OceanST: A Distributed Analytic System for Large-Scale Spatiotemporal Mobile Broadband Data

Mingxuan Yuan[†], Ke Deng[†], Jia Zeng[‡,†], Yanhua Li[†],
Bing Ni[†], Xiuqiang He[†], Fei Wang[†], Wenyuan Dai[†] and Qiang Yang[†]
[†]Huawei Noah's Ark Lab, Hong Kong
[‡]School of Computer Science and Technology, Soochow University, Suzhou 215006, China
yuan.mingxuan,deng.ke,zeng.jia,Li.Yanhua1@huawei.com

ABSTRACT

With the increasing prevalence of versatile mobile devices and the fast deployment of broadband mobile networks, a huge volume of Mobile Broadband (MBB) data has been generated over time. The MBB data naturally contain rich information of a large number of mobile users, covering a considerable fraction of whole population nowadays, including the mobile applications they are using at different locations and time; the MBB data may present the unprecedentedly large knowledge base of human behavior which has highly recognized commercial and social value. However, the storage, management and analysis of the huge and fast growing volume of MBB data post new and significant challenges to the industrial practitioners and research community. In this demonstration, we present a new, MBB data tailored, distributed analytic system named *OceanST* which has addressed a series of problems and weaknesses of the existing systems, originally designed for more general purpose and capable to handle MBB data to some extent. OceanST is featured by **(i)** efficiently loading of ever-growing MBB data, **(ii)** a bunch of spatiotemporal aggregate queries and basic analysis APIs frequently found in various MBB data application scenarios, and **(iii)** sampling-based approximate solution with provable accuracy bound to cope with huge volume of MBB data. The demonstration will show the advantage of OceanST in a cluster of 5 machines using 3TB data.

1. INTRODUCTION

With the fast deployment of 3G/4G cellular networks and the increasing popularity of mobile devices, it allows people to access the Internet (almost) anytime and anywhere. It is reported that more than 96% Hong Kong citizens use mobile phones to access the Internet every day [2]. When mobile users access the Internet, the system logs, called Mobile Broadband (MBB) data, record rich information including *user id* (i.e., IMSI, International mobile Subscriber Identity), *location* (longitude, latitude), *time-stamp*, *mobile device type*, *mobile App type*, *package size*, and etc. The MBB data may present unprecedentedly large knowledge base of human behavior in terms of the scale of the population covered and the fine-grained spatiotemporal granularity. The MBB data bring great opportunities to build a full angle picture about crowds, which enable a wide range of applications to improve the experience of the mobile users and to help decision making of business/government in a wide range of application scenarios, e.g., location-based marketing, mobile App recommendation, population movement pattern in a city, the home-work location analysis, the public traffic optimization, and etc.

It is highly challenging to handle the MBB data because of the huge volume and the velocity of new data arriving. For example, 5TB MBB raw data is generated every day in November 2013 in Shenzhen, a city in China with 10 million population. According to our experiences working on the MBB data in the past few years, we have identified three desirable features that a large-scale MBB data analytic system must possess.

- Support the distributed data storage architecture to accommodate the huge volume of MBB data which have been collected over time, and the efficiency of loading the ever-growing new MBB data to the storage architecture periodically;
- Support efficient spatiotemporal query processing, in particular, a bunch of spatiotemporal aggregation queries and basic analysis APIs over distributed data storage system, since almost all interesting problems of the MBB data we have observed are spatiotemporal information relevant. Importantly, the practical value of MBB data focuses on the aggregated behaviour of population because querying the information of individual user is usually problematic due to privacy concerns;
- Support sampling-based approximate solution of spatiotemporal aggregation queries and basic analysis APIs, which are much more efficient with theoretically provable error bound, because the exact solution is often very time consuming.

We have investigated the existing distributed systems of spatial/spatiotemporal data management [3, 4, 6, 1]. None of them possesses all of the above three desirable features. The system proposed in [3] is a spatial data warehousing system that provides spatial querying based on Hadoop MapReduce through spatial partitioning. The system proposed in [4] is also based on Hadoop MapReduce with high-level language support for spatial data. However, they cannot properly manage the temporal aspect of MBB data without significant extension. MongoDB [1] is a high-performance NoSQL database with built-in support of spatiotemporal indices. MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. However, the

Proceedings of the VLDB Endowment, Vol. 7, No. 13

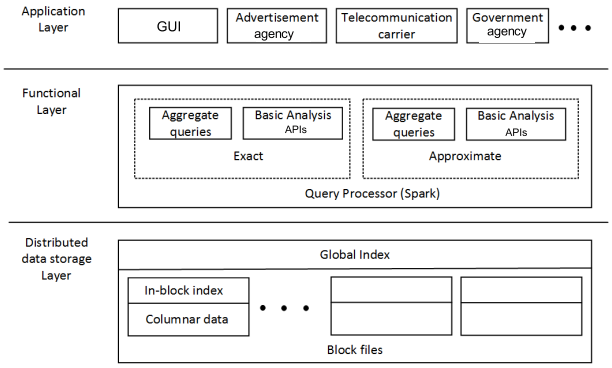

Figure 1: The structure of OceanST.

poor loading efficiency of MongoDB has been observed in both our practice and by the previous study [6]. CloST [6] is a spatiotemporal database based on Hadoop MapReduce. While it directly inherits the proven scalability of Hadoop MapReduce, it also shows much more efficient parallel data loading compared to MongoDB due to the deliberated designed hierarchical partitioning strategy. The weakness of CloST to work as a MBB data analytic system is that it only supports relatively simple spatiotemporal query, i.e., retrieving the records in the spatiotemporal query range, while the MBB data analytic system mainly supports spatiotemporal aggregate query and analysis, i.e., the (exact/approximate) statistics of the records in the spatiotemporal query range.

In contrast, OceanST holds all three desirable features of a reliable MBB data analytic system. First, the high loading efficiency is achieved in OceanST by borrowing the ideas from CloST. Second, OceanST is based on Spark MapReduce [7] which is an in-memory MapReduce solution, i.e., the data in OceanST are processed repeatedly in memory, so that the query results can be obtained much faster than Hadoop MapReduce solution which is disk-based [1]; in particular, the new spatiotemporal index structures have been developed to process exact/approximate spatiotemporal aggregate queries and basic analysis APIs over the distributed storage system. Third, a set of novel approximate solutions based on random sampling have been developed to handle huge volume of MBB data with theoretically provable error bound.

In the demonstration, the advantage of OceanST is illustrated by comparing against CloST [6] and MongoDB [1] on a cluster with 5 machines using the real-world MBB data of 3TB.

2. STRUCTURE OF OCEANST

OceanST has three layers, namely, the distributed data storage layer, the functional layer, and the application layer, as shown in Figure 1.

2.1 Distributed Data Storage Layer

All MBB data records are in the form [*user id*, *time*, *longitude*, *latitude*, *attribute1*, *attribute2*,$\cdots$] where the first four attributes together uniquely identify a MBB data record. The *attribute1*, *attribute2*, $\cdots$ are additional attributes such as *device type*, *mobile App type*, *package type* and etc. All the records and indices are stored as regular files on HDFS. To achieve high loading efficiency, the hierarchical partitioning strategy of CloST is applied. Under this strategy, the MBB data are first divided into a number of level-1 partitions according to hash values of *user id* and coarse ranges of Time. Then, each level-1 partition is divided into a number of level-2 partitions according to a spatial index on location (*longitude*, *latitude*). Finally, each level-2 partition is further divided into a sequence of level-3 partitions according to finer ranges of Time. Level-3 partitions contain actual MBB records and the higher level partitions serve as indexes for level-3 partitions. The level-1 partition is named as a bucket, a level-2 partition as a region, and a level-3 partition as a block which corresponds to a block file of 64MB. Each bucket can independently perform data loading and storage optimizing, we can gradually append data or tune storage structures one bucket after another such that the high loading efficiency is achieved.

While the hierarchical partitioning strategy allows us to parallelize the relatively simple spatiotemporal query in CloST to retrieve records, it is insufficient in OceanST to properly support spatiotemporal aggregate query and analysis of the MBB data, which are mainly interested in the exact/approximate statistics of the retrieved records. Therefore, two additional data structures have been developed in OceanST, namely, in-block index and inverted list.

In-block index. The in-block index aims to refine the spatiotemporal granularity of the index structure. It is insensible to scan the entire block file, if only a fraction of MBB data records in the block file are relevant to the query. For example, the query "count the mobile users whose trajectories are in 100 meters along a specified road in a time period" may involve many block files which are partially relevant to the spatiotemporal query.

In OceanST, the size of each leaf node corresponds to the default memory page (4KB) of the operating system. In block file, the records are grouped by *user id*, and in the same partition the records are sorted on *time*. Clearly, all records in the same group belong to the same user and they are organized in the sequence of time that the records are collected. A B^+-tree is built to index the *user id* by maintaining the offset of the corresponding group in the block file. As a consequence, given an *user id*, the trajectory of the user can be easily retrieved without accessing the trajectories of other users. In addition to the B^+-tree, a 3-dimensional quadtree (i.e., on *time*, *latitude* and *longitude*) is built in block file to index MBB data records. Given a spatiotemporal query range, for example, all *user id*s whose MBB data records are contained by the range are retrieved using the quadtree; then the complete trajectory of these users can be accessed in block files using the B^+-tree.

Inverted Index. OceanST has a mechanism to allow various inverted indices which indicate the index leaf nodes (in the 3-dimensional quadtree) associated with the attribute value of interest. For example, suppose"iPhone" is a value of attribute *mobile App type*; "iPhone" keeps a list of index leaf nodes which contain the MBB data record(s) with "iPhone" in *mobile App type* attribute; in particular, each leaf node in the list can be attached with various aggregated information such as the number of users using "iPhone" in this leaf node. As a result, the aggregate query, e.g., "what is the total number of users using *iPhone* in a spatiotemporal query range?", can be answered efficiently. Another motivation of the inverted index is to support the approximate solution in OceanST which is based on randomly sampling the leaf nodes of MBB data index. Given the sampling budget (i.e., a certain percentage of index leaf nodes to be sampled), the finer the granularity is, the less variance the approximate solutions is [5]. The storage of the inverted index is reasonable since the number of leaf nodes are much smaller than the original MBB data and the number of attribute values requiring inverted index is very limited in practice.

2.2 Functional Layer

[1]Note that both Hadoop MapReduce and Spark MapReduce are on HDFS which is designed to scale to tens of petabytes of storage.

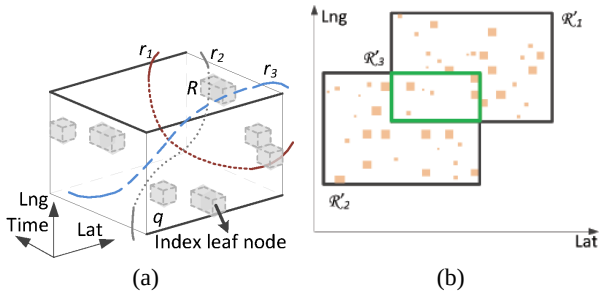

Figure 2: Random sampling based spatiotemporal aggregate query and basic analysis API.

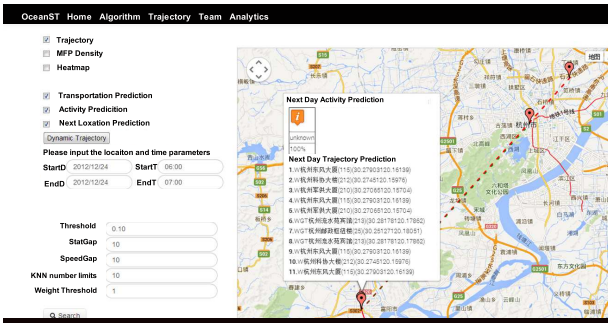

Figure 3: OceanST graphic user interface.

The functional layer consists of a bunch of spatiotemporal aggregate queries and basic analysis APIs which work independently or support various complex analytic tasks found in the application layer. OceanST provides two types of solutions, i.e., *exact* and *approximate*. The approximate solutions are based on randomly sampling the MBB data. The approximate solution is a must because exact solution may involve considerably large volume of MBB data and thus very time consuming. In this case, a theoretically error bounded approximate solution is promising.

2.2.1 *Exact Solution*

The spatiotemporal aggregate queries in OceanST include *count*, *distinct count*, *maximum*, *average*, *minimum*, *sum*, and etc. Given a spatiotemporal query range, for example, *count* returns the number of mobile users who are in the range; *maximum* returns the mobile App which is used by most mobile users in the range; *average* returns the average movement speed of mobile users.

The basic analysis APIs include a number of basic analysis functions. Some of them are: **(i)** given two spatiotemporal ranges, identify the most frequent path, the path distribution and the distribution of transportation tools used by the mobile users travelling between them; **(ii)** given the current location, a temporal range and a mobile user, predict the next locations of the user and the next mobile APP to be used by the user with a probability by mining the historical data; **(iii)** given a spatiotemporal range, identify the top-k locations from which the mobile users come from; **(iv)** given a spatiotemporal range, identify the top-k locations to which the mobile users go; **(v)** given a spatial region, partition it into subregions according to the density of mobile users at different time slots of a day, e,g,. hourly; **(vi)** find out the home and work locations; and more.

2.2.2 *Approximate Solution*

The spatiotemporal aggregate query and basic analysis API processing with large volume of disk-resident MBB data take very long time to produce exact answers. Hence, the approximate solution is promising in many application scenarios which have stringent response time requirements and accept approximate result with theoretical error bound.

By utilizing the well-established spatiotemporal index and a specific inverted list to trajectory data, we have developed random index sampling (RIS) algorithm in OceanST to estimate the answer with a guaranteed error bound. An example is shown in Figure 2(a) where r_1, r_2, r_3 are three trajectories belonging to different mobile users, R is the spatiotemporal query range of query q, and the small gray blocks are leaf nodes of the MBB data index in a block file. Now, we use spatiotemporal *distinct count* query as an example to show how to estimate the answer. Let B denote the sampling budget, i.e., the maximum number of index leaf nodes allowed to collect. In our analysis, we assume that B is always sufficiently large. We uniformly at random pick up B index leaf nodes from the leaf node set covered by the spatiotemporal query range with replacement. In OceanST, we have developed an asymptotically unbiased estimator to the *distinct count* query and proved the relationship between the accuracy and sampling budget. An estimator is a function of a sequence of observations that outputs an estimate of an unknown population parameter.

The proposed RIS algorithm properly deals with a single spatiotemporal aggregate query with a guaranteed estimation error bound. However, when it comes to OceanST, a large-scale MBB data analytic system, a large number of spatiotemporal aggregate queries may come concurrently. These queries may have overlapped spatiotemporal query ranges. As illustrated in Figure 2(b) where R'_1 and R'_2 are the spatiotemporal query ranges of two queries which arrive concurrently and overlap with each other. A naive method for handling concurrent queries is to perform single random index sampling (RIS) algorithm individually for each query. However, by reusing the samples obtained in the overlapped range of queries, the estimation accuracy can be significantly improved. In OceanST, we propose concurrent random index sampling (CRIS) algorithm, that performs stratified sampling and overlapping sample reuse on concurrent spatiotemporal aggregate queries. As illustrated in Figure 2(b), the small blocks are samples used in the estimation. Our theoretical results show that CRIS can achieve higher estimation accuracy for each concurrent trajectory aggregate query, with less sampling budget than simply running RIS for each query.

Our extensive evaluation results indicate that both RIS and CRIS outperform exhaustive search for single and concurrent spatiotemporal aggregate queries by two orders of magnitude in terms of the query processing time and they guarantee the answer accuracy of 0.003 to 0.2 in normalized mean square error.

2.3 Application Layer

The application layer incorporates the graphic user interface (GUI) and the complex analytic tasks. The GUI provides an effective way to directly communicate with the queries and APIs in the functional layer. As shown in Figure 3, the temporal query range (and other parameters) are input using the text fields in the left side of the interface; the spatial range is specified on the map by clicking and dragging mouse; the output is visualized. One more example is shown in Figure 4; we select a rectangle in the middle of Figure 4(a) and input a temporal range; the results of some spatiotemporal aggregate queries are visualized as shown in Figure 4(b)-(e).

Using the queries and APIs in the functional layer as the building blocks, various complex analytic tasks can be built. Let us use the business scenarios of outdoor advertising as example: **(i)** An advertising agency wants to know the number of peoples who are covered by an outdoor advertising panel (i.e., the peoples passing

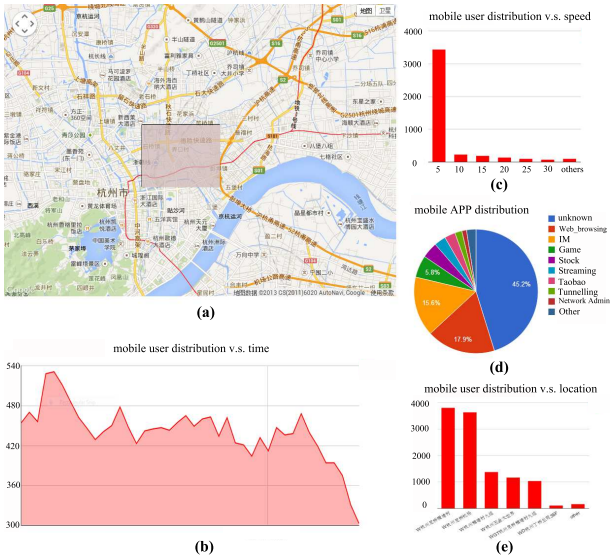

Figure 4: An example of spatiotemporal aggregate query. (a) the spatial query range, (b) the distribution of mobile users, (c) the speed distribution of mobile users, (d) the mobile App distribution, (e) the distribution of mobile users at different locations

the effective region of the panel) in a certain time period, such information is the basis to set the price on the panel; **(ii)** given a customer whose budget is $\$50,000$, which panels from many options can cover the maximum number of (distinct) peoples and the total cost is under the budget; **(iii)** the advertising agency owns a large number of outdoor panels of different prices and many customers of different budgets, the problem is how to assign the outdoor panels to customers so as to maximize the benefit of the advertising agency. In such complex analytic tasks, the *count* or *distinct count* query of OceanST provide the fundamental information, i.e., the number of mobile users covered by a spatiotemporal range, since mobile users are random sample of the entire population. Other examples of complex analytic tasks can be found in government agency in the traffic congestion analysis, the movement pattern of population in a city, the event recognition, and etc. In such tasks, the basic analysis APIs in the functional layer such as the path distribution between locations, the prediction of the next locations and the distribution of population are closely relevant.

3. DEMONSTRATION

The demonstration will be conducted on a cluster with 5 machines. Each machine has 24 six-cores Intel X5670 2.93GHz processors and 94GB memory. All machines run on Suse Linux Enterprise Server 11. Spark-0.7.3 is selected as the running system. Around 3TB MBB data are used in the demonstration. We also configure CloST and MongoDB on the three machines for comparison. This demonstration consists of three phases.

In the first phase, we will show the structure of OceanST and explain the rationale behind the advantage against the existing systems, i.e., CloST and MongoDB. To help deliver the ideas, a number of diagrams will be presented with animated effects to show the input/output and the processing flows.

In the second phase, we will demonstrate spatiotemporal aggregate query and the basic analysis API processing. In particular, this phase will demonstrate both the exact solution and the approximate solution of OceanST. Two typical scenarios will be presented. The first scenario is to process *distinct user count* query, given a spatiotemporal query range; the second scenario is to process *attribute distribution* query which returns requested attribute's (for example, mobile APP usage) distribution in a spatiotemporal query range. In the approximate solution, the balance of accuracy and the response time will be demonstrated in the same scenarios. The demonstration will report the query response time when the spatiotemporal query range varies, in specific, 10 temporal ranges and 10 spatial ranges are used. All temporal ranges have the same starting time and all spatial ranges are rectangle bounding boxes with the same centroid.

The third phase will demonstrate the performance of OceanST and its advantage against CloST and MongoDB. We focus on two important performance indicators for the large MBB data analytic system: data loading time and query/API response time. The data loading time of OceanST is consumed for loading data into OceanST block files and for building index. For example, it takes about 10 minutes for building index and takes about 40 minutes for writing blocks in OceanST and CloST to load 147GB MBB data; MongoDB failed to load this data within an acceptable time (cannot finish after 5 days). In our experience, MongoDB takes 40 hours to load 3GB data and shuffling them across the three sharded machines. The demonstration, therefore, only compares the performance between OceanST and CloST on processing the queries in the three scenarios in the second phase. In all settings of spatiotemporal query range, OceanST returns the exact solutions at least 8 times faster than CloST, the sampling based approximate solution achieves another 10 times acceleration with high accuracy.

4. ACKNOWLEDGEMENTS

This work is supported by National Grant Fundamental Research (973 Program) of China under Grant 2014CB340304, NSFC (Grant No. 61373092 and 61033013), Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant No. 12KJA520004), and Innovative Research Team in Soochow University (Grant No. SDT2012B02).

5. REFERENCES

[1] Mongodb. www.mongodb.org.

[2] Sin chew daily. http://news.sinchew.com.my/node/318251.

[3] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop gis: A high performance spatial data warehousing system over mapreduce. In *Proceedings of VLDB Endowment*, pages 1009–1020, 2013.

[4] A. Eldawy and M. F. Mokbel. A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. In *Proceedings of VLDB Endowment*, pages 1230–1233, 2013.

[5] Y. Li, M. Steiner, J. Bao, L. Wang, and T. Zhu. Region sampling and estimation of geosocial data with dynamic range calibration. In *Proceedings of ICDE*, pages 1–12. IEEE, 2014.

[6] H. Tan, W. Luo, and L. M. Ni. Clost: a hadoop-based storage system for big spatio-temporal data analytics. In *Proceedings of CIKM*, pages 2139–2143. ACM, 2012.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.