

Form-Based Semantic Caching on Time Series

Trung-Dung Le[1(✉)], Verena Kantere[2], and Laurent d'Orazio[3]

[1] Thuyloi University, Hanoi, Vietnam
dung_lt@tlu.edu.vn
[2] University of Ottawa, 75 Laurier Ave E, Ottawa, ON K1N 6N5, Canada
vkantere@uottawa.ca
[3] Univ Rennes, 2 rue du Thabor - CS 46510, 35065 Rennes Cedex, France
laurent.dorazio@univ-rennes1.fr

Abstract. Time Series Databases Management System (TSMS) has been overcoming the Database Management Systems (DBMS) in storing vast amounts of data [33]. Nevertheless, TSMS only supports simple aggregate functions to analyze Time Series Data (TSD). Besides, to accelerate and save data transferring between clients and servers in the DBMS, semantic caching can be used. However, the semantic caching approach is not efficient because of not fully supporting aggregate functions in TSMS. Furthermore, the query result of TSD in the semantic caching technique could be huge for the in-memory database where the semantic caching technique is running on. A model-based compression can be used to compress data, reducing the data space in the in-memory database. In this paper, we present Form-based semantic caching for TSD system. The approach reduces both query result storing based on semantic caching technique and the data transfer between clients and servers. In particular, the approach accelerates up to 122 and 1.82 times the execution speed, comparing to the without cache and basic semantic caching approaches, respectively. On the public Reference Energy Disaggregation Data Set, the compression model ratio in the approach can be reached to 526.8:1.

Keywords: Semantic Caching · Time Series Data · Linear Regression

1 Introduction

Smart cities applications face increasing pressure on the number of Internet of Things (IoT) connected devices and their data. In order to organize Time Series Data (TSD), which is generated by IoT devices, the Database Management System (DBMS) technology is used to represent, store, and query sensor data. It can be standard relational systems, key-value stores, document databases, or time series stores. Some systems are designed to provide fast ingestion rates and fast selection on time ranges, such as InfluxDB [5], GridDB [4], and Warp 10^{TM} [9], called Time Series Management System (TSMS).

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024, LNCS 14813, pp. 298–315, 2024.
https://doi.org/10.1007/978-3-031-64605-8_21

The standard relational DBMS can process complex Online analytical processing (OLAP) queries, and semantic caching is often used to speed up the standard relational DBMS. However, it is hard to speed up these queries in a TSMS. The previous work, Think-Cities® [15], improves query processing by Form-based semantic caching technique in a specific TSMS, Warp 10™. However, Form-based semantic caching needs to improve experiments in TSMS.

Semantic caching is an approach which stores the answer of a query instead of just the raw data. It allows exploiting resources in the cache and knowledge contained in the queries themselves [19]. Form-based semantic caching technique is used to accelerate the complex OLAP queries in TSMS [15]. However, the sensor data can be stored in the in-memory database is huge. It leverages to reduce TSD semantic cache size to store more data in the same in-memory database memory. Besides, ModelarDB [34] uses models to store sensor data with lossy and lossless compression. This approach has not been applied to the semantic caching technique. Hence, the sensor data and semantic cache in TSMS can be compressed using *model-based* compression.

In this context of queries in a time series database system, the problem is how to improve time series processing in TMSM by the semantic caching technique and reducing the cache size in the in-memory database.

Semantic caching approaches [10,18,19,26,30,32] have been studies to reduce network traffic and improve response time. These methods can be used in the standard relational DBMS. In addition, Form-based query makes it possible to optimize queries submitted via HTML forms [28,29]. However, the existing works [10,18,19,26,28–30,32] were built for Structured Query Language (SQL) query and the backend DBMS to execute, not TSMS.

Moreover, InfluxDB [5], GridDB [4], and ModelarDB [34–36] provide fast ingestion rates, good compression, and quick selection on time ranges. However, they fail to support complicated OLAP queries [24]. Hence, Form-based semantic caching [15] is proposed as a novel to solve the limitations above based on many aggregated functions in Warp 10™ [9], a specific time series system. Nevertheless, Warp 10™ does not support Structured Query Language directly. Any query should be converted into a script written in WarpScript™ language. It is not easy to translate an OLAP query into the form of WarpScript™.

Furthermore, the sensors are sampled at regular intervals, and it is currently impossible to store the huge amounts of data points in in-memory databases. The comparison of common storage solutions in ModarDB [34] showed that the compression solution can reduce 782.87 GiB of data storing in PostgreSQL to 2.41 GiB of data in ModarDB. This solution has not been used in semantic caching techniques.

To the best of our knowledge, none of these efforts fully address the issues of Form-based semantic caching of time series data systems.

This paper expands Form-based semantic caching in the previous work [15] for a specific time series data system. The approach aims to improve the performance of query processing in TSMS. In particular, the implementation shows that the approach accelerates up to 122 times the execution speed, comparing to

the without cache approach. Comparing to the basic semantic caching technique, the proposed semantic caching can speed up to 1.82 times the execution speed, as can be shown in Sect. 4.

We also integrate the *model-based* compression to reduce the time series data storage in the semantic caching system. The *model-based* compression is also reused to restore all the data points and answer the incoming queries. In particular, the compress model ratio can be reached to 526.8:1 with the public Reference Energy Disaggregation Data Set [17], as can be shown in Sect. 4.

This paper is organized as follows. Section 2 defines the context and the motivation. Section 3 presents the detail of Form-based semantic caching approach. The implementations are described in Sect. 4. Section 5 describes the related works. The conclusion is presented in Sect. 6.

2 Preliminaries

First of all, the preliminaries of this paper show the background of techniques we use to propose Form-based semantic caching approach.

2.1 Time Series Data

Time Series Data Model can be as a relations *T(st, v)* with two attributes: st(time stamp) and v(value). For example, the time series of ocean tides heights are Time Series Data. Data in a smart city are from various sensors, such as rainwater, carbon footprint sensors, etc.

Methods for time series analysis [14,16] may be divided into two classes: frequency-domain methods and time-domain methods. The former includes spectral analysis and wavelet analysis; the latter includes auto-correlation and cross-correlation analysis. In the time domain, correlation and analysis can be made in a filter-like manner using scaled correlation, thereby mitigating the need to operate in the frequency domain. Time series analysis can be divided into linear and non-linear, and univariate and multivariate [22,25].

2.2 Linear Regression

The *model-based* compression [34] is used to compress a group of time series data by Linear Regression model:

$$y_i = \beta_0 + \beta_1 t_i + \epsilon, \tag{1}$$

where β_0, β_1 are parameter of the model; $t_i, i = 1, ..., N$, are time stamp values; $y_i, i = 1, ..., N$, are the time series data value, and ϵ is random error following normal distribution $\mathcal{N}(0, \sigma^2)$ with zero mean and variance σ^2.

The **fitted equation** is defined by:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 t. \tag{2}$$

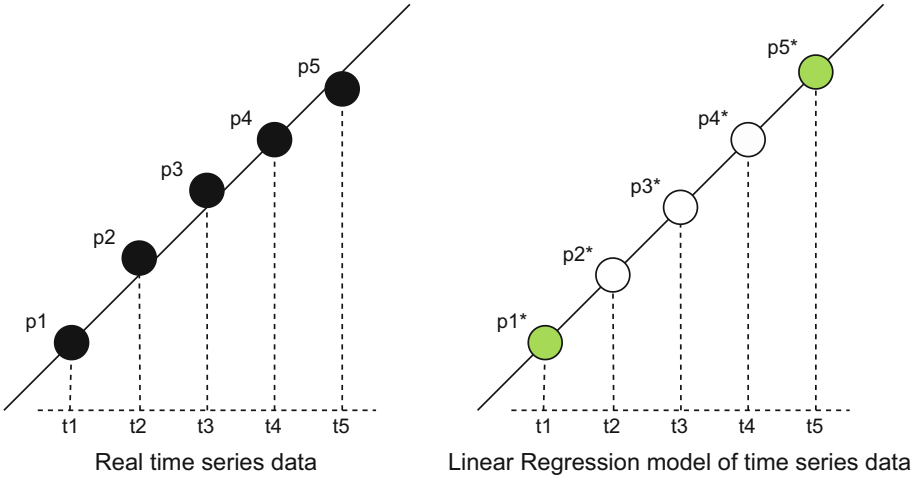


Fig. 1. Linear Regression Model.

Instead of storing all the data points, the *model-based* compression method stores the model's information, such as the size of the observation window, the min and max value of time series data. Figure 1 shows the *model-based* compression of time series data. For example, by storing $T_1(t_1, p_1^*)$ and $T_2(t_5, p_5^*)$, we can restore other data points by Linear Regression model with the user-defined error bound.

2.3 Semantic Caching

Semantic caching [10,32] allows us to exploit resources in the cache and knowledge contained in the previous queries. Consequently, it enables effective reasoning, delegating part of the computation process to the cache, reducing data transfers and the load on servers.

When a query is submitted, it is divided into two disjoint pieces: (1) a probe query, Q_{probe}, which retrieves the portion of the result available in the local cache, and (2) a remainder query, Q_{remain}, which retrieves any missing tuples in the answer from the server. If the remainder query exists, it is sent to the server for processing.

2.4 Motivation

Form-based semantic caching technique makes us possible to optimize, accelerate query processing in the environment of time series data and a specific TSMS. Moreover, we can use the *model-based* compression to store the value of data points in TSMS with the user-defined error bound or the data points in the gap time where there is no real data in the in-memory database. Using the *model-based* compression reduces TSD semantic cache size to store more data in the

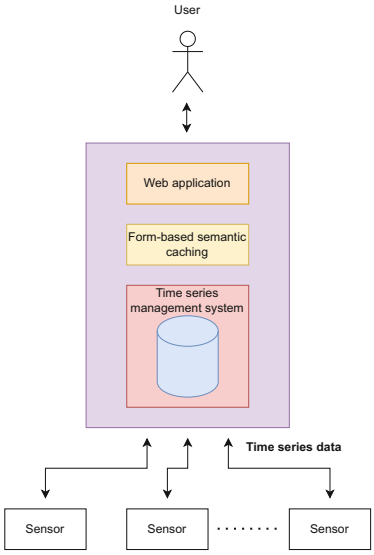


Fig. 2. The architecture.

same in-memory database memory. Also, processing a query with the *model-based* compression can reduce the volumes of data transferred between the client and server when the time data series is requested many times. Using the *model-based* compression also reduces TSD semantic cache size to store more data in the same in-memory database memory.

In conclusion, this paper introduces the novel and implementation of Form-based semantic caching on time series data to reproduce query result storing based on the semantic caching and the *model-based* compression techniques.

3 Form-Based Semantic Caching

The Form-based semantic caching architecture is shown in Fig. 2. A TSMS is used to organize TSD and Form-based semantic caching to accelerate query processing and reduce query results storing by semantic caching technique.

3.1 Form-Based Query

Example 1. The form of queries is:

```
SELECT sen.time, MIN(sen.value)
FROM SENSOR sen
WHERE T1 < sen.time < T2
```

We start with a Form-based query example. The query as shown in *Example* 1 is a function of calculating the min value of *sen.value* in a period of time from T1 to T2. When a user submits the update function in our project, a request

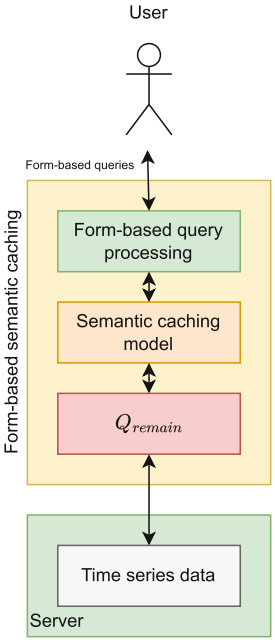


Fig. 3. Form-based semantic caching.

containing the parameters is sent to the server running a time series database engine.

Semantic caching is used to exploit resources in the cache and knowledge contained in the previous queries. Semantic caching approach is considered. When a query is submitted, only Q_{remain} is sent to the server to retrieve any missing tuples. A responding SQL query from the example template is given in *Example* 1. In particular, any aggregate function can be used in the Form-based query.

3.2 Cache Management

Memcached [6], Redis [8] are in-memory databases that are key-value stores. They are designed to simplify memory management [12]. Form-based query approach analyses, optimizes queries, and communicates through a simple set of APIs: Set, Add, Replace to store semantic data, Get or Remove to retrieve or remove semantic data. The data structure is simplified, and the access time is kept in specific period times. We use the Least Recently Used (LRU) policy for eviction of the data. Form-based semantic caching uses Memcached as the query result cache store. Furthermore, we use Linear Regression compression to reduce the data points and store these time series data on the key-value stores. By uncompressing data using model-based compression, Linear Regression model, in key-value stores, we can restore all data points of time series data with the user-defined error bound which is defined before the model is built.

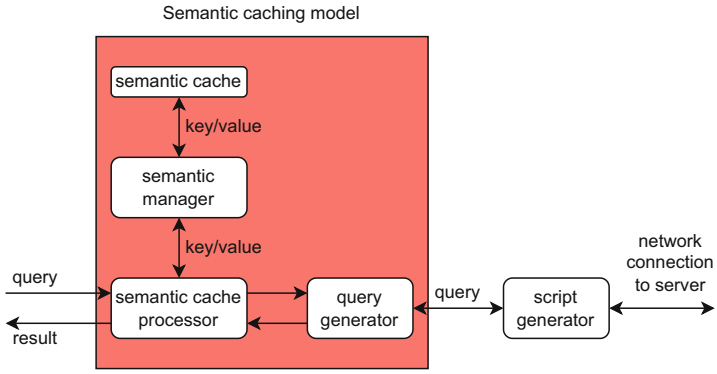


Fig. 4. The model.

3.3 Form-Based Semantic Caching Architecture

Form-based semantic caching architecture is shown in Fig. 3. It is built based on Form-based query and semantic caching approach. Form-based semantic caching allows us to exploit resources in the cache and knowledge contained in the queries themselves. Consequently, it enables effective reasoning, delegating part of the computation process to the cache, reducing both data transferring and the load on servers. The semantic caching model is built in a client in the client/server model where a time series data database engine is installed on servers to manage and simplify time series data processing.

Figure 3 presents an overview of our Form-based semantic caching approach. It relies on 3 phases: (1) Form-based querying, (2) semantic processing, and (3) time series optimization. A specific in-memory database is used to manage the *semantic cache*. When the client receives a query, as shown in Fig. 4, the *semantic cache processor* will check the previous query results via *semantic manager*. The necessary queries are generated by *query generator*. After that, Q_{remain} is translated into a script by *script generator*. Finally, the scripts are sent to servers to get the results. In particular, Form-based query processing enables users to access data via template queries. The obtained parameterized queries can then be decomposed into probe queries and remainder queries. The remainder queries will then be grouped into a query and translated into the instruction of time series database language, such as WarpScript^{TM} in Warp 10^{TM}. After that, the script is sent to the specific time series database engine to get the results. As shown in Fig. 4 and Sect. 2, the semantic caching model can be deployed with an in-memory database. The semantic caching model and time series database engine can be implemented in the separate or same servers.

Example 2. Figure 5 shows an example of Form-based query and Form-based semantic caching. The form of queries is shown in *Example* 1, where the cache has the results of queries

– From 01-02-2020 to 05-02-2020,

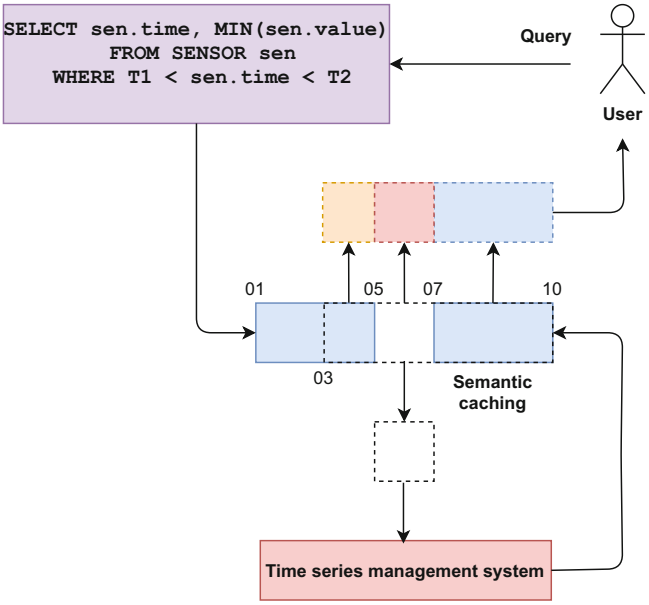


Fig. 5. An example of form-based semantic caching

– From 07-02-2020 to 10-02-2020.

The incoming query requires results from 03-02-2020 to 10-02-2020. Then, it is decomposed into Q_{probe} and Q_{remain}. Q_{probe} is optimized in the semantic caching to get results:

– From 03-02-2020 to 05-02-2020,
– From 07-02-2020 to 10-02-2020.

Q_{remain} is optimized and translated into a script. After that the script is sent and get results from Warp 10^{TM} server:

– From 05-02-2020 to 07-02-2020,

Finally, the results are integrated from 2 parts in caches and a part from distance server. They are shown as follows:

– Semantic caching: From 03-02-2020 to 05-02-2020,
– Warp 10^{TM} server: From 05-02-2020 to 07-02-2020,
– Semantic caching: From 07-02-2020 to 10-02-2020.

4 Validation

Our experiments use Memcached [6] to organize the cache on a client built-in Open JDK Java 1.8. All experiments are run on a machine with following parameters: Intel(R) Core(TM) i7-6600U CPU @ 2.60 GHz × 4, 16 GB RAM. The operating system is Linux Ubuntu 16.04.6 Long Term Support distribution based on kernel 4.15.0-106-generic.

4.1 Think-Cities®and Warp 10™

In this experiment, the servers organize time series data in Think-Cities® by Warp 10™ platform, as shown in Fig. 6. So, the remainder query should be translated into WarpScript™ scripts to process time series data.

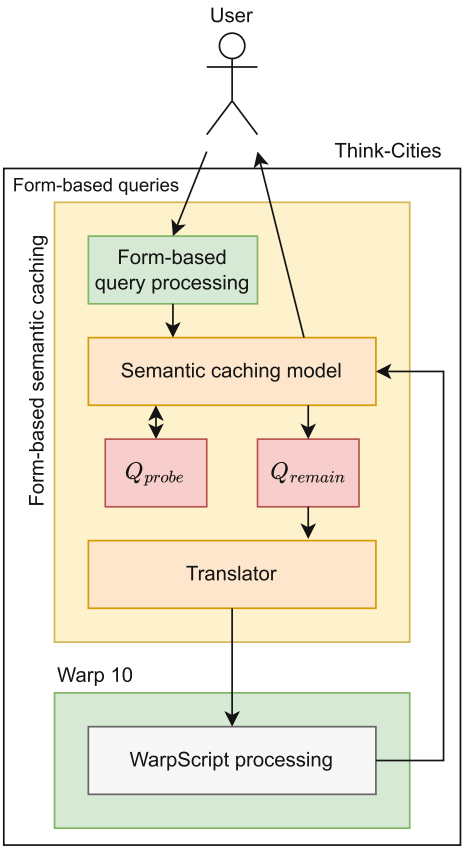


Fig. 6. Form-based semantic caching and Warp10 in Think-Cities®

Context. To analyze the efficiency in terms of time, we assume that time series data is organized in servers. A query often requires to get data in the form of an aggregate function of sensor values, such as the scoring metric of CO_2 in an area should be taken by the average function in a minute or an hour. The aggregate function of values should be calculated in the form of database engine languages, such as WarpScript™ in Warp 10™.

The results of 10 queries have already been stored in the cache, as shown in Table 1. The queries are formed in the uniform of Query X in Area *A* From *T1* to

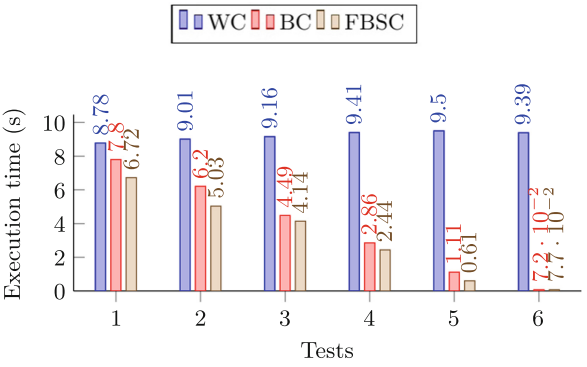
$T2$. In this form, Query X means: Select TimeStamp, AggregateFunction(x), and area A means that the values of x are taken from all the sensors in geographic A. From $T1$ to $T2$ means that time series data are taken in a period of time from $T1$ to $T2$. The data of TimeStamp and the mean value of sensors around TimeStamp can be reused in the application. For example, the metric score of CO_2 in an area in the 1^{st} quarter of 2020 can be calculated via the results of Q_1: Query X in Area A From 2020-01-01 to 2020-03-01 and Q_2: Query X in Area A From 2020-03-01 to 2020-03-31.

Results. The experiment focuses on comparing the execution time among Without Caching (WC), Basic semantic Caching (BC), and Form-Based Semantic Caching (FBSC) approaches. In BC and FBSC experiments, all the query results, as shown in Table 1, have already been stored in the semantic cache. Different workloads in various tests, as shown in Table 2, are used to compare the execution time in three methods. In the query processing, the data is transferred from servers to the client cache when queries miss hit (M) the previous results in the cache. In the basic semantic caching approach, the entire result that has already been stored in caches is returned from caches to answer the respective query, which are Hits (H) cases. In FBSC method, a part of the previous queries can be reused to return the request, which calls Partial hits (P). In particular, the approach reads all the keys in the cache and finds exactly where the answers are stored. Only the miss information in caches should be sent from servers to client caches in Form-based semantic caching. The ratios of Hits (H), Miss hits (M), or Partial hits (P) in tests are showed in Table 3.

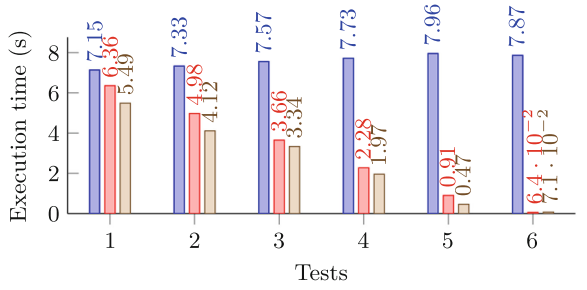
Each test runs 100 times. Their average execution time is illustrated in Fig. 7. We have 4 experiments with different sizes of caches. Figure 7a, 7b, 7c, and 7d show the experiments of 597071, 479870, 359582, and 239231 bytes in the semantic cache, respectively. As shown in Fig. 7, the response time of query processing using FBSC is shorter than using WC and BC in most cases.

Table 1. Queries are stored in the cache.

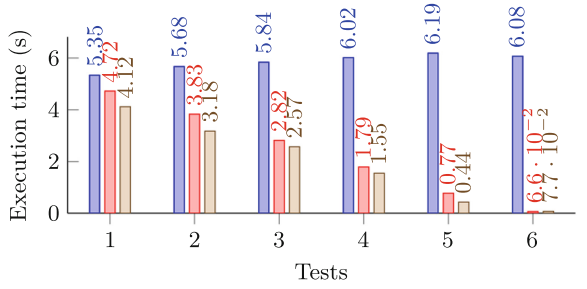
	Query	Days
Q1	Query X Area A From 2020-02-01 To 2020-02-11	10
Q2	Query X Area A From 2020-02-11 To 2020-02-21	
Q3	Query X Area A From 2020-02-21 To 2020-03-02	
Q4	Query X Area A From 2020-03-02 To 2020-03-12	
Q5	Query X Area A From 2020-03-12 To 2020-03-22	
Q6	Query X Area A From 2020-03-22 To 2020-04-01	
Q7	Query X Area A From 2020-04-01 To 2020-04-11	
Q8	Query X Area A From 2020-04-11 To 2020-04-21	
Q9	Query X Area A From 2020-04-21 To 2020-05-01	
Q10	Query X Area A From 2020-05-01 To 2020-05-11	



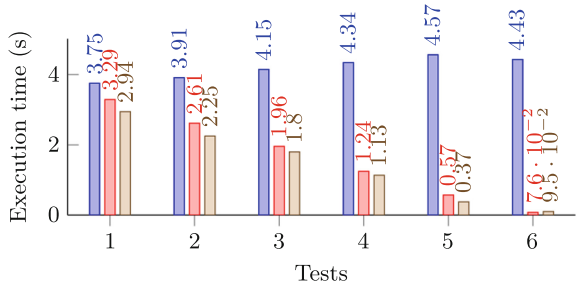
(a) 597071 bytes of cache



(b) 479870 bytes of cache



(c) 359582 bytes of cache



(d) 239231 bytes of cache

Fig. 7. Execution time in second of WC, BC and FBSC.

Table 2. Workload in tests

Test	Workload				
	Part1		Part2 (Q11)		
		Days	From	To	Days
1	Q1	10	2020-03-27	2020-06-25	90
2	Q1–Q3	30	2020-04-06	2020-06-15	70
3	Q1–Q5	50	2020-04-16	2020-06-05	30
4	Q1–Q7	70	2020-04-26	2020-05-26	30
5	Q1–Q9	90	2020-05-06	2020-05-16	10
6	Q1–Q10	100	Non	Non	0

Table 3. Percentage of hits in tests using to measure the execution time

Test	WC	BC	FBSC
1	100%M	10%H 90%M	10%H 45%P 45%M
2		30%H, 70%M	30%H, 35%P, 35%M
3		50%H, 50%M	50%H, 25%P, 25%M
4		70%H, 30%M	70%H, 15%P, 15%M
5		90%H, 10%M	90%H, 5%P, 5%M
6		100%H	100%H

4.2 Model-Based Compression

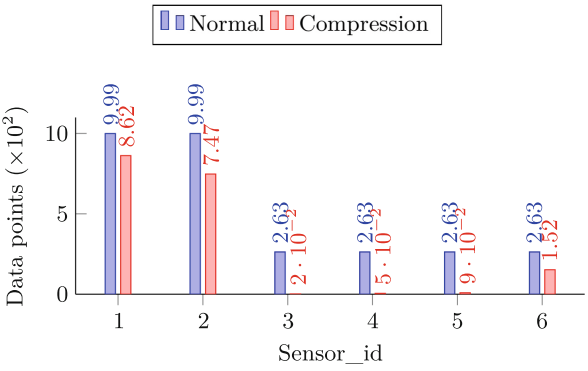
In this section, we show the semantic caching technique with and without the *model-based* compression. The data sets we used is The public Reference Energy Disaggregation Data Set (REDD) [17]. REDD includes data sets of energy consumption from six houses collected over two months. The Form-based semantic caching query has a form as shown below:

```
SELECT timestamp, value
FROM house1
WHERE sensor_id = sensor_x
AND timestamp > timeStamp1
AND timestamp < timeStamp2
```

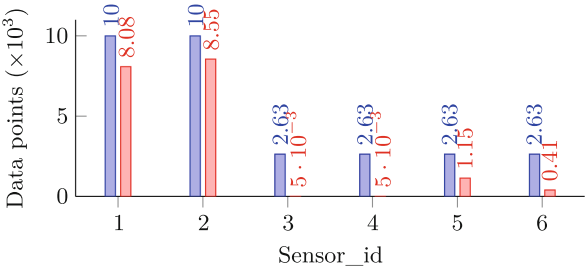
Figure 8 and Fig. 9 shows the data of House1 and House2 of REDD [17] in four cases of storing, respectively. It compares the real data points and the data stored by Form-based semantic caching. As shown in Fig. 8, and Fig. 9, the Linear Regression model's compress ratio varies from 1.1:1 to 526:1 (number of data points/number of data points in Linear Regression models on the memory, Fig. 8b).

5 Related Work

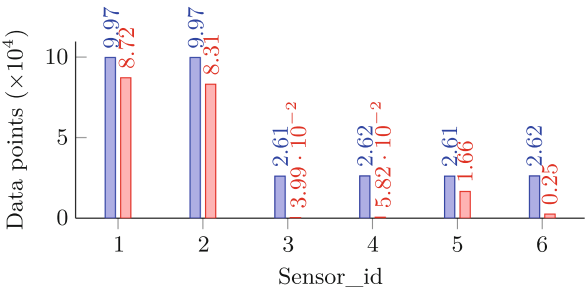
This section presents some of the main works related to big data technologies (see Subsect. 5.1), semantic caching and Form-based query (see Subsect. 5.2).



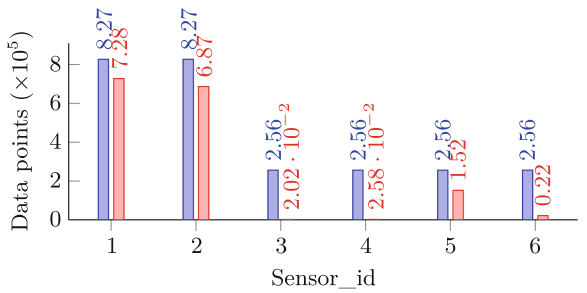
(a) 1,000 time stamps



(b) 10,000 time stamps

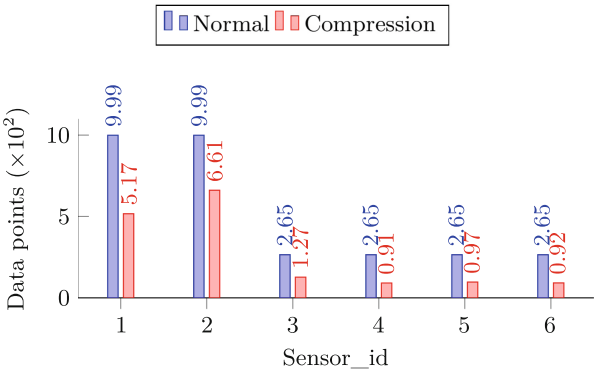


(c) 100,000 time stamps

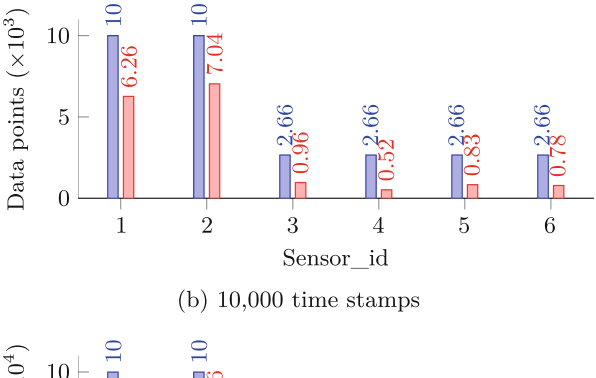


(d) 1,000,000 time stamps

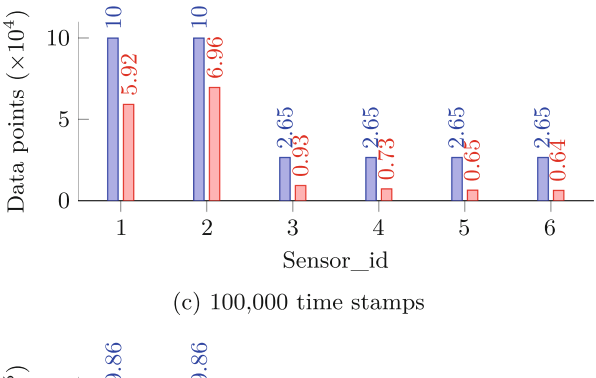
Fig. 8. Data point of time series data of house 1 of REDD.



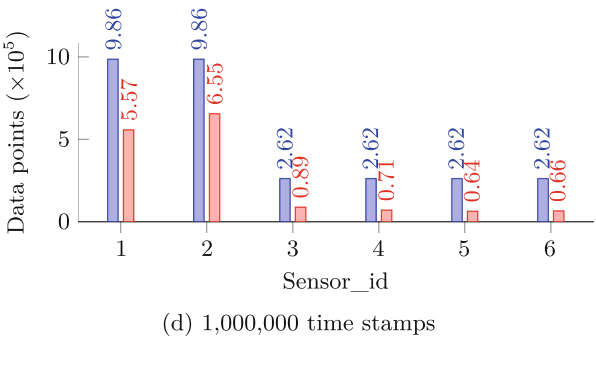
(a) 1,000 time stamps



(b) 10,000 time stamps



(c) 100,000 time stamps



(d) 1,000,000 time stamps

Fig. 9. Data point of time series data of house 2 of REDD.

5.1 Big Data Technologies

Hive [11], SparkSQL [20] provide cluster computing frameworks to manage big data. They provide the way to process complex OLAP queries on large amounts of data on multiple machines. However, they do not work well on data ingestion, simple selections, and real-time queries [24].

The stream processing systems, such as Apache Kafka [1], Storm [2], Flink [23], support fast response times on window-based continuous and real-time queries. However, they do not work well on historical data queries.

Moreover, there are various document store engines, such as MongoDB [7], Couchbase [3], and AsterixDB [31], which are used to store heterogeneous data. They support the time-applicable logical data model. However, they do not focus on time series data.

Furthermore, specialized time series database systems, such as GridDB [4], InfluxDB [5], and ModelarDB [34], are specialized time series database systems. They provide fast ingestion rates and fast selection on time ranges. However, the complex OLAP queries are not supported in these systems [24]. Although Warp 10^{TM} [9] supports many aggregated functions, the scripts should be written in WarpScript^{TM} language. It is not easy to translate an OLAP query into the form of WarpScript^{TM}.

5.2 Semantic Caching and Form-Based Query

As defined in [21], semantic caching is a technique that includes three key features. First, a description of the data stored is maintained in the cache in the form of brief specifications. Second, the policies of replacement policies are flexible in different semantic regions. Third, a semantic description of the data is maintained using sophisticated value functions that incorporate semantic notions locality.

A query cache using a semantic caching approach allows reducing data transfer between clients and servers [10,18,19,26,30,32]. While [32] studied the semantic data caching and replacement approach for a client-server model, Qun Ren et al. [30] focuses on select-project queries in the general application. A mobile environment of semantic caching is also studied in [18]. Besides, a mobile dual cache and a proxy dual cache are presented in [19]. Furthermore, to prepare the relevance cache, the semantic caching for rewriting aggregate queries is considered in [13,26,27]. They use functional dependencies to rewrite an aggregate query to optimize the storage required for materialized views.

Moreover, Form-based query via HTML forms [28,29] is used to optimize queries. They parameterize the query in a form. However, semantic caching is not considered in these methods. Hence, they can be integrated with the semantic caching approach in parameterizing the query.

Furthermore, the compress method in [34] has been integrated into Spark and Cassandra to manage time series data. This solution has not been applied in the semantic caching technique.

5.3 Limitations

In conclusion, TSMS has not supported the semantic caching, and complex OLAP queries as the standard DBMSs have done. The existing Form-based query methods do not fully address semantic caching issues for complex OLAP queries and time series data. Furthermore, Think-Cities® project [15] proposed Form-based semantic caching as a novel to improve query processing in a specific TSMS, Warp 10^{TM}. However, the novel needs to have experimented and extended to organize TSD in many TSMSs. Moreover, the *model-based* compression has not been applied in semantic caching technique as ModelarDB has done with Spark and Cassandra. To the best of our knowledge, none of the existing approach efforts fully address complex OLAP query processing issues in TSMS, and reducing TSD semantic cache size.

6 Conclusion

This paper introduces a Form-based semantic caching approach for time series database systems. Besides, we compress time series data and store it in the key-value store by the *model-based* compression, which helps us restore all the data points of time series data with the user-defined error bound.

We are currently implementing and validating these contributions and integrating both synthetic and real data. Future work includes more heterogeneous caches management. Besides, we will study traditional and specific policies for traditional and semantic caching.

Acknowledgment. The authors would like to thank members of SHAMAN team at Univ Rennes, CNRS, IRISA; School of Electrical and Computer Engineering, National Technical University of Athens; and Faculty of Electrical & Electronics Engineering, Thuyloi University for insightful comments.

References

1. The Apache Kafka, Stream Processing Engine. https://kafka.apache.org
2. The Apache Storm, Stream Processing Engine. https://storm.apache.org
3. The Couchbase, NoSQL Database. https://www.couchbase.com
4. The GridDB, NoSQL Database System For IoT. https://griddb.net/en
5. The InfluxDB, Time Series Database System. https://www.influxdata.com
6. The Memcache. https://memcached.org
7. The MongoDB. https://www.mongodb.com
8. The Redis. https://redis.io
9. The Warp10. https://www.warp10.io
10. Keller, A.M., Basu, J.: A predicate-based caching scheme for client-server database architectures. VLDB J. **5**(1), 35–47 (1996)
11. Thusoo, A., et al.: Hive: a warehousing solution over a map-reduce framework. PVLDB **2**(2), 1626–1629 (2009)

12. Wiggins, A., Langston, J.: Enhancing the Scalability of Memcached. In Intel document (2012). http://software.intel.com/en-us/articles/enhancing-the-scalability-of-memcached
13. Laurent, D., Spyratos, N.: Rewriting aggregate queries using functional dependencies. In: MEDES 2011, pp. 40–47 (2011)
14. Keogh, E.J., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowl. Inf. Syst. **7**(3), 358–386 (2005)
15. Carfantan, G., et al.: Think Cities: the accelerator for sustainable planning. In: ICDE Workshops 2020, pp. 64–70 (2020)
16. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: DMKD 2003, pp. 2–11 (2011)
17. Kolter, J.Z., Johnson, M.J.: REDD: a public data set for energy disaggregation research. Artif. Intell. **25** (2011)
18. Ken, C.K., Lee, H.V., Leong, A.: SI: semantic query caching in a mobile environment. ACM SIGMOBILE Mob. Comput. Commun. Rev. **3**(2), 28–36 (1999)
19. d'Orazio, L., Traoré, M.K.: Semantic caching for pervasive grids. In: IDEAS 2009, pp. 227–233 (2009)
20. Armbrust, M., et al.: Spark SQL: relational data processing in spark. In: SIGMOD Conference 2015, pp. 1383–1394 (2015)
21. Maghzaoui, M., d'Orazio, L., Lallet, J.: Toward FPGA-based semantic caching for accelerating data analysis with spark and HDFS. In: ISIP 2018, pp. 104–115 (2018)
22. Taniguchi, M., Matsunaka, R., Nakamichi, K.: A time-series analysis of the relationship between urban layout and automobile reliance: have cities shifted to integration of land use and transport? 415–424 (2008). https://doi.org/10.2495/UT080411
23. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache Flink^{TM}: stream and batch processing in a single engine. IEEE Data Eng. Bull. **38**(4), 28–38 (2015)
24. Gupta, P., Carey, M.J., Mehrotra, S., Yus, R.: SmartBench: a benchmark for data management in smart spaces. Proc. VLDB Endow. **13**(11), 1807–1820 (2020)
25. Vijai, P., Sivakumar, P.B.: Design of IoT systems and analytics in the context of smart city initiatives in India. Procedia Comput. Sci. **92**, 583–588 (2016). ISSN 1877-0509
26. Perriot, R., d'Orazio, L., Laurent, D., Spyratos, N.: A semantic matrix for aggregate query rewriting. In: ISIP 2015, pp. 46–66 (2015)
27. Perriot, R., d'Orazio, L., Laurent, D., Spyratos, N.: Rewriting aggregate queries using functional dependencies within the cloud. In: ISIP 2013, pp. 31–42 (2013)
28. Luo, Q., Naughton, J.F.: Form-based proxy caching for database-backed web sites. In: VLDB 2001, pp. 191–200 (2001)
29. Luo, Q., Naughton, J.F., Xue, W.: Form-based proxy caching for database-backed web sites: keywords and functions. VLDB J. **17**(3), 489–513 (2008)
30. Ren, Q., Dunham, M.H., Kumar, V.: Semantic caching and query processing. IEEE Trans. Knowl. Data Eng. **15**(1), 192–210 (2003)
31. Alsubaiee, S., et al.: AsterixDB: a scalable, open source BDMS. Proc. VLDB Endow. **7**(14), 1905–1916 (2014)
32. Dar, S., Franklin, M.J., Þór Jónsson, B., Srivastava, D., Tan, M.: Semantic data caching and replacement. In: VLDB 1996, pp. 330–341 (1996)
33. Jensen, S.K., Pedersen, T.B., Thomsen, C.: Time series management systems: a survey. IEEE Trans. Knowl. Data Eng. **29**(11), 2581–2600 (2017). https://doi.org/10.1109/TKDE.2017.2740932

34. Jensen, S.K., Pedersen, T.B., Thomsen, C.: ModelarDB: modular model-based time series management with spark and Cassandra. Proc. VLDB Endow. **11**(11), 1688–1701 (2018)
35. Jensen, S.K., Pedersen, T.B., Thomsen, C.: Scalable model-based management of correlated dimensional time series in ModelarDB+. In: IEEE 37th ICDE, pp. 1380–1391 (2021)
36. Jensen, S.K., Thomsen, C., Pedersen, T.B.: ModelarDB: integrated model-based management of time series from edge to cloud. In: Hameurlain, A., Tjoa, A.M. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems LIII. LNCS, vol. 13840, pp. 1–33. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-662-66863-4_1