=''2015'' AND ts<=''2016'';`

T3. Aggregate: this aims to retrieve the NMS counters for the drop calls of each cell tower and calculate the drop call rate for each cluster of cells, e.g., `SELECT cellid, SUM(val) FROM NMS WHERE ... GROUP BY cellid;`

T4. Join: this query involves a self-join among two CDR tables. Particularly, it aims to identify the products that have changed their location (as identified by the cell towers).

T5. Privacy: This task retrieves and anonymizes the result set based on the k-anonymity model proposed in [33] through the ARX [34] Java library. Particularly, it generates a k-anonymized dataset by generalizing, substituting, inserting, and removing information as appropriate in order to make the quasi-identifiers indistinguishable among k rows.

T6. Statistics: This task aims to generate a variety of multivariate statistics using Spark's Machine Learning (ML) library (i.e., `Statistics.colStats(observations)`). The goal is to calculate the column-wise max, min, mean, variance, number of non-zeros and the total count.

T7. Clustering: This task aims to cluster a specific range of snapshots using the k-means algorithm in Spark based on the CDR and NMS data.

T8. Regression: This task estimates relationships among the attributes in our dataset using linear regression over a specific temporal window. Spark's Machine Learning (ML) library is used (i.e., `regression.LinearRegression`).

VIII. Experimental Evaluation

In this section we carry out an extensive experimental evaluation that aims to uncover the performance properties of our propositions.

A. Performance over varying day-periods

This experimental series studies the effect that the various day periods (i.e., morning, afternoon, evening, night) have on the ingestion time and disk space needed to index and store the incoming snapshots of both NMS and CDR data.

In Figure 7 we see that *SPATE* has the slowest ingestion time, although still comparable (i.e., a maximum of 1.25x slower). We also observe that the data load per snapshot affects the ingestion time only negligibly, assuming that the data load varies among day periods. Even though *SPATE* is somewhat slower during ingestion, its benefits are manifested in Figure 8, where we observe that *SPATE* requires an order of magnitude less disk space compared to the other techniques. This performance is also steady with respect to the varying data loads associated with each different day period.

We conclude that *SPATE* achieves a large improvement in expensive disk space requirements trading-off a negligible overhead on the ingestion time. Since each snapshot arrives only every 30 minutes (i.e., 1800 seconds) and the ingestion completes within two orders of magnitude less time, we can claim that the small delay in ingestion time is acceptable.

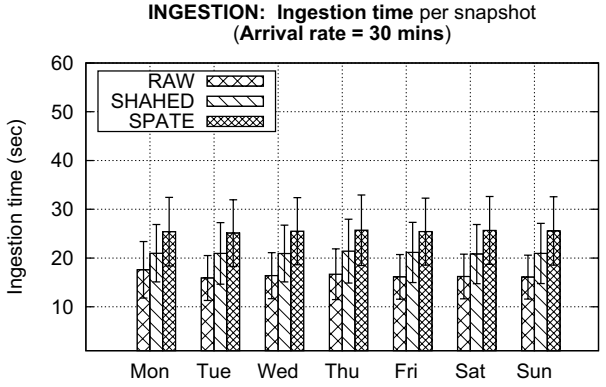

Fig. 9. **Ingestion time:** We compare *SPATE* against *RAW* and *SHAHED* on real data partitioned by day of week.

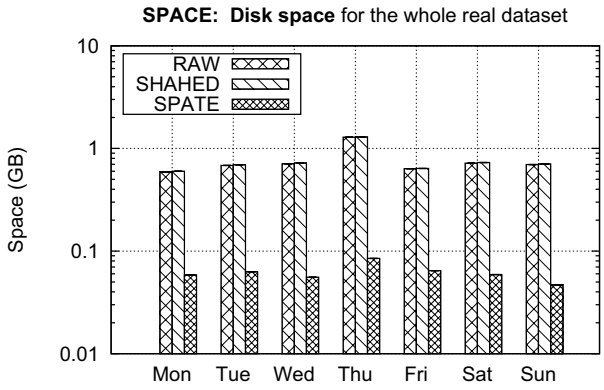

Fig. 10. **Disk space:** We compare *SPATE* against *RAW* and *SHAHED* on real data partitioned by day of week.

B. Performance over days of the week

This experimental series studies the effect that the various week days (i.e., Monday through Sunday) have on the ingestion time and disk space needed to index and store the incoming snapshots of both NMS and CDR data. In Figure 9, we see again that *SPATE* has the slowest ingestion time, although still comparable (i.e., a maximum of 1.2x slower this time). We also observe that the data load per snapshot affects the ingestion time only negligibly, assuming that the data load varies between days. In Figure 10, we observe that *SPATE* requires again an order of magnitude less disk space. This performance is also steady with respect to the varying data loads among week days. The conclusion here is similar, *SPATE* achieves a large improvement in disk space, trading-off a negligible overhead on the ingestion time.

C. Response time

This experimental series studies the response time each ingestion framework achieves for the tasks described in Section VII-E using the CDR dataset.

In Figure 11, we present the query response time for the simpler tasks T1-T5, all of which either involve a single scan or a nested loop on the data stored on HDFS. We observe that *SPATE* is only slightly slower than *SHAHED* for T1-T3 and T5 (i.e., from 0.1s to 3s), which is due to the fact that *SPATE* requires time to decompress the files stored on the HDFS before returning the results. On the other hand, for the join task T4, *SPATE* achieves a 4-5x speed up compared to *SHAHED*, mainly due to the fact that T4 involves a nested loop and that such a loop is much faster in *SPATE* where the HDFS input streams are already compressed in GZIP.

In Figure 12, we present the query response time for the heavier tasks T6-T8 in log-scale, which are executed with Spark parallelization on. Our first observation is that the query response time of all three tasks are now in the order of many thousand seconds, but *SPATE* remains close the running time of *SHAHED* in all cases. The second observation is that some tasks, such as the k-means computation in T7 and the Linear Regression in T8, the query response time benefits for *SPATE* are not so apparent. This happens as both T7 and T8 are CPU-bound rather than I/O-bound problems and as such, the existence of compressed input streams is not helping the query response time significantly. The significant benefit of *SPATE* over the other two frameworks for T7 and T8 are still of course that it requires significantly less storage (i.e., 10x reduction). For all eight tasks, *SPATE* requires the least storage space, i.e., 0.49GB vs. 5.37GB and 5.32GB required by *SHAHED* and *RAW*, respectively.

IX. Related Work

In this section we present existing research on telco big data and on compressing incremental archives. These works are not directly comparable to SPATE, but are presented for completeness.

A. Telco Big Data Research

Telco big data research falls in the following categories: (i) real-time analytics and detection; (ii) experience, behavior and retention analytics; and (iii) privacy. There is also traditional telco research not related to big data, rather comprises of topics related to business (BSS) data in relational databases.

Real-time Analytics and Detection: Zhang et al. [5] developed *OceanRT*, which was one of the first real-time telco big data analytic demonstrations. Yuan et al. [2] present *OceanST* which features: (i) an efficient loading mechanism of ever-growing telco MBB data; (ii) new spatiotemporal index structures to process exact and approximate spatiotemporal aggregate queries. Iyer et al. [6] present *CellIQ* to optimize queries such as *"spatiotemporal traffic hotspots"* and *"handoff sequences with performance problems"*. It represents the snapshots of cellular network data as graphs and leverages on the spatial and temporal locality of cellular network data. Zhu et al. [23] deal with the usage of telco MR data for city-scale localization, which is complementary to the scope of our work.

Braun et al. [35] develop a scalable distributed system that efficiently processes mixed workloads to answer event stream

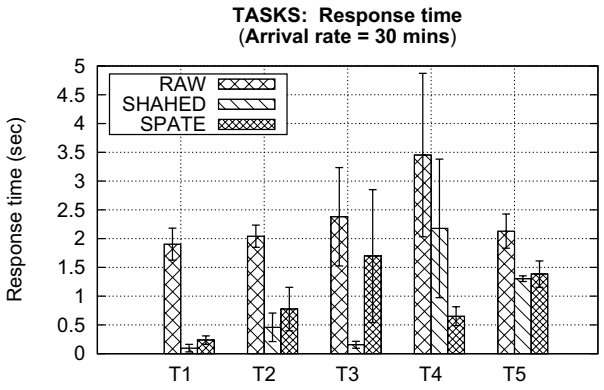

Fig. 11. **Response time for simpler tasks T1-T4:** We compare *SPATE* against *RAW* and *SHAHED* on the complete real dataset.

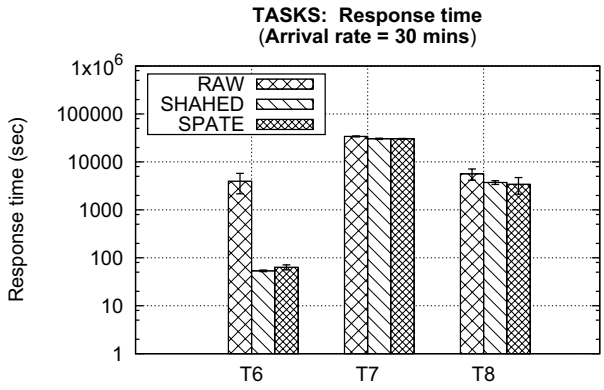

Fig. 12. **Response time for heavier tasks T5-T8:** We compare *SPATE* against *RAW* and *SHAHED* on the complete real dataset.

and analytic queries over telco data. Bouillet et al. [36] develop a system on top of IBM's InfoSphere Streams middleware that analyzes 6 billion CDR per day in real-time. Abbasoğlu et al. [37] present a system for maintaining call profiles of customers in a streaming setting by applying scalable distributed stream processing. All aforementioned efforts have a similar scope to SPATE, but don't incorporate concepts of compression or decaying of data.

Experience, Behavior and Retention Analytics: Huang et al. [4] empirically demonstrate that customer churn prediction performance can be significantly improved with telco big data. Although BSS data have been utilized in churn prediction very well in the past decade, the authors show how with a primitive Random Forest classifier telco big data can improve churn prediction accuracy from 68% to 95%. Luo et al. [38] propose a framework to predict user behavior involving more than one million telco users. They represent users as documents containing a collection of changing spatiotemporal "words" that express user behavior. By extracting the users' space-time access records from MBB data, they learn user-specific compact topic features that they use for user activity level prediction. Ho et. al. [39] propose a distributed community detection algorithm that aims to discover groups of users that share similar edge properties reflecting customer behavior. Iterative learning algorithms are not the primary target for SPATE, but can be supported at similar costs to RAW data (i.e., decompression occurs in first iteration).

Privacy: Hu et al. [40] study Differential Privacy for data mining applications over telco big data and show that for real-word industrial data mining systems the strong privacy guarantees given by differential privacy are traded with a 15% to 30% loss of accuracy. Privacy and confidentiality are critical for telcos' reliability due to the highly sensitive attributes of user data located in CDR, such as billing records, calling numbers, call duration, data sessions, and trajectory information. *SPATE* deals with privacy-aware data sharing as a functionality for next generation smart-city applications.

B. Compressing Incremental Archives

Domain-specific compression techniques have previously been proposed, e.g., for compressing spatiotemporal climate data [41], text document collections [42], scientific simulation floating point data [18], [19], [20], [21], and floating point data streams [43]. None of these prior works has been proposed for distributed systems and can not be directly applied to telco data, which mostly contains generic string and integer values.

Works [44], [45], [46] have studied differential compression techniques and the trade-off between compression ratio and decompression times for incremental archival data. Differential compression is a topic we will investigate more carefully in the future as it can reduce the storage layer overheads in each acquisition cycle.

X. Conclusions and Future Work

In this paper we proposed an innovative telco big data exploration stack, coined *SPATE*, whose objectives are two-fold: (i) minimizing the storage space needed to incrementally retain data *over time*; and (ii) minimizing the response time for spatiotemporal data exploration queries over recent data. We have measured the efficiency of our proposition using a real telco trace and a variety of telco-specific tasks, such as OLAP and OLTP querying, clustering, regression and privacy sanitizing, and showed that we can achieve comparable response times to the state-of-the-art with an order of magnitude less storage space. In the future, we aim to investigate a variety of advanced smart city application scenarios on top of *SPATE*, namely an automated car traffic mapping system and an emergency recovery system after natural disasters.

Acknowledgments

This work was supported in part by the University of Cyprus, an industrial sponsorship by MTN Cyprus and EU COST Action IC1304. The third author's research is supported by the Alexander von Humboldt-Foundation, Germany. The last author's research is supported by NSF grants IIS-0952977, IIS-1218168, IIS-1525953, CNS-1512877.

REFERENCES

[1] Ericsson.com, "Cellular Networks For Massive IoT enabling low power wide area applications," 2016. [Online]. Available: https://goo.gl/Sf2Cj4

[2] M. Yuan, K. Deng, J. Zeng, Y. Li, B. Ni, X. He, F. Wang, W. Dai, and Q. Yang, "Oceanst: A distributed analytic system for large-scale spatiotemporal mobile broadband data," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1561–1564, Aug. 2014.

[3] J. Reades, F. Calabrese, A. Sevtsuk, and C. Ratti, "Cellular census: Explorations in urban data collection," *IEEE Pervasive Computing*, vol. 6, no. 3, pp. 30–38, 2007.

[4] Y. Huang, F. Zhu, M. Yuan, K. Deng, Y. Li, B. Ni, W. Dai, Q. Yang, and J. Zeng, "Telco churn prediction with big data," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD. New York, NY, USA: ACM, 2015, pp. 607–618.

[5] S. Zhang, Y. Yang, W. Fan, L. Lan, and M. Yuan, "Oceanrt: Real-time analytics over large temporal data," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 1099–1102.

[6] A. P. Iyer, L. E. Li, and I. Stoica, "Celliq: Real-time cellular network analytics at scale," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 309–322.

[7] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact." *MIS quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012.

[8] TeraLab, "TeraLab Data Science for Europe," 2016. [Online]. Available: http://www.teralab-datascience.fr/

[9] A. Eldawy, M. F. Mokbel, S. Alharthi, A. Alzaidy, K. Tarek, and S. Ghani, "Shahed: A mapreduce-based system for querying and visualizing spatio-temporal satellite data," in *2015 IEEE 31st International Conference on Data Engineering*, April 2015, pp. 1585–1596.

[10] C. LaChapelle, "The cost of data storage and management: where is the it headed in 2016?" 2016. [Online]. Available: http://www.datacenterjournal.com/cost-data-storage-management-headed-2016/

[11] Z. Li, A. Mukker, and E. Zadok, "On the importance of evaluating storage systems' $costs," in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, Philadelphia, PA, 2014.

[12] M. Athanassoulis, M. S. Kester, L. M. Maas, R. Stoica, S. Idreos, A. Ailamaki, and M. Callaghan, "Designing access methods: The rum conjecture," in *Intl. Conf. on Ext. Database Technology (EDBT)*, 2016.

[13] Y. Chen, A. Ganapathi, and R. H. Katz, "To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency," in *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, ser. Green Networking '10, 2010, pp. 23–28.

[14] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. Ross, "Improving i/o forwarding throughput with data compression," in *2011 IEEE Intl. Conference on Cluster Computing*, Sept 2011, pp. 438–445.

[15] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, "Integrating online compression to accelerate large-scale data analytics applications," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 1205–1216.

[16] M. L. Kersten, "Big data space fungus," in *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[17] M. Stonebraker, R. Castro, F. Dong Deng, and M. Brodie, "Database decay and what to do about it." 2016. [Online]. Available: https://goo.gl/tJNa9m

[18] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *European Conference on Parallel Processing*. Springer, 2011, pp. 366–379.

[19] E. R. Schendel, Y. Jin, N. Shah, J. Chen, C.-S. Chang, S.-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross *et al.*, "Isobar preconditioner for effective and high-throughput lossless data compression," in *IEEE 28th Intl. Conference on Data Engineering*, 2012, pp. 138–149.

[20] J. Jenkins, I. Arkatkar, S. Lakshminarasimhan, D. A. Boyuka II, E. R. Schendel, N. Shah, S. Ethier, C.-S. Chang, J. Chen, H. Kolla *et al.*, "Alacrity: Analytics-driven lossless data compression for rapid in-situ indexing, storing, and querying," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems X*. Springer, 2013, pp. 95–114.

[21] E. Soroush and M. Balazinska, "Time travel in a scientific array database," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 98–109.

[22] A. Gupta and R. K. Jha, "A survey of 5g network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.

[23] F. Zhu, C. Luo, M. Yuan, Y. Zhu, Z. Zhang, T. Gu, K. Deng, W. Rao, and J. Zeng, "City-scale localization with telco big data," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM. New York, NY, USA: ACM, 2016, pp. 439–448.

[24] C. Shannon, "A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656," *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948.

[25] "Gzip." [Online]. Available: http://gzip.org/

[26] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theor.*, vol. 23, no. 3, pp. 337–343, Sep. 2006.

[27] "7z." [Online]. Available: http://7-zip.org/

[28] "Snappy." [Online]. Available: http://google.github.io/snappy/

[29] "Zstd." [Online]. Available: https://github.com/facebook/zstd

[30] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A MapReduce Framework for Spatial Data," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015, pp. 1352–1363.

[31] "Apache Spark." [Online]. Available: http://spark.apache.org/

[32] "Apache Hive." [Online]. Available: http://hadoop.apache.org/

[33] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002.

[34] "Arx data anonymization tool." [Online]. Available: http://arx.deidentifier.org/

[35] L. Braun, T. Etter, G. Gasparis, M. Kaufmann, D. Kossmann, D. Widmer, A. Avitzur, A. Iliopoulos, E. Levy, and N. Liang, "Analytics in motion: High performance event-processing and real-time analytics in the same database," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD. New York, NY, USA: ACM, 2015, pp. 251–264.

[36] E. Bouillet, R. Kothari, V. Kumar, L. Mignet, S. Nathan, A. Ranganathan, D. S. Turaga, O. Udrea, and O. Verscheure, "Processing 6 billion cdrs/day: From research to production (experience report)," in *Proceedings of the 6th ACM Intl. Conference on Distributed Event-Based Systems*, ser. DEBS, 2012, pp. 264–267.

[37] M. A. Abbasoğlu, B. Gedik, and H. Ferhatosmanoğlu, "Aggregate profile clustering for telco analytics," *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1234–1237, Aug. 2013.

[38] C. Luo, J. Zeng, M. Yuan, W. Dai, and Q. Yang, "Telco user activity level prediction with massive mobile broadband data," *ACM Trans. Intell. Syst. Technol.*, vol. 7, no. 4, pp. 63:1–63:30, May 2016.

[39] Q. Ho, W. Lin, E. Shaham, S. Krishnaswamy, T. A. Dang, J. Wang, I. C. Zhongyan, and A. She-Nash, "A distributed graph algorithm for discovering unique behavioral groups from large-scale telco data," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM. New York, NY, USA: ACM, 2016, pp. 1353–1362.

[40] X. Hu, M. Yuan, J. Yao, Y. Deng, L. Chen, Q. Yang, H. Guan, and J. Zeng, "Differential privacy in telco big data platform," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1692–1703, Aug. 2015.

[41] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, "Integrating online compression to accelerate large-scale data analytics applications," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 1205–1216.

[42] H. Yan, S. Ding, and T. Suel, "Inverted index compression and query processing with optimized document ordering," in *Proceedings of the 18th intl. conference on World wide web*. ACM, 2009, pp. 401–410.

[43] M. Burtscher and P. Ratanaworabhan, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, 2009.

[44] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblance detection." in *USENIX Annual Technical Conference, General Track*, 2003, pp. 113–126.

[45] L. L. You, K. T. Pollack, D. D. Long, and K. Gopinath, "Presidio: a framework for efficient archival data storage," *ACM Transactions on Storage (TOS)*, vol. 7, no. 2, p. 6, 2011.

[46] S. Bhattacherjee, A. Chavan, S. Huang, A. Deshpande, and A. Parameswaran, "Principles of dataset versioning: Exploring the recreation/storage tradeoff," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1346–1357, 2015.