SAMIZDATA: a scale-independent data management system for time series

Paolo Atzeni[1], Luigi Bellomarini[2], Eleonora Laurenza[3], and Marco Lippi[4]

[1] Università Roma Tre [2] Università Roma Tre and Bank of Italy
[3] Sapienza University of Rome and Bank of Italy
[4] Einaudi Institute for Economics and Finance

Abstract. Time series data are becoming more and more pervasive in many business and human endeavors. The challenge of developing dedicated systems for managing such data lies in handling the problem of the scale: time series are inherently growing and so the algorithms to compute answers to queries soon become ineffective. In this paper we describe Samizdata, a data management system we have developed for Einaudi Institute for Economics and Finance, a research institute controlled by the Bank of Italy. We leverage a recent systematization of the concept of scale-independent query answering from the database literature to deliver a scalable management system for time series.

1 Introduction

We are assisting to a big proliferation of data sources that produce ordered sequential measurements, each referred to a specific point in time. This kind of data is generally named *time series* [10]. Many important applications of time series data are in the financial context, where prominent examples are the sequences of trading ticks, that is, the upward and downward movements of the price of a security [8, 14]. Series play a fundamental role also in many other fields, where they represent a range of different events, such as: scalar quantities measured by sensors, machine logs, metrics from social network analysis, transport data collected from mobile devices, biological tracks, and so on [3, 9, 11, 13, 15].

The *Einaudi Institute for Economics and Finance* (EIEF) is a research institute, controlled by the Bank of Italy, which produces world-class research in economics and finance. Together with the Bank of Italy, it has been playing a key role in the standardization and automation of processes concerning the production of official statistics. Their business processes are centered on time series: as a consumer, EIEF exploits third-party time series for research needs, importing data from other systems; as a producer, EIEF compiles and delivers new time series as the final deliverable research products.

The increasing availability of new time series resulted in many new research opportunities for the institute that could be exploited only with a new dedicated management system able to guarantee stable performances with rapidly growing volumes. Besides, the increased amount of data produced by the various sources

also put the current data management system under stress, affecting the ability to respond to contingent needs. This paper is based on our experience at EIEF in the development of SAMIZDATA, a data management system for time series.

Time series data have special characteristics: they tend to be *immutable*, *rapidly growing* and substantially *schema-less*. Immutable because they represent a sequence of events, each meaningful at a point in time. Thus, keeping track of all of them is essential: more recent values do not update older ones, but are queued and memorized. Since values are never deleted, time series also grow rapidly. They are usually fed by high-frequency sources, reaching, for instance, hundreds of thousands of ticks per second (like for stock quotes). Finally, in this context, the database schema is barely present and indeed very simple: all the series in a system are only characterized by their values and the respective timestamps. Due to these characteristics, in the development of systems for time series, the challenge is providing solutions "at scale", that is, the techniques adopted for certain amounts of data must remain effective when the volumes grow.

Our experience at EIEF showed that a data management system for time series has to support their whole lifecycle through a range of high-level operations, which enable the various phases of the business processes. Examples are: the application of a research model, the compilation of a new time series calculated from existing ones, the visualization, the export into a known format, and so on. All these operations rely on the basic task of finding and retrieving the time series, which are often searched by some characterizing features, such as their name in natural language, their description, their frequency, the unit of measure, the original producer of the data, the copyright notes, and so on. For example, let us suppose we memorize our time series in a database $\mathcal{D}$, in particular in a relation `TS(id, name, description, frequency, ..., ts_time, ts_data)`, where `id` is a progressive key, `name` is the name of the series, `description`, `frequency` and `currency` are some of its features, `ts_data` stands for the values and `ts_time` is the timestamp each value refers to. The following expression is an example of *feature-based* query: $\texttt{name} \wedge \texttt{ts\_time} \wedge \texttt{ts\_data} \wedge \texttt{description} = \texttt{"industrial production"}$. It asks for the `name` and all the time-value pairs of the series about "industrial production".

The problem we address in SAMIZDATA is guaranteeing the *scale independence* of such feature-based queries: independently of how many time-value pairs are currently stored, we want the system to respond with stable performances. More precisely, we want it to respect upper bounds for both the execution time (the query must not take too long) and the size of the result set (the result must be manageable, hence not too large).

The database literature has recently proposed the notion of *access schema* [7], a theoretical device that formally defines the classes of queries that can be executed in a scale-independent way on a database $\mathcal{D}$. In particular, an access schema is composed of a set of entries $a_i = (R_i, \{f_1, \ldots, f_n\}, N, T)$. Each of them specifies that for a relation R_i of $\mathcal{D}$, a query over attributes $f_1, \ldots, f_n$, will be answered by the system in at most T units of time, producing no more than N tuples. Hence, to express the scale independency of the query of our example,

we would have the following entry: $a_i = (\texttt{TS}, \{\texttt{description}\}, N, T)$. We believe that the access schema is a precious tool to define and validate the scalability requirements of a data management system.

In SAMIZDATA we use the notion of access schema, concretely in an industrial product, taking the chance of EIEF present needs to deliver a software useful for an emerging category of business scenarios. In particular, SAMIZDATA achieves scale independence on feature-based queries by using access schemas to drive and validate the design of the system. First, we characterize the class of queries that cover the user requirements and then formulate an access schema that expresses the scale independence requirements for such queries. In order to satisfy the proposed access schema, we argue that relational solutions have some limitations, basically due to the impossibility to contain the growth of the number of tuples and guarantee stable query performances when new time-value pairs are added. Thus, we propose a non-relational data model for our scenario, where a time series is represented with documents (i.e. a recursive key-value data structure) and its time-value pairs as nested sub-documents. In our data model, new time-value pairs do not increase the number of first-level documents and do not affect the scalability. For the feature-based queries, we propose an inverted index, also modeled as a document, which for each possible feature value points to the most meaningful time series. We build a set of basic primitives on top of the feature-based queries and use them to support a number of satellite components in the system, which offer all the functionalities needed to support time series lifecycle in EIEF business processes.

The remainder of the paper is organized as follows. In Section 2, we present the scenario, giving context and motivations to our solution. In Section 3, we introduce the problem of scale-independent query answering, put it in the context of time series data, and present how we approached the problem in SAMIZDATA. Then Section 4 describes the overall architecture of the system. In Section 5 we examine the related work and, in Section 6, we draw up our conclusions.

2 The scenario

The *Einaudi Institute for Economics and Finance* has relevant responsibilities in the production of Italian official statistics and, together with the Bank of Italy, it delivers products that have significant impact on policymaking. For the compilation of such products, it relies on a business process that foresees the following phases: the institute collects, cleans and normalizes a large amount of time series data originating from heterogeneous external sources, such as other research institutes, providers of official statistics, central banks and so on; then, EIEF applies its own research models to these data, in the form of programs and scripts in domain-specific tools for economic and statistical analysis, such as R, MATLAB, MS Excel, STATA, eViews, Speakeasy, Octave, SAS and so on. As a result of these calculations, EIEF produces aggregated time series data and compiles new ones, which are then assembled into panels and datasets, which

form the final deliverables of the research process. These data are then put at the disposal of the external community, such as other researchers and policy makers.

The rise of the available time series data offers new research opportunities to the institute, allowing for the design of more refined and robust descriptive or forecasting models. Yet, the challenge is to cope with the continuously soaring volumes, which indeed put the present system under stress and affect even the ability to respond to contingent requests for data.

The main goal of SAMIZDATA project was enabling these new research opportunities and, at the same time, shortening the "time to research", that is, the amount of effort and preparatory activities needed to have the data ready for the research job and for publishing the results.

EIEF business processes can be seen as an ecosystem of applications, each encapsulating a different service or functionality. Since the production of research deliverables involves several iterations and revisions, the whole workflow is an orchestration of such services, which can be dynamically modified or rearranged according to the specific needs. The services play a client role and query SAMIZDATA to store, search, transform and export time series data, that is, to implement their lifecycle.

In this context, SAMIZDATA has a number of motivations:

- for the institute, the system is a long-term investment, since its ability to scale contains the upfront costs to the present needs, while enabling EIEF, in the future, to increase the investment and leverage the forthcoming new sources so as to craft innovative research products;
- it allows to solve the contingent needs, by providing a system that is practically able to respond to queries upon the present volume of data;
- it represents a step forward in the integration of enterprise information: in a large institute, such as EIEF, the various research groups tend to work with a strong vertical focus on their analysis goals, laying the ground for a data environment with duplicated objects, inconsistencies and low quality in general. SAMIZDATA makes up for this, by offering a set of basic functionalities to the groups and resulting in an integrated database as a side effect;
- it offers a plenty of value-added services, such as normalization and transformation of the series, conversion into many disparate formats in a unified way, guarding the quality, the homogeneity of the intermediate data products and maximizing the opportunities of reuse.

3 Achieving scale independence for time series data

A time series is a sequence of n real values $T = (x_1, \ldots, x_n), x_i \in \mathbb{R}$, each associated with a time value $(t_1, \ldots, t_n)$. It originates from a data source as the the sampling of an underlying process. Each pair (x_i, t_i) is usually referred to as an *observation*.

Although in this definition each observation carries only one real value, in a broader sense, time series represent generic events occurring at instants of time,

thus observations can also carry non-numeric values, such as strings, texts and multimedia.

SAMIZDATA is a *time series management system*: it supports the storage of such kind of data and provides with functionalities to perform queries upon them. In SAMIZDATA we enrich the time series with a set of features carrying important descriptive information. We name such features as *metadata* and model them as a function $\mu_{T_i} : \mathcal{F} \to \mathcal{V}$, for each time series T_i, where $\mathcal{F}$ is an infinite set of features and $\mathcal{V}$ a set of values for those features. For example, the metadata of a time series T_1 concerning the gross domestic product of the United States are represented by the following function: $\{\mu_{T_1}(\text{“}\texttt{description}\text{”}) = \text{“}\texttt{The GDP of the US}\text{”}$, $\mu_{T_1}(\text{“}\texttt{source}\text{”}) = \text{“}\texttt{US Federal Reserve}\text{”}$, $\mu_{T_1}(\text{“}\texttt{Num. of Obs.}\text{”}) = 3800, \ldots\}$.

Handling time series inherently poses the problem of scale, since the observations are never modified or deleted and the new ones are always appended. Moreover, the growth rate is often unforeseeable, and some series are even sampled with mutable frequency at the source.

In developing systems, while the storage needed is a very immediate aspect, the most challenging and indeed interesting problem is the ability to answer queries. As we have mentioned in the Section 1, all the major processing tasks at EIEF (such as applying the research models, export and visualization and so on) preliminarily require to query the system to individuate the specific series and completely fetch them. In this, the basic issue is performing metadata- or *feature-based queries*, that is finding the time series that have certain properties, deal with some given topics, or cover some desired subjects. Our goal in SAMIZDATA is answering to this kind of feature-based queries in a scale-independent fashion.

In other terms, we want to allow users and client applications (including the modules of SAMIZDATA) to retrieve the time series on the basis of a textual description v to be searched for among all the features; more precisely, we want to be able to return all the time series T_i that have some features $f \in \mathcal{F}$ such that $\mu_{T_i}(f) \sim v$.[1] For example, given $v =$ `"industrial production"`, all the series having `"industrial production"` as a substring of any of their features, such as the name, the description, the notes and so on, should be returned.

In order to explain the problem, let us refer to a generic *time series database* $\mathcal{D}$, which is a set $\{T_1, \ldots, T_q\}$ of time series, and to a generic a query $Q(\mathcal{D}, \mathbf{p}\ \mathbf{f})$, where $\mathbf{p}$ is the vector of features in $\mathcal{F}$ and observations that are desired as output, and $\mathbf{f}$ is the vector of features in $\mathcal{F}$ that are constrained in the query. For example $\texttt{name} \wedge \texttt{ts\_time} \wedge \texttt{ts\_data} \wedge \texttt{frequency} = \texttt{"monthly"} \wedge \texttt{description} = \texttt{"literacy rate"}$ retrieves all the series, with their observations, that have specific values for the features `frequency` and `description`.[2] Our goal is achieving in SAMIZDATA scale independence on this kind of queries, which means answering $Q(\mathcal{D}, \mathbf{f})$ with an effort that is independent of the size of $\mathcal{D}$.

To this end, we leverage the concept of *access schema*, a useful characterization, recently proposed in the database literature [7], that allows to formalize

[1] The symbol $\sim$ denotes some similarity metric between the values of $\mathcal{V}$ and v.

[2] As in our scenario all the queries return the `name` of the series as well as their observations (pairs (`ts_time`,`ts_data`)), we omit vector $\mathbf{p}$ from hereinafter.

and validate the scalability requirements. Given a relational database schema $\mathcal{R} = (R_1, \ldots, R_k)$, where R_i is a relation symbol, an access schema $\mathcal{A}$ is a set of entries of the form (R_i, X, N, T), where X is a subset of the attributes of R_i, and N and T are positive integers. A database $\mathcal{D}$ *conforms* to $\mathcal{A}$ if, for each $(R_i, X, N, T) \in \mathcal{A}$ the following two conditions hold: i) there are at most N tuples in R_i having a given combination $\hat{\mathbf{v}}$ of values for attributes X (in other words, the cardinality of a selection $\sigma_{X=\hat{\mathbf{v}}}(R)$ for any assignment $\hat{\mathbf{v}}$ of values of X is at most N); ii) such tuples can be retrieved in at most T units of time.

If a query $Q(\mathcal{D}, \mathbf{f})$ only poses conjunctions and disjunctions of conditions $f_i = v_i$ over attributes of $\mathcal{A}$ entries, we say that the query *respects* the access schema and is scale independent, as the time required to perform the query and the number of output tuples depend on Q and $\mathcal{A}$, but not on $\mathcal{D}$.

Since in our scenario we assume that all the time series are stored with the "same technique" (for example in the same relation[3] or, as we will see, as documents with a partially defined structure), we can relax the notation without loss of generality and consider entries of the form $(\mathcal{D}, X, M, T)$, meaning that a query over a set of features X retrieves a time series in $\mathcal{D}$ in at most T units of time, producing no more than N results.

Thus, in SAMIZDATA, the key to achieve scale independence requires two important high-level tasks: individuating a sufficiently general range of queries suitable for users' needs; devising an appropriate access schema, which expresses the scalability requirements for such queries. Such access schema is then used to drive and validate the actual implementation. The next two sections are devoted to these two tasks.

3.1 The queries

EIEF users and application developers show to prefer full-text and unstructured queries, where they only have to specify one value v to be searched for in all the metadata features of the various series, without even knowing the name of the respective features. The requirements explicitly involve the possibility to search for any feature, even if that specific feature is present only in a subset of the stored time series and, clearly, the ones without that property are not to be included in the results. Moreover, EIEF wants to be able to handle in the future any other time series, independently of their features, which clearly can differ from the ones already in the system.

More formally, our class of queries can be represented as $Q(\mathcal{D}, \mathbf{f}) = (f_1 = v \vee \ldots \vee f_s = v)$, where s is the maximum number of features that are compared with v for a time series. Now, in order to fully characterize our queries, we have to define a finite set $\mathcal{F}' = \{f_1, \ldots, f_s\}$ of features (with $\mathcal{F}' \subset \mathcal{F}$) that can be used to retrieve time series in the system. Since the functions μ_{T_i} that define our metadata are of course non-total (i.e. not all the series have values for all the features) and their domain varies for each time series T_i (i.e. different time

[3] Notice that having one relation for each time series would not be feasible in contexts with tens of millions of time series, such as the one at EIEF.

series frequently have non-overlapping domains), we choose $\mathcal{F}'$ as the union of all the active domains of the various μ_{T_i}. More intuitively, we allow queries on any feature of any time series. Notice that $\mathcal{F}'$ is not huge since in practice the number of unique feature names among all the time series is indeed small (a few hundreds).

As we will explain in Section 4, we build SAMIZDATA architecture in a REST-oriented style, basing all the components on top this kind of feature-based queries: `GET`, to retrieve the stored series, `DELETE` to erase them and `PUT`, to add new series and observations.

3.2 The access schema

In this Section, we start by defining an access schema that represents our scalability requirements. Then we discuss two relational approaches for building time series management systems, which are the most intuitive and immediate solutions and, indeed, often adopted; we use the defined access schema to argue that these approaches are not scale independent. Finally, we present our implementation in SAMIZDATA and show how the limitations of the two approaches are overcome. In particular, we validate our solution with the same access schema and show that our class of queries $Q(\mathcal{D}, \mathbf{f})$ is handled in a scale-independent way.

Starting from an empty access schema $\mathcal{A}$, we express our scale independence requirements by adding entries a_i to $\mathcal{A}$. For each feature f_i we add an entry $a_{f_i} = (\mathcal{D}, \{f_i\}, M, T_f)$, meaning that each feature must be sufficiently selective, so that the tuples in the result of the query are at most M and are retrieved in at most T_f. One particular feature, the `name`, deserves more care as in the system it is a unique identifier for a single time series. Therefore, we add a specific entry of the form $a_u = (\mathcal{D}, \{\texttt{name}\}, 1, T_u)$, meaning that our system should respond to name-based queries in no more than T_u units of time and return at most one result.

We now consider two relational settings, which we show to be not scale independent. In the first setting, the time series would be stored in a single relation `TS(`<u>`id`</u>`, name,` $f_1, \ldots, f_s$`, ts_time, ts_data)`, where `id` is a progressive key, `name` is the time series name (unique in the system), $f_1, \ldots f_s$ the various features, `ts_data` stands for the values of the observations and `ts_time` is the respective timestamp. Clearly, we would have an index for the `name` and each of the other features. In the relation `TS` we would have a tuple for each observation of each time series, hence violating the entry a_u, which restricts to at most one result tuple for a given `name`.[4] In this setting, we do not have any constraint on the number of time series having a specific value for a feature, hence we would also violate entries a_{f_i}. The access time constraints would be soon violated for the same reasons.

There are other important issues with this approach. While the `name` is common for all the time series, for the other features only a partial overlap can

[4] Notice that also a normalized version of `TS`, where features and observations are split in two relations, with a foreign key constraint on the series name, would violate a_u as well.

be assumed; therefore `TS` would be very sparse (with many null values) and so potentially space- and access-inefficient. Moreover, the addition of new time series to the store would cause the introduction of new features and, as a consequence, the relational schema would be subject to continuous modifications. Furthermore, indices in relational systems are suitable for exact matches, while lend themselves less effectively to full-text search and, in our scenario, metadata-based queries involve, instead, the evaluation of some similarity metric between an input text value v and the content of each feature.

Let us consider a second relational setting, where the time series would be stored in a relation `TS(id, name,` $f_1, \ldots, f_s$`, ts_data)`. Here, there is no `ts_time` attribute and `ts_data` would hold an encoding of all the time-value pairs.[5] This solution would limit the output of name-based queries to exactly one tuple. However this solution would be ineffective for the queries on the other features and, in facts, we observe that the number of distinct time series having a given value v for some feature is not limited by M in any case. Moreover, this approach has another drawback: it adopts an opaque representation of data. Since the time-value pairs are hidden in the encoding of `ts_data`, punctual access to observations is laborious, adding new values is inefficient and and the data are difficultly inspectable.

We overcome the limitations of the two relational representations we have described by proposing a *document-based data model* for time series. A *document* is a recursive key-value data structure. The keys are strings and identify the "fields" of the document, and the respective values can be primitive types, other documents (sub-documents) or lists of documents. The documents can be memorized in *document stores*, specialized NoSQL systems, which support them natively, offering a range of functionalities, such as indexing, search by field or sub-field (i.e. field in a sub-document), addition of new key-value pairs and so on. Interestingly, all these functionalities consider documents along with their sub-documents as first-class objects and support very quick read, write and append operations at this granularity.

In our data model, each time series corresponds to one single document and, in the store, indexing allows for an efficient access by `name`. Then, the actual time-value pairs are nested into each time series document. As mentioned, this nesting is transparently offered by the store. What is more, a document-based data model better supports the continuous introduction of new features, which in a relational context, would be structural elements and would require frequent schema updates. Each time series document has two sub-documents, one for metadata and one for data. The various metadata features are elements in the first sub-document and each observation is in turn a sub-document in `data`. The following JSON-like snippet exemplifies what we have in the system for the GDP of the US:

[5] Technically, this can be obtained in systems by choosing an appropriate data type for raw data, such as BLOB.

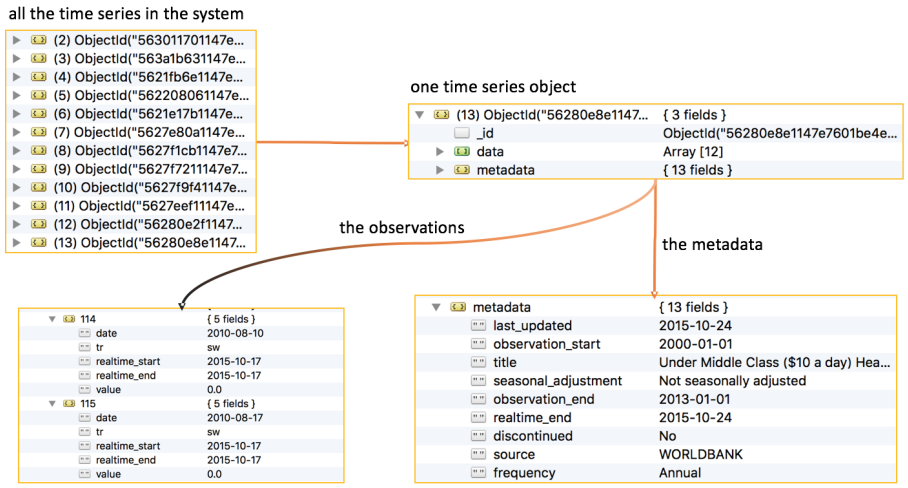

Fig. 1. Fragments from SAMIZDATA document store.

```
{ metadata:
   { id: "/fed/all/gdp", description: "The GDP of the US", ... },
data: [
   { timestamp: 2012-12-31 12:13:51...UTC, value: 16.16},
   { timestamp: 2013-12-31 12:13:51...UTC, value: 16.77}, ... ]}
```

Actually the snippet has some simplifications and in the real system even the observations are characterized by some observation-level metadata, defining specific properties. Figure 1 shows some fragments of the real document store.

We satisfied name- and feature-based queries with two different strategies. The various documents are sharded into many different physical repositories on the basis of the series name. Since the documents are the atomic elements for read and write operations (and in the store there are internal limits), we partition top frequency (i.e. longest) time series into several fixed-length documents, each with an identifier `name + offset` that can be dynamically calculated on the basis of the desired portion of the series. Then, a centralized hash function takes the name as input and allows to search the right repository and access the data. As more series come into SAMIZDATA, the system scales out by adding more shards. The access time is limited by $T_u \leq t_s + t_d$, where t_s is the time needed to access a shard individuated by the hash function and t_d is the time needed to access an indexed document within a shard. Moreover, the size of the output is limited by $N = 1$, which is the specific time series document or its portion, in case of high frequency series. Hence, for this kind of queries, the entry a_u of $\mathcal{A}$ is respected.

To satisfy the entries a_{f_i} of $\mathcal{A}$ for the feature-based queries, we adopt an approach based on a *distributed inverted index*. We consider all the fragments of text in the values of the features, ordered by selectivity, and we keep only the most selective ones. In other terms, we fix an upper bound α, denoting the maximum number of time series in which the fragment can occur to be considered. As a consequence, very common fragments (such as prepositions, articles, etc.) are discarded, while more selective ones (such as "US GDP", "industrial production", "oil") are kept. For each text fragment, we store the list of at most α identifiers of the time series in which it has been used. We also memorize the inverted-index document in the store, as exemplified by the following JSON-like example:

```
{ industrial production:    ["/wb/it/national_accounts",
                  "/fed/all/gdp", "/un/all/overall_debt",...]},
{ oil price:       ["/opec/all/op","/wb/all/oil_price",...]},...
```

We adopt sharding also in this case, partitioning the index documents by text fragment ("industrial production" or "oil price" in the example). In this way, whenever a feature-based query is issued, first SAMIZDATA accesses the right shard, it obtains the list of the names of the time series in which it appears and finally retrieves them by `name`. The overall time is limited by $T_f \leq (\alpha+1) \times t_s + t_w + \alpha \times t_d$, where t_s is the time needed to access a shard of the repository (and we access once for the index and at most α times for the respective series, in case they are all in different shards), t_w is the time needed to retrieve a document of the inverted index and t_d the time needed to access an indexed time series. Moreover, the number of the fetched documents will be limited by $M \leq 1 + \alpha$, that is, one for the index and at most α for the respective time series. All these conditions show that also entries a_{f_i} of $\mathcal{A}$ are respected.

4 Architecture of the system

As shown in Figure 2, the architecture of SAMIZDATA is composed of a core module, *SamizDB*, which implements the document-based data store that we have described in Section 3.2. *SamizDB* exposes a set of basic primitives (`GET`, `PUT`, `DELETE`), which allow to access time series data in a REST style, with the scalability properties we have discussed at length. The other modules then surround *SamizDB* and work as clients for it and are described in the following subsections.

4.1 Importing time series from external sources

SamizFunnel has the responsibility of fetching and extracting time series from third-party sources. It applies the appropriate conversions and feeds the series into *SamizDB*. The adapters that are depicted in Figure 2, mirror the present and next-future requirements of EIEF. We import data from traditional providers,

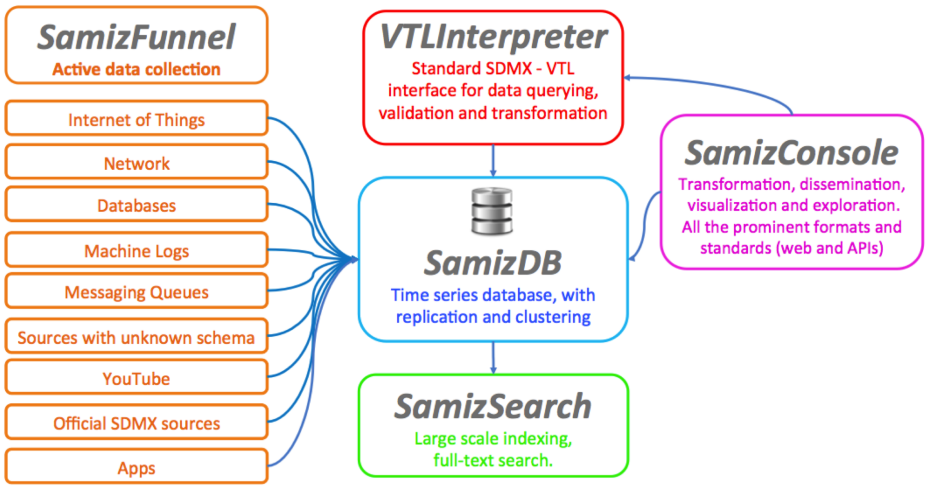

Fig. 2. The architecture of SAMIZDATA.

which offer SQL access to their databases or, more frequently, adhere to consolidated data exchange standards in the sector, such as SDMX.[6]

However, thanks to the scalability of SAMIZDATA, EIEF can take the opportunity to leverage non-conventional sources such as data from social networks, machine logs, messaging queues and even Apps and sensors of the Internet of Things. Interesting new sources, which could be used even for official statistics in the next future, come from the web, where the social media such as YouTube, for which we offer a specific adapter, represent an effective meter of people behaviors and trends of the society. *SamizFunnel* relies on a declarative approach and a semi-automatic generation of the adapters, for which we leveraged our experience on translations of database models and data exchange [2].

The adapters support two modes: *eager* and *lazy*. In eager mode, the user specifies a sampling rate, and the adapter actively pulls the new data from the source with that fixed frequency. When the sources provide new series, they are indexed by the *SamizSearch* module, and copied into the respective document store of *SamizDB*, with calls to the `PUT` primitive. In lazy mode, the time series in the sources are only indexed, yet the actual observations are not copied into *SamizDB*. In this way, the initial import is much faster, but it requires the access to real sources when the users need the actual data. Lazy mode also showed to be the best choice with top-frequency sources, since the retrieval could be done in batch mode.

[6] Statistical Data and Metadata eXchange is an international standard for the exchange of statistical data and metadata adopted by all the major providers of official statistics (www.sdmx.org).

A final important remark about this module concerns the *versioning*, which SAMIZDATA fully supports. Indeed, when *SamizFunnel* imports the series from an external source, it invokes the `PUT` primitive of *SamizDB* with the validity interval as a supplementary parameter. As it can be seen in the fragments in Figure 1, this parameter is then stored at observation level and allows to retrieve any series as of any instant of time.

4.2 Validating and transforming time series

VTLInterpreter is another satellite module, which plays a particularly relevant role for a stakeholder active in the official statistics domain, such as EIEF. Before using the time series originating from external sources, they must be validated, cleaned and normalized in order to guarantee their quality. These validation and transformation logics are expressed by the domain experts in form of *rules*, specified with a domain-specific language. *VTLInterpreter* accepts rules expressed according to an emerging standard syntax in the official statistics field.[7] The module executes these rules, by accessing *SamizDB* with `GET` operations, to retrieve the actual data and `PUT` to store the results back into *SamizDB*.

4.3 Searching, visualizing and exporting time series

SamizConsole is the interface of SAMIZDATA and consists of a web GUI and set of APIs. The web GUI basically allows to search the series in *SamizDB*, modify and export them in all the prominent formats for statistical data. All these functionalities are provided within one single dashboard (some screenshots are shown in Figure 3), which also allows for data visualization and exploration. [8] In a typical interaction, the user enters her full-text query. It is interpreted as a feature-based query as we have explained in Section 3 and compared against all the features of all the series. Then, the user chooses a series to work with and the system shows the metadata of interest for that time series in a panel. The user then indicates a version for the time series. Moreover, the user can indicate a preliminary transformation, in the sense that she may require the time series to be altered before being exported. For this purpose, the GUI offers a set of standard operations that are usually performed. The GUI then invokes the *VTLInterpreter*, asking for the application of the required transformation, fetches the time series again from *SamizDB* and shows the updated version. The user can then browse and explore the various observations, both graphically and in a textual grid. Finally, she can either export the data in a desired format or add them to a personal basket so as to download multiple series at the same time at the end of her work session.

Another option to access the data is using the REST APIs that SAMIZDATA offers for many common statistical tools. This approach is very powerful and

[7] https://sdmx.org/?page_id=5096

[8] An open but functionally limited version of the web console is available on line for demo purposes at http://www.samizdata.it

Fig. 3. Screenshots of SAMIZDATA web GUI.

only requires the client tool to support HTTP protocol. Nevertheless, handling the data in this way is laborious since the APIs return a JSON output, which must be decoded in the application logic with custom code. Hence, SAMIZDATA also offers a rich set of user-friendly higher-level libraries [9] (an example is in Figure 4).

5 Related work

The experience of SAMIZDATA shows how the recent formalization of query answering in the context of big data by Libkin et al. [7], can be applied in a principled way to design and validate the architecture of a data management system. Indeed, the results our work refers to come as a neat formalization of many years of experience where patterns and good practices have been applied in the development of scalable systems. Here we follow the reverse approach and aim at leveraging this huge expertise starting from the formal results that condense it; in particular we adapt the promising idea of access schema and use it to drive and validate the design.

In order to achieve scalability, the adoption of a document-based data model for time series is being explored in the NoSQL community,[10] as these systems also have good performances for typical time series operations, such as high-frequency inserts and aggregating. Moreover, they have great modularity, supporting, for example, different schema design choices, according to the frequency of the series and the desired performance. SAMIZDATA adopts these practices in a principled

[9] More details can be found here: http://www.samizdata.it/api_doc.html

[10] http://blog.mongodb.org/post/65517193370/schema-design-for-time-series-data-in-mongodb

```
# download from the source 'fred' the time series 'gdp'
x <- getSamiz("fred", "gdp")
```

```
# download from the source 'fred' the time series 'gdp' at the vintage date '2015-0
7-01'
x <- getSamiz("fred", "gdp", vintage="2015-07-01")
```

```
# download from the source 'worldbank' the time series 'ic.tax.totl.cp.zs',
# for the country Albania 'AL', with the transformation 'log'
x <- getSamiz("worldbank", "ic.tax.totl.cp.zs", country="AL", transformation="log")
```

```
# download from the source 'ecb' the time series 'gst_a_fi_n_b0x13_m8v_b1300_sa_e'

# at the transformation 'log'
x <- getSamiz("ecb", "gst_a_fi_n_b0x13_m8v_b1300_sa_e", transformation="log")
```

Fig. 4. Some code snippets exemplifying how to use of SAMIZDATA APIs for importing time series in R (https://cran.r-project.org).

way and extends them fulfilling the conditions modeled by the concept of access schema.

The amount of research about time series data management is impressive and related papers with a rich bibliography exist [6, 1, 5, 12, 17, 16]. However, if we just concentrate on the works concerning fast retrieval of time series data from systems, Agrawal et al. [1] in a seminal work propose one of the first approaches to index time series data, consisting in spotting specific characteristics of the series, by studying them in the domain frequency by means of Fast Fourier Transforms. Many other works then follow this approach and develop specialized methods. What we remark here is that all these techniques concentrate on the values of the series and open the way to the development of search techniques based on clustering [4]. In SAMIZDATA we address the more basic but essential requirement of retrieving the series given some metadata features. Our experience at EIEF and with many other real-world systems, proved that in the context of official statistics, such case is indeed very common and still unsolved by most applications.

6 Conclusions

We have illustrated how the application of the theory of scale-independent query answering to the domain of time series data management can lead to effective results and drive the design of a data management system for an important player in the domain of official statistics. We believe that the requirements we have faced and the techniques that we have adopted in SAMIZDATA are general enough to represent a feedback to guide database research to address present and future problems in this domain.

References

1. R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, FODO '93, pages 69–84, London, UK, UK, 1993. Springer-Verlag.
2. P. Atzeni, L. Bellomarini, F. Bugiotti, and G. Gianforme. Mism: A platform for model-independent solutions to model management problems. *J. Data Semantics*, 14:133–161, 2009.
3. Z. Bar-Joseph, G. K. Gerber, D. K. Gifford, T. S. Jaakkola, and I. Simon. A new approach to analyzing gene expression time series data. In *RECOMB*, pages 39–48, 2002.
4. P. Chang, J. Hsieh, and T. W. Liao. Evolving fuzzy rules for due-date assignment problem in semiconductor manufacturing factory. *J. Intelligent Manufacturing*, 16(4-5):549–557, 2005.
5. H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, Aug. 2008.
6. P. Esling and C. Agón. Time-series data mining. *ACM Comput. Surv.*, 45(1):12, 2012.
7. W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 51–62. ACM, 2014.
8. M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani. Mining the stock market (extended abstract): Which measure is best? In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 487–496, New York, NY, USA, 2000. ACM.
9. S. Goddard, J. Deogun, S. K. Harms, M. J. Hayes, K. G. Hubbard, S. Reichenbach, P. Revesz, W. J. Waltman, and D. A. Wilhite. A geospatial decision support system for drought risk management. In *Proceedings of the 2004 Annual National Conference on Digital Government Research*, dg.o '04, pages 50:1–50:2. Digital Government Society of North America, 2004.
10. J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.
11. R. Honda, S. Wang, T. Kikuchi, and O. Konishi. Mining of moving objects from time-series images and its application to satellite weather imagery. *J. Intell. Inf. Syst.*, 19(1):79–93, 2002.
12. F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 289–300, New York, NY, USA, 1997. ACM.
13. R. N. Mantegna. Hierarchical structure in financial markets. *European Physical Journal B*, 11:193–197, 1999.
14. E. J. Ruiz, V. Hristidis, C. Castillo, A. Gionis, and A. Jaimes. Correlating financial time series with micro-blogging activity. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 513–522, New York, NY, USA, 2012. ACM.
15. K. Uehara and M. Shimada. Extraction of primitive motion and discovery of association rules from human motion data. In *Progress in Discovery Science, Final Report of the Japanese Discovery Science Project*, pages 338–348, London, UK, UK, 2002. Springer-Verlag.

16. H. Wang, Y. Cai, Y. D. Yang, S. Zhang, and N. Mamoulis. Durable queries over historical time series. *IEEE Trans. Knowl. Data Eng.*, 26(3):595–607, 2014.
17. W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Efficient graph similarity search over large graph databases. *IEEE Trans. Knowl. Data Eng.*, 27(4):964–978, 2015.