Log Based Method for Faster IoT Queries

Anubha Jain
DA-IICT
Gandhinagar, India
201511029@daiict.ac.in

Trupti Padiya
DA-IICT
Gandhinagar, India
trupti_padiya@daiict.ac.in

Minal Bhise
DA-IICT
Gandhinagar, India
minal_bhise@daiict.ac.in

***Abstract*—With increase in size of Internet of Things IoT device networks and applications, tremendous increase is witnessed in IoT data. Efficient storage and querying of this data is of major concern. This paper addresses the issue of faster query processing of this data. It presents a Log Based Method LBM to store and query IoT data in Resource Description Framework RDF format. LBM exploits skewedness in access pattern of records by analyzing query workload and partitions the basic triple table into hot and cold sections. Our method is able to answer 64% queries with 83% of time gain, using only 8% of identified hot data over basic triple table. Parallel execution of LBM is found to be 69% faster than its serial execution.**

Keywords—data partitioning; Hot data; IoT; Log Based Method; QET; RDF

I. Introduction

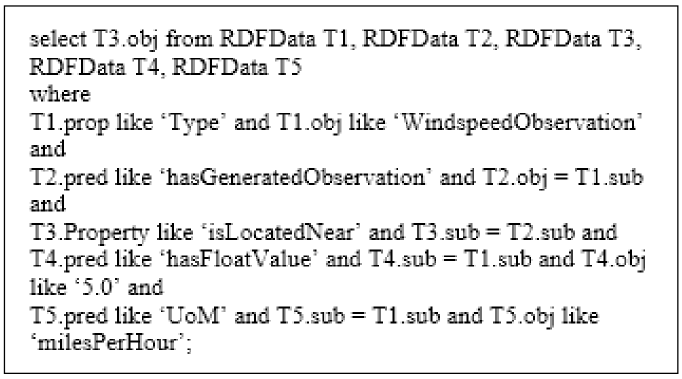
For a smart city's development, smart decisions are required to be taken for managing the public resources which include transport, traffic, energy, health care, waste management and other community services. These decisions should be based on the correct analysis of data integrated by real time monitoring systems of these resources. IoT is an internetwork of such monitoring devices referred to as things. It is a grid of sensors which collect and exchange useful data with other devices. With continuous expansion of the scope of smart city, there has been a rapid increase in use of these devices. According to Cisco Visual Networking Index [1], over the next five years, global IP networks will support up to 10 billion new devices and connections, increasing from 16.3 billion in 2015 to 26.3 billion by 2020. With the increase in connected devices, the volume, velocity and variety of data also multi folds. To be able to process this huge data, we need an efficient solution to store it.

When we analyze any IoT based application, we observe non-uniform access pattern of its data. Some data is queried more frequently than the other e.g. only recent weather data is required to forecast real-time weather. This asymmetry in access patterns leads to data skewedness. There are two factors leading to it: First is natural skewedness, which means that only few properties from the dataset are required to answer the query e.g. to determine the sales in a month, we require only price of items and not the other details. Second factor is data ageing, it is when some recent records in dataset are queried more often than the older entries. As the age of rows increase, their chances of access decrease. Evidently only some data is required to solve a query. Experiments conducted in [2], shows that only 9% of the data is required to solve 73% of the most frequent queries. Thus, if we figure out the patterns of natural skewedness and data ageing on the dataset, we get frequently accessed records. When we put that data in a separate set then most of the queries can be answered by that subset itself and we will not have to analyze the complete dataset.

RDF is accepted as a standard format for storing IoT data [3]. It provides standard libraries and formats so that data from a system can be mapped and connected to data of other systems. It allows exchange of data among devices and to run distributed queries on IoT data. The interconnection of data from different domains can be used to build applications for smart cities. RDF data is in the form of triples. Each triple follows subject-property-object format. Here, subject represents resource, property represents traits of the resource and object represents the value of that trait. The data from sensors can be collected in the form of statements about resources. The data produced by IoT devices can be represented in RDF form using suitable ontologies e.g. M3, SSN ontology and many others. RDF data can be stored using a relational approach as a single table having 3-columns schema (subject, property, object) called triple table. Each RDF triple is stored as a row in the table as shown in Table I.

TABLE I. EXAMPLE OF RDF DATA STORED IN RELATIONAL SCHEMA

Subject	**Property**	**Object**
System1	isLocatedNear	Location1
System1	hasGeneratedObservation	Observation1
Observation1	Type	WindspeedObservation
Observation1	samplingTime	TimeInstant1
Observation1	hasFloatValue	5.0
Observation1	UoM	milesPerHour



```
select T3.obj from RDFData T1, RDFData T2, RDFData T3,
RDFData T4, RDFData T5
where
T1.prop like 'Type' and T1.obj like 'WindspeedObservation'
and
T2.pred like 'hasGeneratedObservation' and T2.obj = T1.sub
and
T3.Property like 'isLocatedNear' and T3.sub = T2.sub and
T4.pred like 'hasFloatValue' and T4.sub = T1.sub and T4.obj
like '5.0' and
T5.pred like 'UoM' and T5.sub = T1.sub and T5.obj like
'milesPerHour';
```

Fig. 1 SQL Query

As shown in Fig. 1, simple query as "Determine the location of systems where wind speed is measured as 5.0 miles/hour" requires 5 self joins on the triple table. To answer typical queries, more number of self joins will be required. Imposing self joins on huge triple table requires large amount of memory. It also makes the query execution very slow. The system developed using this data storage approach is quite slow, poorly scalable and is practically infeasible. To build fast applications, we need a better way to store RDF data than the triple table which can perform well in case of scaling of system. A better approach is to partition the data in smaller sections. To answer a query, we may not require all the sections. This approach takes less amount of time as the size of data to be scanned to answer the query is less. Though this approach will also require joins, but as the table size is smaller, the joins will be feasible.

II. RELATED WORK

A lot of research has been done to effectively partition RDF data. It can be done based upon either the characteristics of data or the query workload executed on it. Former is a data-centric approach. One such approach identifies the unique properties in the RDF data [4]. A binary table is created for each property. Triples defined for that property value are stored in its respective binary table. But if a query accesses several properties, multiple binary tables have to be consulted. This increases the number of joins among binary tables required to answer the query. Another data-centric method clusters RDF data with related properties defined for a large number of subjects, in a table. Rest of the data is stored in binary tables [5]. Similar approach is discussed in [6] to partition IoT data represented in RDF format.

Above approaches clusters data which is defined together. But as the data access is not uniform, it is more suitable to store data which is accessed together in a cluster. Considering the skewedness in data access patterns, several works have been done to partition RDF data. These are called workload aware data partitioning methods. A method proposed in [7], identifies frequently accessed data of a relational dataset by analyzing log of record accesses. It uses methods based on exponential smoothing to analyze the log file. Another work suggested in [2] clusters records based on properties which are queried together. It uses apriori algorithm to distribute data over a combination of binary and property tables.

We propose a method to use log based approach to find frequently accessed records in an IoT application where its data is represented as RDF triples. Rest of the paper is organized in following sections. Section III presents our proposed solution. Section IV discusses the experimental setup followed by results in section V. Section VI concludes the paper.

III. PROPOSED SOLUTION

The proposed solution exploits the fact that data has skewedness in its access pattern. It can be discovered by analyzing the usage of data. A log is maintained for each access of any record. After analyzing this log, frequently accessed data can be identified. This analysis can be done online or offline. Offline analysis on a separate machine does not affect any database schema or transactional process on the main machine. Highly accessed data identified from it is called as hot data and rest as cold data. RDF data from triple table is horizontally partitioned into hot and cold data sections. The method is applied to find out the most queried records ids from triple table [7]. For this a log of all transactions are maintained. Each record access is logged as (record id, time slice). Time slice is denoted as (t_n, t_{n+1}), where t_n and t_{n+1} are start time and end time of a transaction respectively.

Log Based Method LBM: To estimate each record's access frequency, it uses exponential smoothing. This method estimates the current record access frequency using the previous access frequencies of the record. More weight is given to recent observations of the record than the older observations. The weight given decreases exponentially as the record access becomes older. The algorithm scans the log in forward direction. When a record entry is found in log at time slice (t_n, t_{n+1}), its estimated access frequency is calculated as:

$$est_r(t_n) = \alpha + est_r(t_{prev}) * (1-\alpha)^{(t_n - t_{prev})}$$

Here, t_{prev} is time slice when record r was last accessed. t_n is the end time slice. Here α determines the rate at which the previous estimates delay. It is typically set in the range of 0.01 to 0.05. Lower values of α give less weight to newer observations of the record. In a similar way, we calculate estimate access frequency of each unique record that we encounter in the log file. At the end of log file, it finds K records with highest estimated access frequency. These records form the hot set of records. We separate hot set of records from the cold set.

This method requires tagging every tuple with their frequency of access. The complexity of it is directly proportional to the number of accesses recorded in the log file. With increase in log size, we can run the algorithm in parallel to reduce its processing time. In case of large log files, samples from the log file can be drawn for the analysis using any systematic sampling technique. Using sampling, the complete log file need not to be processed which saves time.

IV. EXPERIMENTAL SETUP

The algorithm is implemented using eclipse 4.6.0, java 1.7 and postgreSQL 9.5.3 on a system with 32GB of RAM and Scientific Linux release 7.1 as operating system.

A. *Data Set*

The experiment uses benchmark datasets, LinkedSensorData LSD and LinkedObservationData LOD [8]. It is aggregated sensor data and weather data collected from ~20,000 weather stations in the United States and is represented in RDF format. Former is a RDF dataset containing expressive descriptions of weather stations. On average, there are about five sensors per weather station measuring phenomena such as temperature, visibility, precipitation, pressure, wind speed, humidity, etc. In addition to location attributes such as latitude, longitude, and elevation, there are also links to locations in Geonames that are near each weather station. And the latter is an RDF dataset containing expressive descriptions of hurricane and blizzard observations in the United States. The observations collected include measurements of phenomena such as temperature, visibility, precipitation, pressure, wind speed, humidity, etc. These observations are generated by weather stations described in the LSD dataset. Currently, this dataset contains almost two billion triples. The data collected from these sensors can be used to predict the abnormalities in weather conditions in any area. Such data can provide timely hurricane and blizzard warnings. It can be used in extreme weather conditions to take informed decisions.

B. *Query Set*

Experiment uses benchmark query set for LOD and LSD data. It contains 20 queries. Each query has a frequency assigned to it which is the number of times it will be executed. Frequent and most frequent queries are identified from the query set. Queries are arranged in non-increasing order of their frequencies. From this, top 50% queries are considered as frequent queries. Top 50% frequent queries are considered as most frequent queries.

C. *Log Generation*

Depending on the type of application, some queries are more likely to be asked than the others. In our experiment, to efficiently represent the practical workload, a log file is generated using zipf distribution. As we have created a zipf distribution of queries so their accessed records will also follow the same distribution. We have identified ranking of queries based upon their likelihood of querying. According to zipf distribution for each query i,

$$R_i * f_i = c$$

where, R_i is the rank of query i, f_i is the frequency of execution of query i and c is a constant. By fixing a value of constant, we can determine frequency of each query as we know its rank. To generate the log file, the queries are executed in non-increasing order of their frequencies. The most frequent query is followed by the second frequent query and so on. The least frequent query is executed at the end. Each query logs the record accesses as <Record ID, Time slice> in a log file. A log of any size can be generated by changing the value of constant. A value of constant as 100.0 generates a log of 47.5M record accesses.

D. *Experimental Flow*

The experimental flow is as shown in Fig.2. The RDF data is represented in a triple table. The data access pattern is identified using LBM and the triple table is partitioned in hot and cold sections. The query set is executed on both triple table and the hot and cold sections.

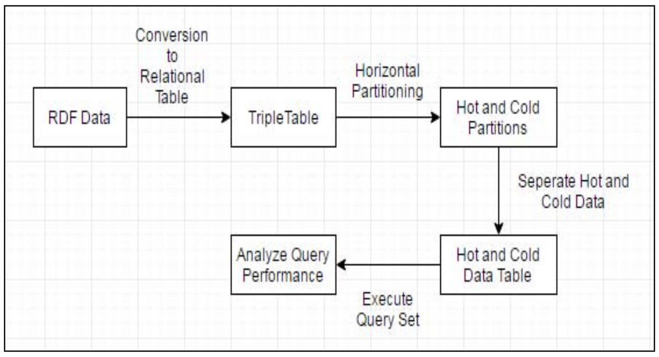


Fig. 2 Experimental Flow

V. RESULTS

This section includes results and discussions of our experiments. Analysis of the queries, workload answered by hot data is presented. Query performance is analyzed in terms of Query Execution Time QET. It also discusses the scaling of serial and parallel execution of LBM.

A. *Hot Data and Workload Answered*

As shown in Fig.3, using 8% of data as hot, 64% of all queries and 75% of frequent queries are answered. It constitutes 78.8% of query workload and 84% of frequent query workload. Using 10% of data, all queries are answered.

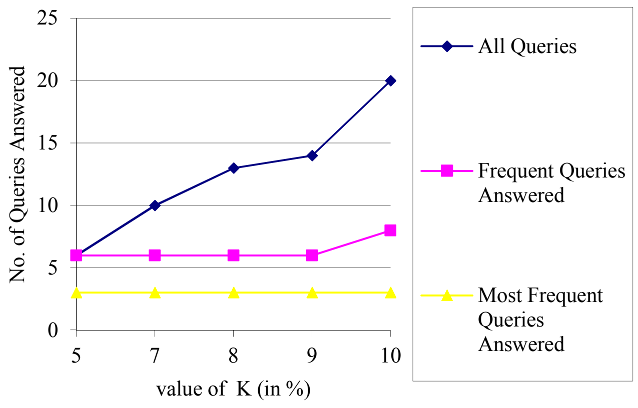


Fig. 3 Hot Data Size vs Queries Answered

We can consider 10% of data as hot in this case, but with the increase in size of dataset, the difference between the 8%

and 10% data widens. Thus, we should choose the hot data size based on the available memory in our system.

TABLE II. WORKLOAD ANSWERED USING 8% HOT DATA

Query Specification	% of Queries	% of Total Workload
All queries answered by hot data	64	78.8
Frequent queries answered by hot data	75	84
Most Frequent queries answered by hot data	100	100

B. *Query Execution Time QET*

It is the total time of query execution for hot run of query set. In a hot run, the query results are available in cache. For each query, execution time is calculated by average time of 3 hot runs. Average QET of hot run for triple table is 28.59s. After the partition, average QET of hot run is reduced to 4.89s. Thus, we are getting a time gain of 83%.

TABLE III. TIME GAIN IN QUERY EXECUTION

Query Specifications	Time Gain (in %)
All Queries	83
Frequent Queries	94
Most Frequent Queries	58

For frequent queries, average QET of hot run for triple table is 65s. After the partition, the average QET of hot run is 3.46s. For our experiment, time gain is 94% over triple table. For most frequent queries, average QET of hot run for triple table is 3.4s. After the partition, the average QET of hot run is 1.45s. For our experiment, time gain is 58% over triple table.

C. *Algorithm Scaling*

LBM is executed in serial as well as parallel. We have used 8 threads for parallel execution. Fig. 4 shows the time taken by both versions of LBM on various log sizes. In real world systems, the workload is dynamic and changes with the usage of application. To make this work applicable to accommodate these changes, the identification algorithms can be re-run frequently. The frequency depends on algorithm complexity and system characteristics.

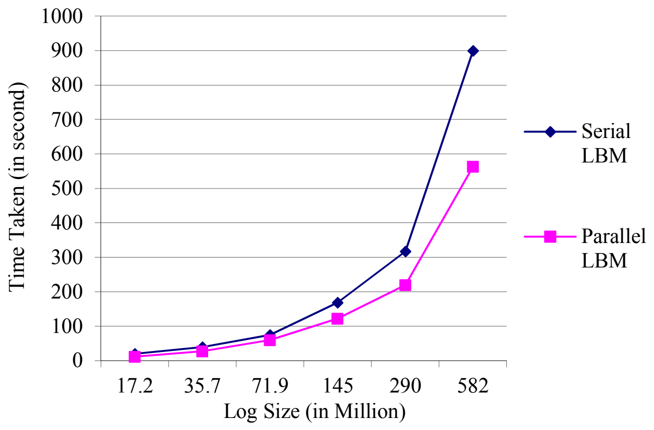


Fig. 4 Time taken by serial and parallel execution of LBM

VI. CONCLUSION

We have used Log Based Method to efficiently store and query IoT data in RDF format. The presented method exploits the skewedness in data access pattern to partition it in hot and cold sections. Using this method, 64% queries are solved with an average time gain of 83% using only 8% hot data. LBM is executed in serial as well as parallel. The parallel execution is 69% faster than the serial execution. To further accelerate the queries, the hot data partition can be moved to main memory. It will eliminate the time required to fetch data from disc and thus queries can be answered faster.

REFERENCES

[1] Cisco's Technology News Site. (2016, June 07). *Cisco Visual Networking Index Predicts Near-Tripling of IP Traffic by 2020* [Online]. Available: https://newsroom.cisco.com/press-release-content?type=press-release&articleId=1771211

[2] Trupti Padiya, Jai Jai Kanwar and Minal Bhise, "Workload Aware Hybrid Partitioning," *Proceedings of the 9th Asnnual ACM India Conference*. ACM COMPUTE, 2016, pp. 51-58.

[3] C. C. Aggarwal, Naveen Ashish, and Amit Sheth, "The Internet of Things: A Survey from the Data-Centric Perspective," *Managing and mining sensor data,* New York, USA, Springer, 2013, pp. 383-428.

[4] Daniel J. Abadi, Adam Marcus, Samuel Madden and Kate Hollenbach, "SW-Store: a vertically partitioned DBMS for Semantic Web data management," *The VLDB Journal—The International Journal on Very Large Data Bases* 18.2, 2009, pp. 385-406.

[5] Justin J. Levandoski and Mohamed F. Mokbel, "RDF Data-Centric Storage," *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 911-918.

[6] Trupti Padiya, Minal Bhise and Prashant Rajkotiya, "Data Management for Internet of Things," Proceedings of the 2015 IEEE Region 10 Symposium TENSYMP, pp. 62-65.

[7] Justin J. Levandoski, Per-Åke Larson and Radu Stoica, "Identifying hot and cold data in main-memory databases," *Data Engineering (ICDE), 2013 IEEE 29th International Conference*, pp. 26-37.

[8] Harshal Patni, Cory A. Henson and Amit P. Sheth, "Linked sensor data," *Collaborative Technologies and Systems (CTS), 2010 International Symposium on*. IEEE, 2010, pp. 362-370.