Massive AIS Data Management Based on HBase and Spark

Jiwei Qin, Liangli Ma, Jinghua Niu
School of Electronic Engineering,
Naval University of Engineering,
Wuhan, China
e-mail: 851293113@qq.com, maliangl@163.com, 517887057@qq.com

***Abstract*—With the popularization of AIS technology on ships, the scale of maritime traffic data are rapidly increasing. The traditional management system of AIS data based on relational databases is faced with difficulties in expansion and low access efficiency. In view of that, we present AISHS, a distributed system based HBase and Spark to offer efficient storage and near-real-time query to massive AIS Data. In order to avoid the communication overhead caused by regroup the trajectory in query process, AISHS stores all data of a ship in one Region of HBase, constructs a global secondary index by the spatial-temporal attributes of AIS data. Based on this, we implement the co-location between HBase Regions and Spark RDD partitions to ensure efficient spatial-temporal query. Experiments on real datasets show that AISHS is suitable for the storage and query of AIS data.**

Keywords-AIS data; HBase; spark; spatial-temporal query

I. INTRODUCTION

Thanks to the rapid development of maritime economy, more and more ships navigate in global oceanic area. In order to monitor the navigational dynamic information in real time, Automatic Identification System (AIS) emerged as the safety navigation of ships [1]. The AIS data are sent periodically during voyage, which includes coordinates, time and other attributes of ships. With the increasing number of ships equipped with AIS, There has been an explosion in the amount of AIS data, the performance of traditional AIS data manage system based on RDBMS decreased significantly. The efficiency of AIS data manage system not only affects the function exertion of AIS, but also restricts the development of various applications based on AIS.

With the rapid development of distributed computing, the rise of technologies such as Hadoop, Spark, and NoSQL has made it possible to access mass data effectively. Compared with traditional scheme, the distributed solutions can easily use cluster resources to satisfy the needs of mass data management for computing and storage, and they have excellent features such as ease of maintenance, manageability, and scalability. Therefore, it gradually becomes a new tendency that mass data are managed based on a distributed solution [2]. However, the traditional distributed environment is not optimized for the data characteristics of AIS. In other words, it cannot handle efficiently the storage and query of AIS data.

AIS data are a typical trajectory data, the current research about trajectory data management mainly focuses on the designs of storage mode and spatial-temporal index.

VREEDE et al. [3] propose a historical AIS data schema based on MongoDB. It uses four B+Tree on individual attributes and a 4D Morton code based on latitude, longitude, MMSI and date-time to index AIS data, and supports trajectory query and range query.

Jie Bao et al. [4] design a trajectory data management system on Microsoft Azure. This system optimizes the storage schema, index method and query strategy based on different functionalities to guarantee the efficient trajectory updates and perform the services in real-time.

Tiantian Liu et al. [5] propose HTLA based on HBase in which AIS data are stored in two tables to satisfy the requirement for different queries. A table which uses MMSI and time as the rowkey is designed to support trajectory query. In the other table, GeoHash is used to linearize the data to enhance the performance of range query.

Zhigang Zhang et al. [6] propose a distributed in-memory system TrajSpark, which offers efficient management for mass trajectory data. It uses IndexTRDD [7] to compress and store trajectory segments, and combines global index and local index to implement efficient parallel queries.

The researches store the trajectory data of a single object dispersedly in different nodes. A major overhead in above methods is that the dispersed data need to be regrouped into a trajectory by data shuffling during query execution. It may involve a heavy data transmission cost and cause problem of poor query response ability.

Inspired by above observations, we design and implement our AIS data management system called AISHS. To avoid regrouping operations among nodes, AISHS stores the data collected of a single ship in a single Region of HBase. For the complex spatial-temporal query, a global secondary index is constructed based on the multiple attributes of AIS data. Furthermore, we optimize the parallel query method based on Spark, it avoids data shuffling by co-locating RDD partitions and HBase Regions, and ensures the localization processing in query. Our main contributions can be summarized as follows:

1) We propose a new data model of AIS data, which stores AIS data uniformly among nodes and constrains the data collected of a single ship in a Region, thus laying the foundation for data locality in query process.

978-1-5386-7226-6/18/$31.00 ©2018 IEEE

2) We introduce an efficient global secondary indexing scheme, and combine with HBase to achieve efficient query for AIS data;

3) We optimize the parallel spatial-temporal query algorithm based on Spark. This optimization strategy reduces the communication overhead in distributed environment by utilizing co-located RDDs, to guarantee the high efficiency and near real-time performance of query.

4) We evaluate our system design with the real AIS datasets, the experimental result verify the efficiency of our system.

Sect. 2 introduces the preliminaries about the AIS data, the basics in HBase and Spark, and provides the overview of our system. The three main modules: (1) Data Storage Module, (2) Secondary Index Module, and (3) Query Processing Module are described in Sect. 3, Sect. 4 and Sect. 5, respectively. Sect. 6 provides an experimental study of our system. Finally, we give a brief conclusion in Sect. 7.

II. PRELIMINARY

A. AIS Data

AIS uses VHF channels (Very High Frequency) to broadcast dynamic information and static information of ships in their navigation area so that nearby ships or shore-base installations can timely acquire information in nearby sea area, which helps to avoid collision and improve navigational safety.

The AIS messages can be divided into 27 categories, involving location, ship, confirmation, addressing and broadcasting. Among them, location-based messages play an important role in ship collision avoidance, the study of ship movement behavior, etc. [8]. Therefore, this paper take the location-based AIS data as study object.

As for related applications, AIS data contain useful attributes including MMSI (Maritime Mobile Service Identity), transmitting time, location and so on. A MMSI is a series of nine digits which are sent in digital form in order to uniquely identify ship stations, ship earth stations, etc. Location and transmitting time reflect the spatial-temporal information of AIS data. The data types of AIS are defined:

Definition 1. A ship trajectory point p is an location-based AIS message denoted as <*mmsi*, *l*, *ts*, *o*>, where *mmsi* is the MMSI of ship and acts as the ship identifier, *l* is the location information, *ts* is the send time, and *o* is the other information which include the speed, heading and so on.

Definition 2. A ship trajectory T contains a sequence of ship trajectory points (p_1, …, p_m) ordered by their timestamp, and all points of T are generated by the same ship.

B. HBase

HBase [9] is an open source distributed key-value database. The technology is derived from Google's Bigtable concept. It dispersedly stores the data of a table over a cluster of nodes, in order to provide scalable storage to massive data.

Different from traditional relational data model, data in HBase is stored in tables, and each data cell is indexed by rowkey, column family, column qualifier and time stamp. Among them, rowkey is the unique index for directly accessing data, it is closely related to data storage and query efficiency, and plays a very important role in data processing.

On the physical structure, HBase uses Region as the minimum unit for distributed storage. An HBase table is usually divided horizontally into multiple Regions scattered in each work nodes (named as RegionServer), each Region stores a certain range of data rows. A Region is decided to be split when store file size goes above threshold or according to defined region split policy, in order to make the data distribution of Regions more uniform. Split operation takes a row as the split point, and the rowkey range of the split Region changes after splitting [9].

C. Spark

Spark [10] is an open-source cluster-computing framework. It uses resilient distributed dataset (RDD) as its architectural foundation, a read-only multiset of data items distributed over a cluster of nodes. In response to limitations of MapReduce, RDD offers a restricted form distributed shared memory for distributed programs. Therefore, the computing speed of Spark has a greatly improvement compare to Mapreduce.

The users can use the RDD to perform parallel computing through a series of transformation operators and action operators. Similar to HBase, Spark also manages distributed datasets based on partition. A reasonable Spark partition strategy can effectively improve computing performance.

D. System Overview

Fig. 1 gives a full picture of our AISHS. It is composed of three modules:

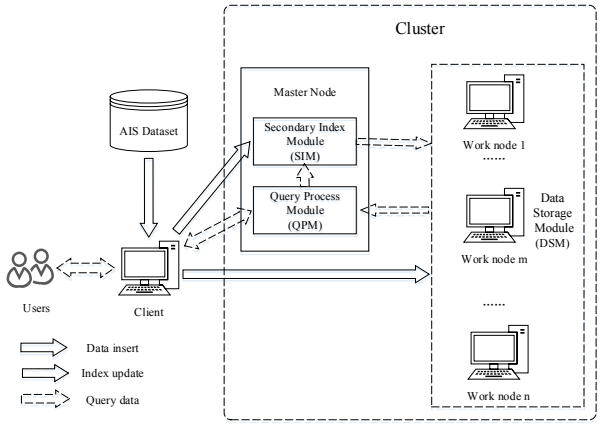

Figure 1. Structure of AISHS.

1) Data Storage Module (DSM). This module built on top of HBase is for the persistent storage of AIS data. It stores all data of a single ship in one Region to avoid data dependence among nodes. This module is detail in sect. 3.

2) Secondary Index Module (SIM). In order to optimize spatial-temporal queries, a three-level hybrid index are employed to index the spatial-temporal attribute of AIS data in this module (detailed in sect. 4).

3) Query Processing Module (QPM). The implementation of two typical spatial-temporal queries is

introduced in this module. We give a detailed description in Sect. 5.

III. DATA STORAGE MODULE

A. Table Schema

Data Storage Module (DSM) uses an HBase table to persistently store all AIS data. Each row of DSM stores a trajectory segment which contains the ship trajectory points generated in a time interval (The length of all time intervals is equal). A DSM rowkey is shown below in (1).

$$rowkey = in(mmsi) + t_{ts} \quad (1)$$

where *mmsi* is the MMSI of a ship, and t_{ts} is the start time of a time interval and act as the identifier of a time interval. The *in*() function is used to inverse *mmsi*. Because the initial digits of MMSI have a certain distribution pattern, it is difficult to guarantee the uniform distribution of rowkeys, resulting in a load imbalance among nodes. The final digits of MMSI show a certain randomness, so *in*(*mmsi*) ensure that the rowkeys are randomly distributed in each Region.

While HBase does not do well with multiple column families. One column family named *AIS* is assigned to store All AIS data. The column qualifier is represented by the transmitting time of ship trajectory point in modulo the start time of trajectory segment, in order to short the length of column qualifier. Location information and other information of ship trajectory point are stored as data cell.

Based on the dictionary order of HBase, the ship trajectory points of a single ship will be stored in ascending order of time. Therefore, if the query rowkey set are orderly, and there is no need to sort the query results. In addition, compare to point, the storage form by segment reduces the space overhead of secondary index.

B. Region Split

HBase table only creates single Region by default. Single Region scenario causes data to be inserted centrally, resulting in Region hotspots and increasing the number of Region split, it will affect the insert performance. In order to avoid Region hotspots and reduce the number of Region split, we divide the DSM into multiple Regions by pre-splitting when DSM is initialized. During the pre-splitting, the rowkey range of each Region is determined by dividing the range of *in(mmsi)*, to ensure that all data of a single ship is stored in one Region. So at the beginning, the AIS data will be directly distributed into the corresponding Regions based on their MMSI and Regions will not split. Until the store file size goes above threshold, the Region needs to splits again.

In order to avoid spreading the data of a single ship to different Regions after Region split, we use the prefix split policy that splits rowkey range by a *in(mmsi)* value. This ensures that rows with the same prefix locate in the same Region after the split, i.e. all data of a single trajectory is still stored in a Region.

C. Data Insert

During the insert process, tens of thousands ship trajectory points per second need to be stored into DSM. If we handle the insert process by point, which will affect the insert efficiency undoubtedly. In order to improve the insert performance, we first preprocess the ship trajectory points into trajectory segments, and then write them in the HBase buffer. If the volume of HBase buffer exceeds the predefined value, HBase calls Put interface to store buffer data into DSM. By using HBase buffer, AIS data can be inserted into DSM in a more efficient manner, thereby reducing the write times and improving the insert efficiency. Fig. 2 displays the insert process of AIS Data, where T_1 and T_2 are two ship trajectories, and $p_{1,1}$ to $p_{1,5}$ and $p_{2,1}$ to $p_{2,6}$ are the ship trajectory points of T_1 and those of T_2 respectively.

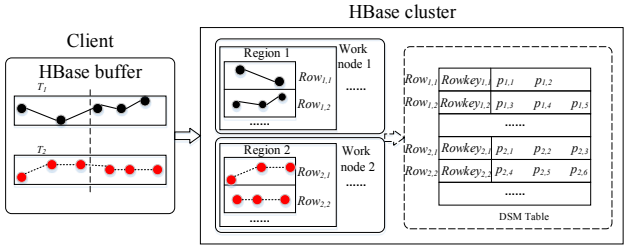

Figure 2. Insert Process.

IV. SECONDARY INDEX MODULE

In HBase, the rowkey provides the only access function as a primary index, and a query for the attributes which are not rowkey structures means a full table scan. Because DSM rowkey structure doesn't include spatial attributes, it is difficult to satisfy the complex spatial-temporal query requirements by using only rowkey. Therefore, we build a Secondary Index Module (SIM) that uses the spatial-temporal attribute of AIS data to index DSM rowkeys to support complex spatial-temporal query.

A. Structure

As shown in Fig. 3, SIM is a three-level hybrid index stored in the memory of master node. The level-1 is a coarse time ranges index based on time series. Each time range corresponds to a level-2 spatial index which is divided by grid. In order to index AIS data in the same grid cell, a level-3 spatial-temporal R-Tree is used to index trajectory segment.

The concrete design of SIM is as follows: First, the time dimension is divided into multiple coarse time ranges in accordance with time series, and a B+ tree is used to organize the time ranges. In order to avoid the time intervals of trajectory segment spanning the boundary of time ranges, we set the length of a coarse time range as multiple times of a time interval and align boundaries between the two during the division. Second, a Hilbert space curve index is used to encode the space grid cells. Finally, in each grid cell, a corresponding spatial-temporal R-tree is constructed by trajectory segments. The index entry of R-tree can be expressed as $rte=<mbr_{ts}, rk_{ts}>$, where mbr_{ts} and rk_{ts} are the spatial-temporal Minimal Bounding Rectangle (MBR) and the rowkey of a trajectory segment, respectively. If the MBR of a trajectory segment spans multiple grid cells, SIM indexes it in each R-tree corresponding to the intersecting grid cell. Because the R-tree only needs to index the spatial-

temporal attribute of trajectory segments, and no needs to index the raw AIS data, the update operation will be more efficient.

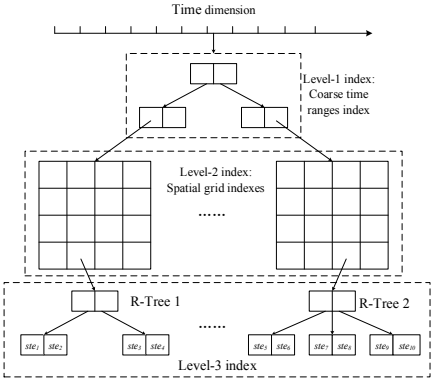

Figure 3. Structure of SIM.

B. Update

The update of SIM and the data insertion of DSM are carried out simultaneously. In order to reduce the update frequency, the update of SIM is processed by trajectory segment. Because SIM only indexes the spatial-temporal attributes of AIS data, in order to reduce the load of network, we only transfer the spatial-temporal attributes to master node for updating SIM, to ensure the update efficiency.

Since the memory capacity of master node is limited, and the storage requirement of SIM is proportional to the size of inserted AIS dataset. When the size of AIS dataset is large enough, the storage requirement of SIM may exceed the memory capacity of master node. As for this point, we persist the newly generated index entries into the file system regularly, and use the Least Recently Used (LRU) policy to replace SIM data which have been cached in memory.

C. Query

In the query process, the level-0 index and level-1 index are used for coarse-grained query, and a set of level-3 R-tree is obtained. Because the level-3 R-trees are mutually independent, multi-threaded parallel query methods can be used to simultaneously access multiple R-trees to speed the query. In the parallel query process, each thread first sorts the internal query results, and then we aggregate the sorted query results of each thread through merge-sorting into a sorted rowkey set for querying DSM.

V. QUERY PROCESSING MODULE

On the basis of DSM and SIM, the Query Processing Module (QPM) supports multiple types of query, such as single object query, spatial-temporal range query, K-NN query and so on. This section selects spatial-temporal range query and K-NN query for detailed description.

A. Spatial-Temporal Range Query

A Spatial-temporal range query retrieves ship trajectories Ts_{st} according to a time range tr_{st} and a spatial range sr_{st}. This query usually involves multiple AIS trajectories stored in different nodes. To improve query efficiency, we builds a parallel query algorithm based on Spark. In the query process, we first use SIM to obtain the rowkeys within the spatial-temporal range, and then query DSM in parallel.

The processing flow is shown in Algorithm 1. At first, we query SIM to get an ordered rowkey set of candidate trajectory, denoted as rks_{st} (line 1). Then, we use rks_{st} to issue a parallel query to DSM, and filter the query results by tr_{st} and sr_{st} to generate the RDD of ship trajectory points defined as $apsrdd_{st}$ (line 2). After that, $apsrdd_{st}$ is regrouped by *in(mmsi)* to create the RDD of ship trajectories $Tsrdd_{st}$ (line 3). Finally, we aggregate the elements of $Tsrdd_{st}$ from cluster to get the query result Ts_{st} and return to the client (line 3-4).

It needs to be pointed out that since all data of a single ship is already stored in one region of DSM, when query is processed based on Spark, we can avoid data shuffling by partitioning RDDs based on co-location, thereby reducing communication cost and improving query efficiency. The co-located partitioning strategy of RDD is as follows: We build a RDD partitioner based on the rowkey ranges of DSM Regions, partition all RDDs through this partitioner, and set the preferred locations of RDDs to the locations of input data. Since the partition setup does not change during the query, and there is no data dependency among nodes, the data shuffling does not occur. The partition setup for RDD is implemented by partitionBy() interface (line 4-5), where *dsmp* is the partitioner which is built based on the rowkey ranges of DSM Regions.

Algorithm 1. Spatial-temporal range query

Input: tr_{st}, sr_{st}
Output: Ts_{st}
1. rks_{st}←SIM.getRowkeysByST(tr_{st}, sr_{st});
2. $apsrdd_{st}$←TSM.bulkGet(rks_{st}).filter(tr_{st}, sr_{st}).partitionBy(*dsmp*);
3. $Tsrdd_{hst}$←$psrdd_{st}$. aggregateByKey (*in(mmsi)*). partitionBy(*dsmp*);
4. Ts_{st}←$Tsrdd_{hst}$.collect();
5. return Ts_{st};

B. K-NN Query

There are many types of K-NN queries. We focus on querying the most similar k trajectories. The similarity metrics such as DTW and LCSS [6] are supported in our algorithm. The K-NN query is defined as follows: Given a time range tr_k and a AIS trajectory T_k, according to the selected similarity metric, the top-k trajectories (represented as Ts_k) are queried which are most similar to T_k within tr_k.

Because the similar trajectories are spatial-temporally close to T_k, SIM can be used to facilitate the pruning of candidate. In the query process of SIM, if there is an intersection relation between the spatial-temporal MBR of a trajectory segment and the spatial-temporal MBR of T_k, we consider the AIS trajectory corresponding to the intersect trajectory segment as a candidate trajectory. After pruning of candidate trajectories, we implement the K-NN query algorithm based on Spark, and co-locate the RDD partitions with DSM Regions to avoid data shuffling. Since the spatial-temporal query has introduced the co-located partitioning strategy, it will not be detailed here.

Algorithm 2 introduces the detailed steps. First of all, we get the spatial-temporal MBR of T_k, defined as mbr_k (line 1). Secondly, we query SIM by mbr_k to get the candidate

trajectory set CTS_k. If the number of elements in CTS_k is smaller than k, we expand the spatial range of mbr_k (the time range does not expand), query SIM to generate CTS_k again, and repeat the process until the number of elements in CTS_k is not less than k (line 2-6). And then the ordered rowkey set rks_k is generated by CTS_k and tr_k (line 7). Next, we initiate a parallel query to DSM by rks_k, filter and regroup the query results of DSM to generate the RDD of ship trajectories, denoted as $Tsrdd_k$. (line 8-9). Finally, we calculate the similarity between T_k and each element in $Tsrdd_k$, collect the similarity results, and choose the k most similar trajectories as the query results (line 10-12).

Algorithm 2. K-NN query

Input: tr_k, T_k
Output: Ts_k
1. mbr_k←getMBR(T_k);
2. CTS_k←SIM.getTrajsByST(mbr_k);
3. while num(CTS_k)<k
4. mbr_k←expand(mbr_k);
5. CTS_k←SIM.getTrajsByST (mbr_k);
6. endwhile
7. rks_k←getRowkeys(CTS_k, tr_{st});
8. $apsrdd_k$←DSM.bulkGet(rks_k).filter(tr_k).partitionBy($dsmp$);
9. $Tsrdd_k$←$apsrdd_{st}$. aggregateByKey ($in(mmsi)$). partitionBy($dsmp$);
10. $simsrdd_k$←$Tsrdd_k$.mapValues(getSimilarities(T_k));
11. Ts_k←$simsrdd_k$.collect().top(k);
12. return Ts_k;

VI. EXPERIMENT

All experiments were conducted in a cluster consisting of 5 nodes, with one master node and 4 data nodes. Each node runs Ubuntu 16.04 LTS on 6 cores Intel Xeon E5-2620 V2 CPU, 16GB memory and 2TB disk, nodes are connected through a Gigabit Ethernet switch. AISHS is implemented based on Hadoop 2.7.3, HBase 1.3.0 and Spark 2.1.0.

The experiment uses the global AIS data from March 1 to March 16, 2012 as the dataset. The dataset contains 320 million AIS data sent by 191,804 ships and is about 40,120,628KB in size. The AIS messages in the dataset have been decoded and sorted in ascending order of time.

During the experiment, the HBase buffer size is set to 6MB, the time interval of trajectory segment is 2 hours, the coarse time range of SIM is 24 hours, and the spatial grid of SIM is divided by 0.5°. We test the insert efficiency and query performance. Because AIS data in dataset are sorted in ascending order of time, the time range of dataset increases with the increase of data scale.

We compare to the performance of AISHS with HLTA and the AIS data management system in [3] (AISMongo for short). Because HTLA and AISMongo do not support K-NN query by default, we implement the K-NN query algorithm of HTLA and AISMongo based on Spark, the process is as follows: First, the candidate trajectories are screened by GeoHash or 4D Morton code. Secondly, the ship trajectory points of the candidate trajectories are obtained by querying the data table. Then, the ship trajectory points are merged into complete trajectories and the trajectory similarities are calculated based on Spark. Finally, the k most similar trajectories are output.

A. Performance of Data Insertion

First, we study the performance of data insertion. Fig. 4 presents the insertion performance of different systems when the data size changes from 40 million to 320 million.

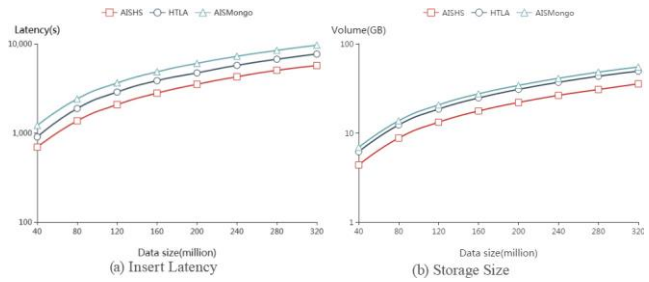

Figure 4. Time and storage overhead of data insert.

Fig. 4 (a) is the comparison of insert latency. The experimental results show that, the insert latency of AISHS is lower than that of HTLA and AISMongo. This is because AISHS inserts data based on HBase buffer, which can effectively reduce the number of insert requests. At the same time, the update of SIM is simultaneous with the data insertion, avoiding extra time cost. In addition, HTLA needs to insert data into two different tables, resulting in extra time cost. AISMongo needs to create 5 additional B+ tree indexes, so AISMongo takes the most time.

Fig. 4 (b) shows the storage size of different systems (including AIS data and secondary index). The SIM of AISHS indexes AIS data by trajectory segment, and takes up less storage space, so the storage size of AISHS and the raw dataset are nearly equal. HTLA needs to organize AIS data in different forms of two tables so that it consumes more storage space. AISMongo indexes each AIS message in 5 B+ trees, and it takes up the most storage cost.

B. Performance of Spatial-Temporal Range Query

Subsequently, we test the spatial-temporal range query. Firstly, we randomly select 100 spatial-temporal ranges as the query conditions when the amount of dataset are increased from 40 million to 320 million. The spatial range is 1%, and the time range is 50%, the average latency of the query are shown in Fig. 5.

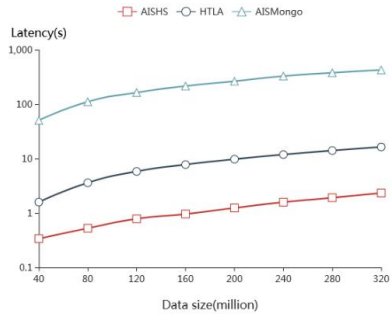

Figure 5. Time cost of single spatial-temporal range query

From the experimental results, it can be seen that the query latency of AISMongo is much higher than the other two systems. This is because the hash distribution of AIS

data greatly increases the seek time of disk scheduling, and the data shuffling also incurs extra time cost. Compared to AISMongo, HTLA stores AIS data based on spatial-temporal distribution. It makes full use of sequential read to ensure the efficiency. However, HLTA also need to regroup trajectory by data shuffling, and there is no index for time attribute, the query result is not ideal. Among the three systems, AISHS has the highest query performance due to the following reasons: (1) The parallel query for SIM ensures that the rowkeys can be obtained quickly; (2) The sequence distribution of all data of a single ship ensures the high hits of HBase block cache; (3) The co-location between HBase Regions and Spark partitions avoids the data shuffling and saves a lot of communication and I/O overhead.

C. *Performance of K-NN Query*

In the K-NN query experiment, we first evaluate the impact of different dataset sizes on the query performance. We randomly select 100 trajectories as query reference, and set the value of *k* and the query time range to 5 and 24 hours respectively. As shown in Fig. 6 (a), AISHS and AISMongo are less affected by the increase of dataset size, while the query latency of HTLA grows with the increase of data size. This is because AISHS and AISMongo use both spatial and temporal attributes to screen the candidate trajectories. However, HTLA can only prune data based on spatial attribute resulting in the overhead of time filtering, and the expansion of dataset size leads to the growth of filtering overhead. In addition, AISHS does not require data shuffling, so the query efficiency is much higher than other two systems.

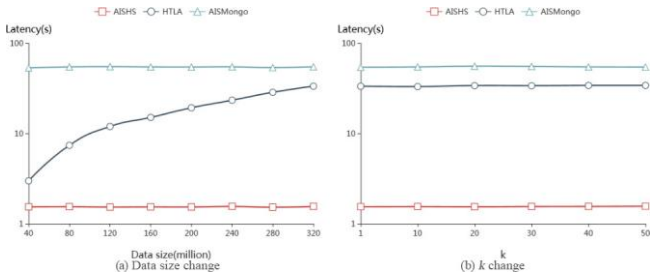

Figure 6. Time cost of K-NN query

Next, we test the impact of different k values on queries. The dataset size is 320 million, and the time range is still 24 hours. The experimental results are shown in Fig. 6 (b), the query delay of the three systems is not affected by the change of k value, This is because when using the SIM or GeoHash or 4D Morton code to obtain the candidate trajectories, the number of trajectories included is greater than the maximum value of *k*, so the query delay will not change substantially.

VII. CONCLUSION

This paper presents AISHS, an AIS data management system based on HBase and Spark. It is mainly used to support efficient storage and near real-time query to the massive historical AIS data. We design and implement the data storage module in HBase based on data locality, which stores all data of a single ship in time sequence and distributes it in the one Region of HBase. Also, we utilize the spatial-temporal attributes of AIS data to implement secondary index module in order to support the spatial-temporal query efficiently. On the basis of these, AISHS avoids data shuffling in the parallel query by co-locating RDD partitions and HBase Regions. We validate the storage overhead and query latency of AISHS by experiments on real AIS dataset. Experimental result show that AISHS outperforms HTLA and AISMongo in terms of insert efficiency and query performance. For future work, we plan to support the index model which combines the local index and global index to make full use of the distributed environment.

ACKNOWLEDGMENT

This work is supported by the Research Development Fund of Naval University of Engineering (Grant no. 20161633). We appreciate the comments from the annoymous reviewers.

REFERENCES

[1] Salmon Loic, and Cyril Ray, "Design principles of a stream-based framework for mobility analysis," GeoInformatica, vol. 21(2), Apr. 2017, pp. 237-261.

[2] Xie X., Mei B., Chen J., Du X., and Jensen C. S., "Elite: an elastic infrastructure for big spatiotemporal trajectories," The VLDB Journal, vol. 25(4), Aug. 2016, pp. 473-493.

[3] De Vreede Irene, "Managing historic Automatic Identification System data by using a proper Database Management System structure," Delft, NED: Delft University of Technology, 2016.

[4] Bao J., Li R., Yi X., and Zheng Y, "Managing massive trajectories on the cloud," In Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, Oct. 2016, pp. 41:1-41:10.

[5] Liu T., Gao S., Chu X., and Lu C., "Storing and querying AIS data in HBase," 2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), IEEE, Jun. 2017, pp. 88-92.

[6] Zhang Z., Jin C., Mao J., Yang X., and Zhou, A., "TrajSpark: A Scalable and Efficient In-Memory Management System for Big Trajectory Data," Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, Springer, Cham, Jul. 2017, pp.11-26.

[7] Xie D., Li F., Yao B., Li G., Zhou L., and Guo M., "Simba: Efficient in-memory spatial analytics," In Proceedings of the 2016 International Conference on Management of Data, ACM, Jun. 2016, pp. 1071-1085.

[8] Harati-Mokhtari A., Wall A., Brooks P., and Wang, J, "Automatic Identification System (AIS): data reliability and human error implications," The Journal of Navigation, vol. 60(3), Sep. 2007, pp.373-389.

[9] Vora, and Mehul Nalin, "Hadoop-HBase for large-scale data," 2011 international conference on Computer science and network technology (ICCSNT), IEEE, Dec. 2011, pp. 601-605.

[10] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I., "Spark: Cluster computing with working sets," In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, USENIX Association Berkeley, Jun. 2010, pp. 10-10.