SharkDB:An In-Memory Storage System for Massive Trajectory Data

Haozhou Wang, Kai Zheng, Xiaofang Zhou, Shazia Sadiq
The University of Queensland, Australia
{h.wang16,kevinz,zxf,shazia}@uq.edu.au

ABSTRACT

An increasing amount of motion history data, which is called trajectory, is being collected from different sources such as GPS-enabled mobile devices, surveillance cameras and social networks. However it is hard to store and manage trajectory data in traditional database systems, since its variable lengths and asynchronous sampling rates do not fit disk-based and tuple-oriented structures, which are the fundamental structures of traditional database systems. We implement a novel trajectory storage system that is motivated by the success of column store and recent development of in-memory based databases. In this storage design, we try to explore the potential opportunities, which can boost the performance of query processing for trajectory data. To achieve this, we partition the trajectories into frames as column-oriented storage in order to store the sample points of a moving object, which are aligned by the time interval, within the main memory. Furthermore, the frames can be highly compressed and well structured to increase the memory utilization ratio and reduce the CPU-cache missing. It is also easier for parallelizing data processing on the multi-core server since the frames are mutually independent.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Performance

Keywords

Trajectory, in-memory, column-oriented data structure

1. INTRODUCTION

The trajectory of a moving object is typically modelled as a time-stamped sequence of consecutive locations in a multidimensional (generally three dimensional) space. Thanks to major advances in sensor technology, trajectories are currently generated from GPS-enabled mobile devices and wireless communications at an increasing rate. Trajectory data offers unprecedented amount of information to help understand the behaviour of moving objects. Effective and efficient technologies to manage large scale trajectory data is highly demanded as the foundation to serve a variety of application domains such as geographical information systems, location-based services, vehicle navigation and so on.

Currently, many commercial relational database management systems (RMDBS) have started to offer additional components or extensions to support spatial types and operators. However, only a few simple spatial data types such as points, lines and polygons are considered and supported in these RDBMS. On the other hand, more complex data types are needed to store trajectories. Because it contains continuous time information and the length of each trajectory is variable (i.e., the number of sample point of each trajectory is different). Traditionally, to solve this problem, it is easy to consider the trajectory of a moving object as one database object. While this approach has a clear semantic meaning for each trajectory record, it has several obvious drawbacks. Firstly, most RDBMS cannot handle records of variable lengths well. Secondly, many trajectory processing operations deal with, explicitly or implicitly, some segments of a trajectory only. For example, to find a point of interest (POI) closest to a trajectory, ideally only one part of the trajectory (that contributes to the identification of the resultant POI) needs to be used should that part be identified (using some indexing and filtering strategies). Treating a trajectory as whole cannot support this type of operations efficiently, as it needs to access the whole trajectory and then discard all but the relevant segments.

Recently, there is growing interest on column-oriented data structure such as MonetDB [5], Positional Delta Tree (PDT) [10], and so on. However they did not consider the trajectory data, which is a natural multi-dimensional data structure with complex data format. Meanwhile, the in-memory data management also receives lots of attention such as memory-based indexing techniques [9, 7, 3] and in-memory database systems [8, 6, 1, 4, 2]. However, these work mainly focus on relational data, which means processing spatial-temporal data (i.e., trajectory data) in memory system cannot be well supported. To address this problem, we introduce SharkDB, an in-memory column-oriented storage system for storing massive trajectory data. We propose a novel read-optimized storage structure, which combines the advantages of in-memory computing and column-oriented data structure for analytical tasks. Aligned with common in-memory database designs, where data compression is a key factor of consideration, SharkDB also supports compressing trajectory data effectively and allows processing analytical query

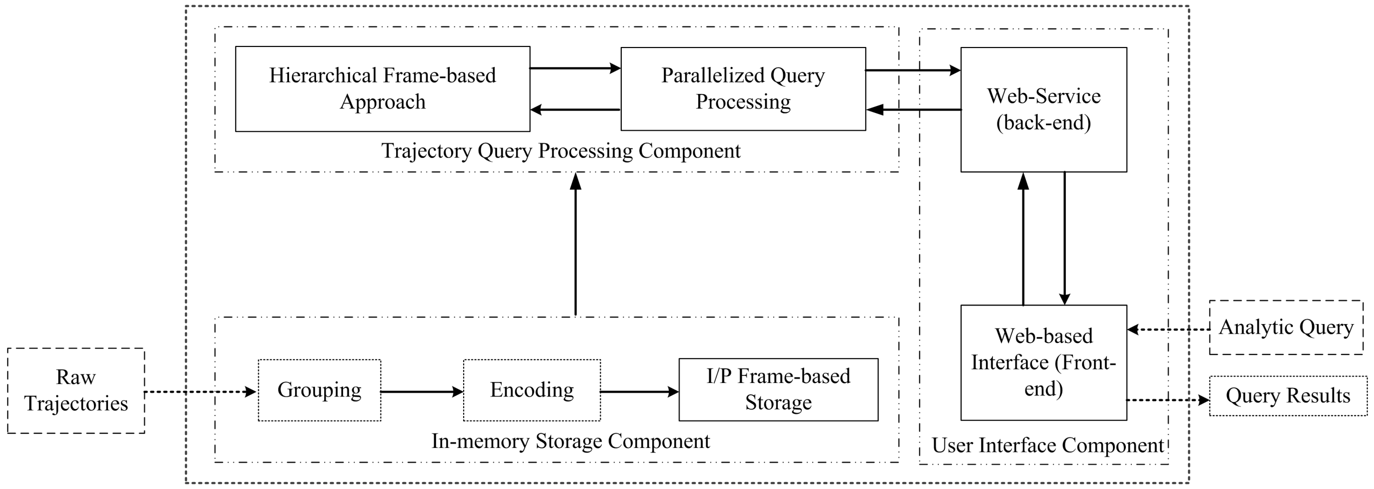

Hierarchical Frame-based Approach
Parallelized Query Processing
Trajectory Query Processing Component
Web-Service (back-end)
Raw Trajectories
Grouping
Encoding
I/P Frame-based Storage
In-memory Storage Component
Web-based Interface (Front-end)
User Interface Component
Analytic Query
Query Results

Figure 1: System Architecture

on the compressed data directly without the need to decompress the whole dataset.

2. SYSTEM OVERVIEW

SharkDB includes three components, the user interface component, the in-memory storage component and the trajectory query processing component. As shown in Fig. 1, the duty of the in-memory data storage component is to encode and compress the raw trajectories, and then to store them in the column-oriented data structure. The trajectory query processing component is responsible for processing trajectory data and answer trajectory-related queries in real time. The user interface component contains a front end (GUI) and a back end. The front end receives queries and communicate with the back end. Meanwhile, the back end is connected with the trajectory query processing component. We give a brief overview of these components in the following:

In-memory Storage System: In this component, we propose a frame-based and column-oriented data structure to store the trajectory data. We partition the whole trajectory dataset and align them into fixed time frame. Each time frame is treated as a virtual column, i.e., all trajectory samples within the same time frame will be stored consecutively in the main memory. We also borrow the idea of video compression techniques and use I/P frame encoding scheme to further reduce the footprint of data in the memory. We will discuss this part in the technical background section.

Trajectory Query Processing System: We propose a hierarchical frame based approach, whereas a multi-level frame data structure has been built, to support efficient trajectory query processing with flexible time granularity. To take full advantage of multi-core computing architecture, we also extend this approach to support parallel processing. We will elaborate this part in technical background section.

User Interface: We use web-based interface as the front end such that it can be accessed from most devices, including PC, tablets and smart phones. Our interface has a control panel to allow users to specify a trajectory-related query and set different parameters. Our back end is implemented as web service, which offers the flexibility to incorporate different storage structures and query processing algorithms for comparison purpose. We will discuss this part in demonstration section.

3. TECHNICAL BACKGROUND

In this section, we first discuss our frame-based data structure as well as its encoding algorithm, which is used to store the raw trajectory data in the column-oriented data structure. Then we briefly discuss the query processing on frame-based structure with parallel processing.

3.1 Frame-based In-memory Storage

Most trajectory-related queries such as window queries, nearest neighbour queries and similarity queries only care about the relevant trajectory segment, instead of the whole trajectory. Apparently keeping each single trajectory as one record is not good storage design for this type of queries, since we need to scan the whole table to get the query answers. Therefore, we implement a frame based data structure in the main memory, by leveraging the column-oriented data structure and the in-memory techniques. To implement this frame based structure, we first build a sequence of frames from trajectory data directly. Then each trajectory sample point is allocated to the related frame based on its timestamp. In this frame based structure, all trajectory sample points that belong to the same time interval are put into the same frame. We name such time interval as frame rate and a sample point in a frame is named as a frame point. For instance, as Fig. 2 shows, if we set the time interval to be 1 minute (i.e. the frame rate is 60 seconds), the time period from 9:01 to 9:05 can be split into 4 frames. Hence, the sample points in the trajectory $T_1 = p'_1, p'_2, p'_3, p'_4$ and $T_2 = p_1, p_2, p_3, p_4$ will be aligned to the related frame. In this example, p'_1, p_1 and p'_2, p_2 are assigned into $Frame1$ and $Frame2$ respectively. However, there are two challenges in our frame-based data structure. The first one is how to reconstruct the raw trajectory from time frames efficiently. The other one is how to reduce the footprint of trajectory data in the memory. We propose I/P frame data structure to tackle these challenges.

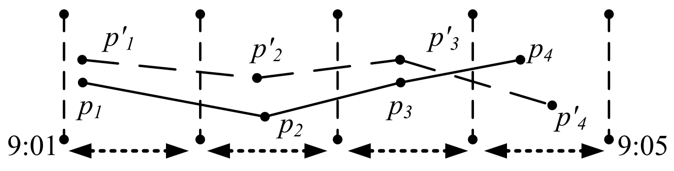


Figure 2: Trajectory Snapshot

There are two steps for building the I/P frame data structure, namely grouping step and encoding step. In the grouping step, the whole sequence of frames are split into small groups, called frame group, and each frame group contains n consecutive frames. Based on the example in Fig. 2 , if we set $n = 4$, then the $Frame1$ to $Frame4$ are allocated in one group. For each group, we set the first frame as I-frame, and the subsequent frames as P-frames. In the encoding process we use the delta encoding scheme to encode the P-frame points by calculating the difference between each P-frame point and its closest prior I-frame point. To increase the reconstruction performance, we set the length of each P-frame column the same as the length of I-frame column. To deal with variable lengths of trajectories, a place-holder Nil will be used once a trajectory segment does not fill a frame group. As shown in Table 1, if a trajectory segment has one I-frame point and one P-frame point in a $n = 4$ frame group, we put Nil codes in the rest of P-frame columns. As a result, the P-frame points now have the same offset (and thus the index) as its related I-frame point if they are belonging to the same trajectory. By this means, the P-frame points can be accessed directly with the index of the related I-frame point without the need to scan the whole P-frame columns. This encoding scheme also enables the delta encoding to compress the P-frame points.

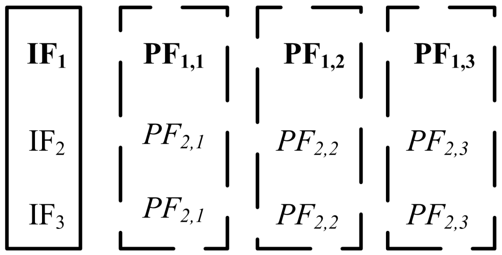
Table 1: Example of I/P Frame-based Structure

9:01-9:02	9:02-9:03	9:03-9:04	9:04-9:05
(1,p)	(1,Δp)	(1,Δp)	(1,Δp)
(2,p)	(2,Δp)	Nil	Nil
(3,p)	(3,Δp)	(3,Δp)	Nil
(4,p)	(4,Δp)	(4,Δp)	(4,Δp)

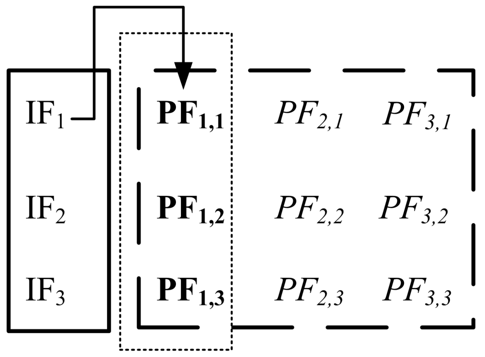
CPU cache optimization is a key factor of consideration for in-memory based storage system. We implement a hybrid data structure to optimize the CPU cache utilization. A two dimensional array $array[x][y]$ is created for each frame group, where x equals to the number of I-frame points in this frame group, and y equals to the number of P-frame columns in this frame group. The value of x in the array is the index of an I-frame point; the value of y indicates the column number of a P-frame group; the object at $array[x][y]$ is the P-frame point. When CPU requests to fetch an I-frame point, its related P-frame points will be also put into the same cache line. Later when CPU tries to access the P-frame points, it can retrieve them directly from its cache that is much faster than memory access. Our experiments have already verified that, this mechanism can increase the trajectory reconstruction speed as well as the whole performance of query processing. Assuming a memory access reads 3 codes vertically from a P-frame array. As Fig. 3(a) shows, it will take 3 memory accesses in order to decode a trajectory segment in a frame group without cache line optimization , since the other codes in the cache line is not related to this segment. To the contrary, as Fig. 3(b) shows, it only takes 1 memory access after cache optimization .

3.2 Trajectory Query Processing

In the frame based data structure, sequential scan for I-frame columns and trajectory reconstruction are the most time consuming operations in the whole query processing. Therefore, reducing the number of sequential scan and trajectory reconstruction can significantly increase the system performance. In order to prune the unnecessary frame groups as much as possible during the sequential scan for I-frame columns, we propose a hierarchical frame based structure, as is shown in Fig. 4.



(a) Non-cache Optimized Structure



(b) Cache Optimized Structure

Figure 3: Example of Cache-aware I/P-frame Structure

First of all, we build hierarchical frame structure in bottom-up style, where the first level (bottom) of the hierarchical frame structure is the normal I/P frame data structure that contains all the frames. Then we only use the I-frame columns from current top level that form the new top level. Afterwards, we split every n I-frame columns and put them into a frame group, where n is the number of frames in a frame group. An advantage of this structure is that we can use the same encoding methods that are proposed before to encode the I-frame columns in the new frame groups. In Fig. 4, an example is provided to illustrate the process. If there are more levels created in this hierarchical structure, the time interval of each frame in the top level will increase and may become greater than the average duration of a trajectory. Therefore if the time duration of frame group is larger than the average time duration of trajectories at the top level, the construction process can terminate.

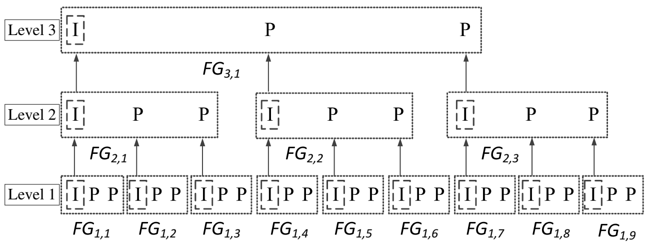


Figure 4: Hierarchical I/P-frame Data Structure

A best-first search paradigm is adopted to traverse the hierarchical frame structure. We first initialize a min-heap and push the top-level frame groups into the heap, of which the time interval of frame groups are in query time range. The entries in the heap are ordered by the MINDIST between their MBRs and query location. Then, for each frame group that is needed to check, we check its MBR first. If the MBR is out of query range, we can prune this frame group and remove it from the heap; otherwise we re-insert

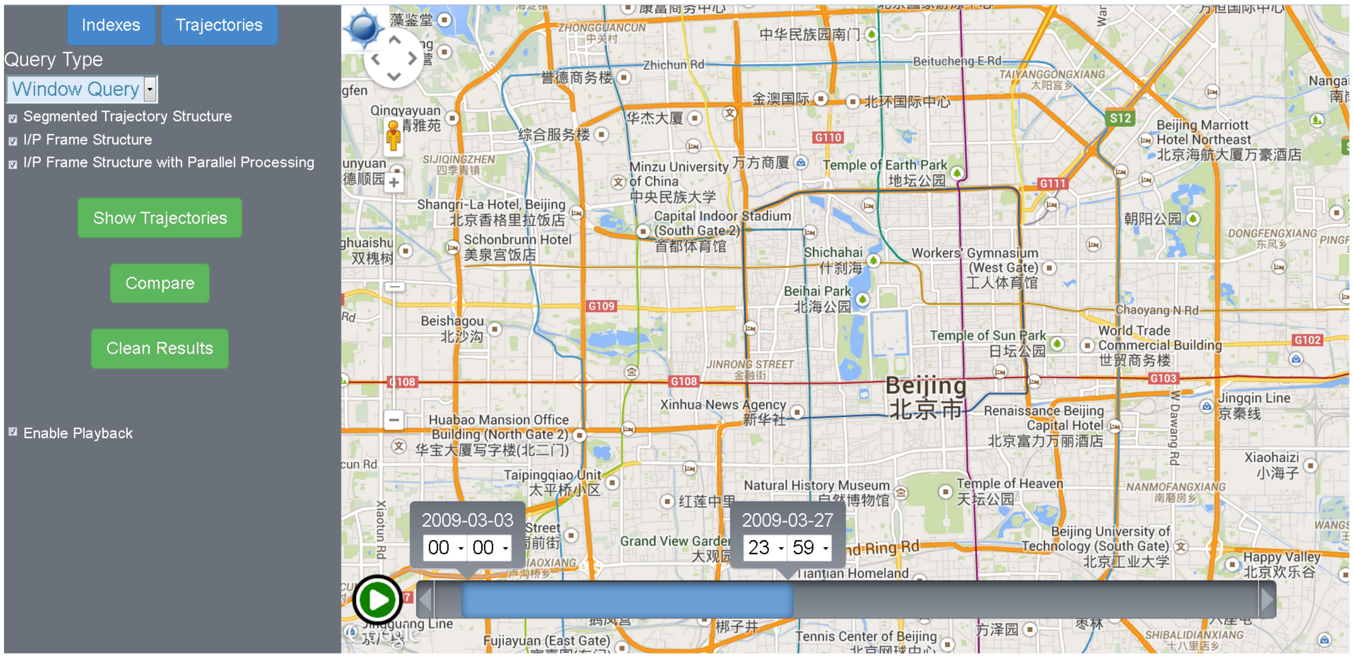


Figure 5: Web Interface

it child frames in the hierarchy back into the heap. When an entry at the bottom level is retrieved from the heap, we decode it to get the raw trajectory segment and calculate the exact distance to the query.

As multi-CPU/core is the standard computing architecture nowadays, supporting parallel processing becomes necessary. One advantage of the frame-based data structure is that each frame group is completely independent and "share nothing" with each others. This allows us to divide a query into a series of sub-queries and assign them to different CPUs/cores for processing. Finally, the results returned from each CPU/cores are merged to get the final answer.

4. DEMONSTRATION

Currently, we implement three storage structures in this demo system, the segmented trajectory structure, the I/P frame structure and I/P frame structure with parallel processing. The last two storage structures are our proposed approaches. The segmented trajectory structure is to simulate a traditional storage, where it considers a trajectory as an object. To increase the performance of the segmented trajectory structure , we split each trajectory into small segments, where a segment contains a fixed number of sample points. Meanwhile, each segment contains its MBR information as well as the starting time and ending time. The aim is to provide an objective time cost comparison among all these structures. These structures can be either deployed on commercial in-memory database (e.g., SAP HANA) or by our own implementation. Two types of trajectory analytic queries are supported at this stage. The first query is window query, which finds all trajectories in the data set that are active during a given period and pass a given spatial region. The other one is kNN query, which finds top-k trajectories in the trajectory data set that are closest to a given point and active during a given period of time. The real datasets used for this demo system is the same as [11]. In this demonstration, we provide a web-based interface as the front end and web service based back end to bridge web interface and the query processing component as a middleware.

The web interface is shown in Fig. 5. The left-hand side is the control panel that allows user to select the storage structure and the type of query from the drop-down list. The right-hand side is the trajectory visualization panel based on Google map, which allows users to input a query intuitively. A user can either draw a query circle (for window query) or pin a reference point (for kNN query) on the map directly. Since our query has the time duration as a parameter, a user can also adjust the query time interval by sliding the time bar in the bottom of panel.

We start the demonstration by briefly explaining the work-flow of our system. Then a user can select the type of queries as well as the type of storage structures from the left panel. As Fig. 6 shows, if the window query is selected, then she can use mouse to draw a circle on the map to specify the query range and adjust the size and location of that circle. The audience can also adjust the time bar to select the day and using drop down list to specify the hour and minute. Upon completion of query input, she can press the "Show Trajectories" button to issue the query and the results should be displayed on the Google Map in real time. The system uses different colors to help users distinguish the trajectories in the result set. Fig. 7 demonstrates an example of the window query in previous example.

After results being shown on the map, the audiences can view the detail of trajectories by switching the control panel tab to "trajectories". Fig. 8 shows an instance of kNN query. In the "trajectories" view, a table is showing the detail of each trajectory in the results, which includes the length of trajectory with its start time and end time. The ID is the rank of the trajectories based on its minimum distance to the query point. If the user would like to investigate the individual trajectory, she can simply click the trajectory in the result table, and the selected trajectory will be highlighted on the map. For example, in Fig. 9 , if we select the first trajectory that listed in the table, then the nearest trajectory will be highlighted in the Map.

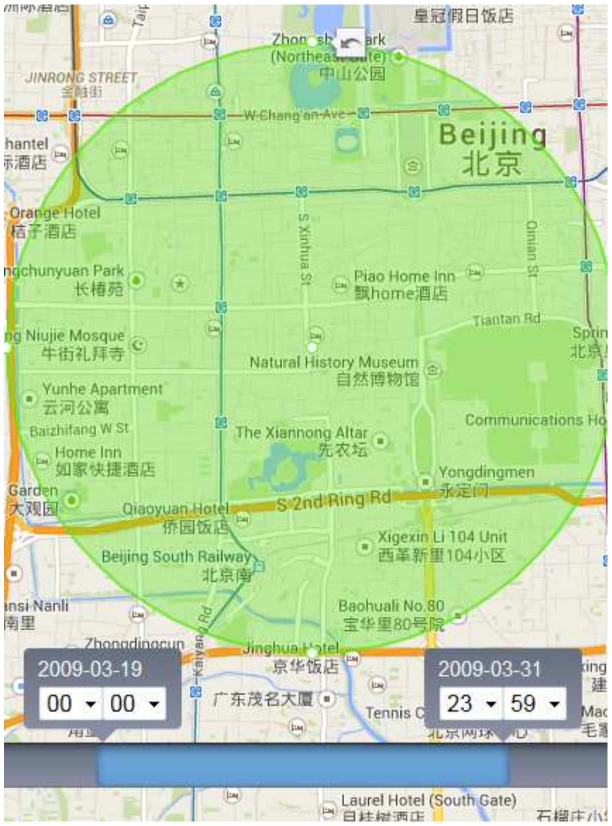


Figure 6: Example of Window Query Input

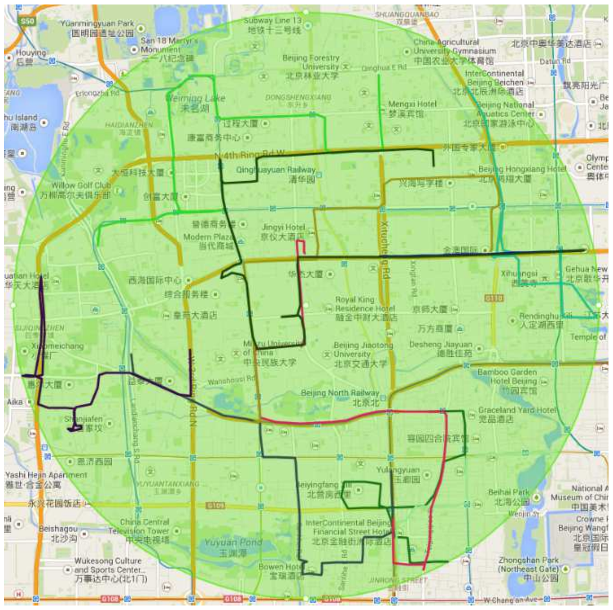
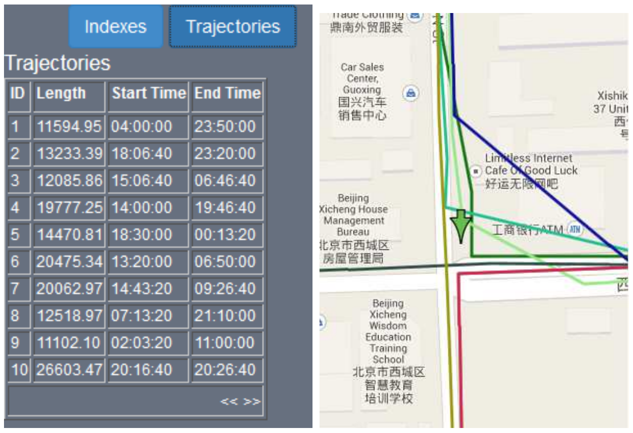


Figure 7: Example of Window Query Results

In addition to showing the results in Google Map, users may be interested in comparing the performances between these storage structures. To do this, they can click the "Compare" button, and the query processing time based on each storage structure will be reported and displayed. An example is shown in Fig. 10, the



Trajectories

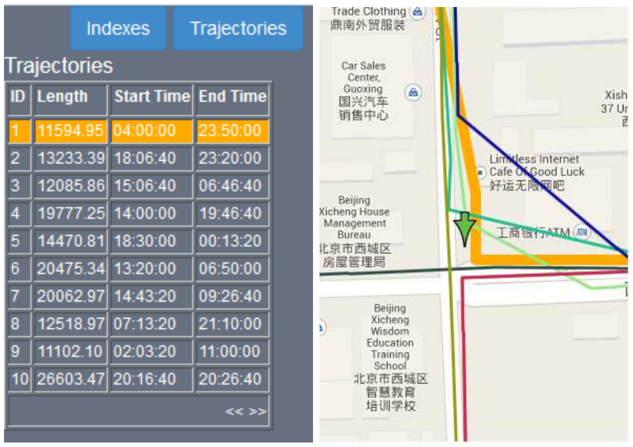
ID	Length	Start Time	End Time
1	11594.95	04:00:00	23:50:00
2	13233.39	18:06:40	23:20:00
3	12085.86	15:06:40	06:46:40
4	19777.25	14:00:00	19:46:40
5	14470.81	18:30:00	00:13:20
6	20475.34	13:20:00	06:50:00
7	20062.97	14:43:20	09:26:40
8	12518.97	07:13:20	21:10:00
9	11102.10	02:03:20	11:00:00
10	26603.47	20:16:40	20:26:40

<< >>

(a) Table (b) Map

Figure 8: Example of kNN Query

segmented trajectory storage, the I/P frame structure and I/P frame structure with parallel processing are denoted as STD, I/P and I/P-P respectively. The y-axis is showing the query processing time of each storage structure in milliseconds.



Trajectories

ID	Length	Start Time	End Time
1	11594.95	04:00:00	23:50:00
2	13233.39	18:06:40	23:20:00
3	12085.86	15:06:40	06:46:40
4	19777.25	14:00:00	19:46:40
5	14470.81	18:30:00	00:13:20
6	20475.34	13:20:00	06:50:00
7	20062.97	14:43:20	09:26:40
8	12518.97	07:13:20	21:10:00
9	11102.10	02:03:20	11:00:00
10	26603.47	20:16:40	20:26:40

<< >>

(a) Table (b) Map

Figure 9: Example of Trajectory Selection on Map

Finally, to make more sense for result visualization, our web interface supports the playback function to visualize the motion of moving objects on the map directly. This function is naturally supported by the frame-based data structure, since all trajectories are synchronized and aligned by time. When a user checks the playback checkbox and issues the query, the back end will turn into playback mode. In this mode, it will send the query to our I/P frame based storage system with playback signal. Then, the system will get the result trajectories and cache the trajectory id, starting time and ending time of these trajectories.

When user presses the play button as shown in Fig. 5, the system searches the frame column by result trajectories' IDs, where the timestamp of this frame group matches the beginning of this query time interval. Then it decodes the frame points to sample points and sends these them back. After receiving such information, the front end displays these sample points on the map. The front end continues to request the sample points in the next frame column based on the result trajectories' IDs, and the points on the map

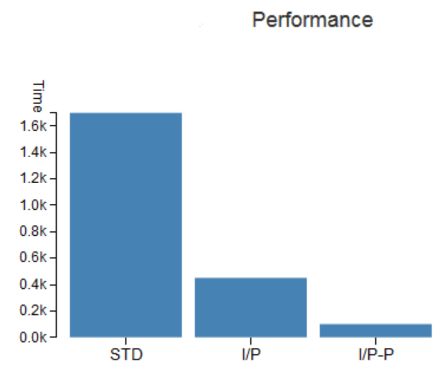


Figure 10: Performance Comparison

will keep moving and represent the motion of the result trajectory. Users can also jump to the time they are more interested directly. Moreover, in the playback mode, there is no need to transfer the whole results set to the client side, so not only the query latency but also the required memory for client devices are reduced.

5. CONCLUSION

In this demonstration proposal, we propose SharkDB, an in-memory frame-based storage design for massive trajectory data. Its novel time-oriented structure is more suitable for processing analytical queries and can make better use of main memory as the permanent storage medium. It also greatly benefits from convenient data compression and parallel processing. We have shown by several demonstration scenarios that SharkDB can process a variant of queries over large-scale trajectory data efficiently and accurately.

6. ACKNOWLEDGEMENTS

This research is partially supported by the Australian Research Council (LP130100164, DP120102829 and DE140100215).

7. REFERENCES

[1] A. C. Ammann, M. Hanrahan, and R. Krishnamurthy. Design of a memory resident DBMS. In *COMPCON*, pages 54–58, 1985.

[2] J. Baulier, P. Bohannon, S. Gogate, C. Gupta, and S. Haldar. DataBlitz storage manager: main-memory database performance for critical applications. In *SIGMOD*, pages 519–520, 1999.

[3] C. Binnig, S. Hildenbrand, and F. Färber. Dictionary-based order-preserving string compression for main memory column stores. In *SIGMOD*, pages 283–296, 2009.

[4] D. Bitton, M. Hanrahan, and C. Turbyfill. Performance of complex queries in main memory database systems. In *ICDE*, pages 72–81, 1987.

[5] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: Hyper-pipelining query execution. In *CIDR*, pages 225–237, 2005.

[6] D. Gawlick and D. Kinkade. Varieties of concurrency control in IMS/VS fast path. *DEB*, 8(2):3–10, 1985.

[7] S. Manegold, P. Boncz, and M. L. Kersten. Generic database cost models for hierarchical memory systems. In *PVLDB*, pages 191–202, 2002.

[8] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD*, pages 1–2, 2009.

[9] J. Rao and K. A. Ross. Making B+- trees cache conscious in main memory. In *SIGMOD*, pages 475–486, 2000.

[10] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented DBMS. In *PVLDB*, pages 553–564, 2005.

[11] H. Wang, K. Zheng, J. Xu, B. Zheng, X. Zhou, and S. Sadiq. SharkDB: An in-memory column-oriented trajectory storage. In *CIKM*, pages 1409–1418, 2014.