On Approximate Querying Large-Scale JSON Data

Teng Lv¹, Ping Yan², * Weimin He³ and Tao Wang⁴

1. School of Information Engineering, Anhui Xinhua University, Hefei, Anhui Province, People’s Republic of China
2. School of Science, Anhui Agricultural University, Hefei, Anhui Province, People’s Republic of China
3. Department of Computing & New Media Technologies, University of Wisconsin-Stevens Point, SP, WI 54481, United States of America
4. Department of Market Order, Bureau of Commerce of Anhui Province, Hefei, Anhui Province, People’s Republic of China
Email: LT0410@163.com, want2fly2002@163.com(*corresponding), whe@uwsp.edu, wanderbj@126.com

Abstract. JSON is a kind of data exchange format based on data types of JavaScript. JSON has been widely used in data exchange applications. To process and query large JSON file is time consuming. It requires not only massive storage disks to store large amounts of JSON data and its intermediate results, but also large CPU times to query and process the massive JSON data. So ordinary methods usually face many difficulties in such cases. Therefore, the key problem of query processing large JSON file is how to process queries efficiently. Therefore, approximate processing large-scale JSON query is an effective way to reduce query response time. So research of approximate query of large-scale JSON data can improve user's query experience and query effect. This paper researches the major questions of approximate query of large JSON file, such as core query features of JSON model, and methods of approximating query large-scale JSON data.

1. Introduction
We have entered the so called "big data" society. The key theories and technologies of "big data" is an engine driving the rapid development of all walks of life in society. Scientific research represented by data science is a new generation of scientific research paradigm after experimental science, theoretical science and computer simulation.

As a data representation and exchange format, JSON (JavaScript Object Notation) is based on JavaScript data types. JSON is user friendly for humans and readable for computers. It is a dominant data exchange format in World Wide Web. In data intensive applications, JSON can be used to construct NoSQL DBS (database systems) and graph-based DBS. A JSON file is logically organized as a set of key/value pair, where the "value" of a key/value pair can also be a JSON file again, which leads to hierarchy of nesting. A detailed analysis of JSON model and schema can be found in Refs. [1,2].

To query and process large JSON file is time consuming. It requires not only massive storage disks to store large amounts of JSON data and intermediate results, but also large CPU times to manage and process the massive JSON data. Because of the huge amount of data, it is difficult to meet the users’ needs and impossible to get an accurate result in time[3]. Therefore, one of the main problems of query processing large JSON file is how to process queries efficiently, especially in online or real-time applications. It is realistic to get an approximate result to the query in an acceptable time than an exact
result to the query in an unacceptable time[4]. Therefore, approximate processing of large-scale JSON queries is an effective way to reduce query response time. To sum up, researching approximate query of large-scale JSON data has significant effects both in theoretical and technological aspects.

There is no common agreement on many problems of JSON query. These problems can be summarized as follows: what core query features should JSON model support, and how to approximate query large-scale JSON data. The following is an analysis of the relevant research on these issues.

2. Queries of JSON Data

Unlike relational databases and XML data, which already have relatively mature and recognized query language standards, such as SQL, XPath and XQuery, there is no unified standard for JSON query. There are many JSON query languages, in which some representative JSON query languages are: SQL+ [5], XPath-based JSONPath and XQuery-based JSONiq, and so on. These query languages have many differences in grammar and supported operation types. JNL (JSON Navigation Logic) [6] is a navigation query language based on MongoDB’s “find” function. It takes into account the deterministic characteristics of JSON and can handle recursive queries, but it does not realize the “projection” function of “find” function. J-Logic [7] is a JSON query language based on DataLog, which provides a "wrapping" mechanism to generate new keys according to paths or subpaths. However, there are still many theoretical issues pending, such as the processing of recursive queries, the determinability of implication problems, and the complexity of inclusion problems, which deserve further study.

3. Approximate Queries of JSON Data

For traditional database queries, the role of historical query information for future queries is very limited. Common technical means are limited to view selection, adaptive index or buffer technology. If the workload is predictable, the columns or expressions involved in historical queries may provide some clues for future queries, such as which indexes [8,9] or materialized views [10] should be created, or if newly queried tuples still reside in memory, this will buffer future queries involving the same tuples. Obviously, the above methods are only valid for accurate matching queries. To put it further, even for accurate matching queries, if large-scale data is involved, the index or materialized view may still be too large to be further processed in time, or the buffered data is too small for large-scale data.

Therefore, approximate queries are widely used in large-scale data queries to reduce query response time. For example, sampling technology [11], that is, the user explicitly specifies a sampling operation in the query expression, including the types of sampling operation, sampling proportion and various optional parameters, to speed up query process and reduce resources consumption. However, for general queries, the query system using this method cannot guarantee the statistical correctness of query results. On-line aggregation method [12] sampled input consistently based on asymptotic strategy, i.e. updating the results of queries with the increasing number of processed inputs, thereby gradually improving the accuracy of queries. Its advantage is that it can return an approximate query quickly and when the query should end can be decided by users according to the results returned. However, how to integrate aggregation queries into existing data platforms still has a lot of work to consider, such as data access methods in database physical design and the need to implement specific types of operations to support progressive query execution[13].

All the above query strategies do not make good use of the historical query results. Considering the more general case of large-scale data, the CPU and I/O resources consumed by historical queries are wasted. In fact, historical query information often contains useful information that may be used in future queries, which can be used repeatedly for future queries to speed up queries and reduce resource consumption. Therefore, we can build an approximate query model, and gradually learn the data distribution through historical queries and its results, so as to support the new query, and ultimately make the query system response to the new query faster and more accurate.

In summary, for JSON file with large-scale volume, especially in data-intensive cases, accurate querying JSON data are expensive. Therefore, JSON approximate query is more realistic in real situations. But at present, there are few research results in this area. The only relevant research work
such as FAD.js is a JIT-based (Just In Time) runtime system, which can quickly encode and decode JSON data to achieve fast access. However, this research itself does not involve approximate queries at present.

4. Conclusions
This paper presented the problem of approximate query of large-scale JSON data. We researched how to approximate querying large-scale JSON file. We thoroughly pointed out the advantages and disadvantages of various kinds of ideas, theories, and technologies to treat the problem. We also suggested some potential directions to the problems, which may be meaningful to future research of the related problems, including semi-structured data exchange and integrity.

5. Acknowledgments
The paper is partly supported by Science Research Team (kytd201902) and Research Start Project (2015kyqd002) of Anhui Xinhua University and Anhui Province Quality Engineering (2019jxtd119).

6. References
[1] Lv T, Yan P, and He W 2018 Survey on JSON data modelling. Journal of Physics: Conference Series, Vol. 1069, pp.683-687.
[2] Lv T, Yan P, and He W 2019 On Massive JSON Data Model and Schema. Journal of Physics: Conference Series, Vol. 1302: 022031.
[3] Mozafari B, Ramnarayan J, Menon S, et al 2017 Snappydata: A unified cluster for streaming, transactions, and interactive analytics. Conference on Innovative Data Systems Research, Jan.8-11, 2017, California, USA.
[4] Mozafari B 2017 Approximate Query Engines: Commercial Challenges and Research Opportunities. ACM SIGMOD/PODS Conference 2017, pp.521-524.
[5] Ong K, Papakonstantinou Y, and Vernoux R 2014 The SQL++ semi-structured data model and query language: A capabilities survey of sql-on-hadoop, nosql and newsql databases. CoRR, abs/1405.3631.
[6] Pierre B, and Reutter J L 2017. JSON: Data model, Query languages and Schema specification. ACM SIGMOD/PODS Conference 2017, pp.123-135.
[7] Jan H, Paredaens J, and Jan V 2017 J-Logic: Logical Foundations for JSON Querying. ACM SIGMOD/PODS Conference 2017, pp.137-149.
[8] Graefe G and Kuno H 2010 Adaptive indexing for relational keys. International Conference on Data Engineering (ICDE) 2010 Workshops, pp.69-74.
[9] Petraki E, Iredos S, and Manegold S 2015 Holistic Indexing in Main-memory Column-stores. ACM SIGMOD/PODS Conference 2015, pp.1153-1166.
[10] Agrawal S, Chaudhuri S, and Narasayya V. R 2000 Automated Selection of Materialized Views and Indexes in SQL Databases. The 36th International Conference on Very Large Data Bases (VLDB) 2010, pp.496-505.
[11] Armbrust M, Xin R, Lian C et al 2015 Spark SQL: Relational Data Processing in Spark. ACM SIGMOD/PODS Conference 2015, pp.1383-1394.
[12] Chandramoulfi B, Goldstein J, and Quamar A 2013 Scalable progressive analytics on big data in the cloud. VLDB Journal, 6(14):1726-1737.
[13] Li F, Wu B, Yi K, et al 2016 Wander Join: Online Aggregation via Random Walks. ACM SIGMOD/PODS Conference 2016, pp.615-629.