DEVELOPING A STORAGE SYSTEMFORGPS DAT USED INTRAFFIC APPLICATIONS

Pham Tran Vu, Nguyen Duc Hai*, Doan Khue

Faculty of Computer Scienceand Engineering, HCMC University of Technology, VNU-HCM

*Corresponding Author: haind@cse.hcmut.edu.vn,

(Received: July 30th, 2016; Accepted: August 31th, 2016)

ABSTRACT

In recent years, GPS signals collected from mobile devices have become an important data source for Intelligent Transportation Systems. However, the rapid increase in the number of GPS transmitters makes those systems receive and process a huge amount of data. This poses a daunting problem for the system designers to build a system with the ability to process the data quickly with high accuracy. In this paper, we introduce a new index structure for GPS data storage system to help solve this problem. Particularly, we store all GPS data in main memory to provide high read/write performance. We also exploit the spatio-temporal characteristics of GPS datasets to create a static partition strategy to evenly distributed the datasets across data nodes in data centers and help the storage system efficiently process range queries. The experiments show that the proposed method not only has high read and write performance but also scales well in distributed environment.

Keywords: *GPS; Intelligent Transportation Systems; spatio-temporal; in-memory; real-time*

TÓM TẮT

Trong những năm gần đây, tín hiệu GPS thu thập từ các thiết bị di động đã trở thành một nguồn dữ liệu quan trọng cho các Hệ thống Giao thông Thông minh. Tuy nhiên, sự gia tăng nhanh chóng về số lượng thiết bị thu phát GPS khiến các hệ thống này phải tiếp thu và xử lý rất nhiều dữ liệu. Điều này gây trở ngại rất lớn cho việc xây dựng các hệ thống có khả năng xử lý dữ liệu nhanh với độ chính xác cao. Trong bài báo này, tác giả giới thiệu một phương pháp tạo chỉ mục mới cho dữ liệu GPS để nguồn thông tin này được sử dụng một cách hiệu quả. Cụ thể, tác giả lưu toàn bộ nguồn dữ liệu GPS trên bộ nhớ chính máy tính để tăng hiệu suất ghi và đọc. Đồng thời, tác giả cũng khai thác những đặc trưng về không-thời gian của dữ liệu để tạo ra một phương pháp phân hoạch tĩnh giúp phân phối đều dữ liệu giữa các máy tính lưu trữ trong các trung tâm dữ liệu. Ngoài ra, phương pháp này cũng hộ trợ rất tốt các câu truy vấn theo vùng. Kết quả thực nghiệm cho thấy, giải pháp được đề ra không chỉ có hiệu suất đọc và ghi cao mà còn hoạt đông tốt trong môi trường lưu trữ phân bố.

OVERVIEW

The rapid development of communication and mobile technology has made applications on the Internet flourish and become popular to mobile phone users. Inside those, GPS is one of the most widely used technologies. Applications in Intelligent Transportation Systems (ITS) such as traffic flow prediction (Necula, 2014) and real-time routing (Ghiani, et al., 2003) are just a few examples mainly using this technology.

High accuracy and low cost of GPS transmitters are parts of the reason why GPS is popular. In addition, information used for navigation embedded in GPS signal is very useful for many applications, especially traffic ones. For examples, we could use GPS signals collected from vehicles moving around a city to monitor the traffic condition. The temporal and spatial information of each signal even help the authorities quickly identify traffic congestion and accidents to take proper actions in time. Therefore, GPS transmitters have been integrated into most of the mobile devices. This, however, causes many difficulties to exploit this potential type of data efficiently. Let us take Ho Chi Minh City as an example. By 2015, the city has about seven million motorbikes, more than 4000 buses, and 12 thousand taxis. Assume that the GPS transmitter integrated in each vehicle creates a new GPS signal to update its current position every one minute. With the average velocity of 24 kilometers per second, the geographical distance between two consecutive signals will be about 400 meters. This figure is relatively high and could greatly affect the accuracy of traffic applications. If we reduce the distance to 100m in order to improve the accuracy, each GPS transmitter must generate GPS signals four times faster. Doing so will create approximately 40 billion GPS signals per day and there will be about 500 thousand signals generated every second. If each signal is only 20 bytes long, we still have to spend up to 1TB for storing data of one day. All of those figures are very difficult to handle with current technologies.

Since most of the users do not want to wait for the results of their request for so long, most of the applications focus on reducing the execution time of processing requests to improve user experience. With the data and workload mentioned above, it is difficult to handle them within a short amount of time.

Motivated by those demands and challenges, designing an efficient storage system for GPS data is an urgent need and has been interested by many research groups. GPS is considered as spatio-temporal data so it has characteristics of both time and space. A lot of research associated with this data type focus on developing index structures to store and efficiently exploit its multi- dimensional characteristics. Classical indexes such as R-Tree (Guttman, 1984) and QuadTree (Samet, June, 1984) have been invented for a long time but are still being used by many applications. Besides, Geohash (Niemeyer, n.d.), an encoding technique which transforms geographical information to strings, are also used widely in many databases.

Additionally, in the Big Data era, the rapid increase of data volume makes storing data across multiple data nodes become a popular trend. Index structures also evolve to adapt with those changes. SD-Rtree (Mouza & Litwin, 2007) is an example of a variance of R-Tree designed for distributed systems. There are also dedicated storage systems for spatio-temporal data such as PIST (Botea, et al., 2008). This system splits the dataset into multiple cells according to its spatial characteristics then constructs an index structure based on temporal features in each cell to accelerate the searching process. Furthermore, many works have tried to use MapReduce/Hadoop platform in order to deal with huge GPS datasets efficiently. Hadoop-GIS (Aji, et al., 2013) is a representative example of this approach. Such systems develop several tools work upon the Hadoop platform to decompose requests from the applications into multiple small MapReduce jobs and schedule their execution. Those solutions, however, only work well with batch applications which require high throughput. They do not offer fast data processing which is greatly needed by real-time applications.

Methods mentioned above are all designed for secondary storage devices such as hard disk and flash. Those devices have large capacity but perform slowly. As the amount of workload is keeping increasing and the time restriction is getting tighter, it seems to be inefficient to keep all data on those devices. Recent studies tend to move data to main memory (RAM) to take its advantage in speed to improve the performance of data access. There are even high-quality stores that keep all of its data in RAM and widely used by many large applications. Redis and Memcached are just two representative examples of such products. However, to the best of our knowledge, the organization of those stores is still simple and none of them fully supports spatio-temporal data.

In this paper, we propose a novel method to manage spatio-temporal datasets (typically GPS) in memory of distributed storage systems. Particularly, we exploit the temporal and spatial characteristics of GPS data to construct a multi-layer index which allows data to be read at very low latency and written at very high throughput. In addition, our design also guarantees load balancing and scales well in distributed environment.

METHODS

Characteristics of GPS Data

We collect GPS data generated from buses moving around Ho Chi Minh City in several days and plot them on a digital map of the city. We split the map into uniform cells by a grid with the size of 40x40 cells.GPS signals in each cell are counted and reported with the color of the cell as shown in Figure 1.

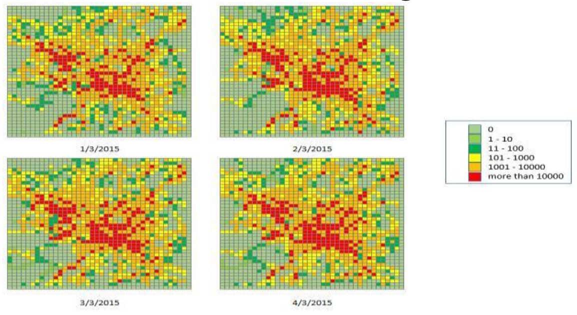

Figure 1. The spatial distribution of GPS data in Ho Chi Minh City in four consecutive days in March, 2015.

Clearly, there is no significant difference in the spatial distribution of the data observed among reported days. This is because the characteristics and behaviors of the moving objects do not change much as time being. For example, a student could follow different paths to go to school on different days. However, within a month or a year, he would just follow a couple of paths. Therefore, if we collect the sample data in long enough time, there should be no difference among intervals. Furthermore, the distribution of data reflects the distribution of population and city planning: most of GPS signals are created in the city center and areas with high population while there are just a few ones in the suburban areas.

As city planning and infrastructure change slowly and the behavior of moving objects tends to repeat periodically, we assume that the spatial distribution of GPS data is stable. This is an important assumption for our proposed method.

A Storage System for GPS Data

We decide to keep data in main memory (RAM) to satisfy strict time requirements of real- time applications. Figure 2 describes the overall architecture of the storage system constructed according to this decision. Fundamentally, the design is based on the organization of S4STRD (Pham, et al., 2015). The system consists of two parts: a RAM Cluster and a disk-based database. RAM Cluster is responsible for storing data in main memory and using them to directly process requests from applications. The disk-based database is a NoSQL database whose main role is to keep the data permanently on secondary storage devices.

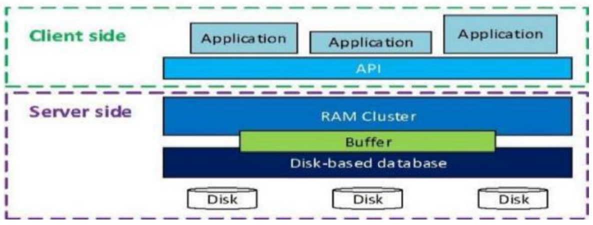

Figure 2. The overall architecture of S4STRD.

Every signal sent to the system is written into RAM Cluster before being synchronized into the disk-based database. Since there is a large gap between RAM speed and that of secondary storage, we implement a buffer between the two stores. After RAM Cluster writes a new signal to main memory, the signal will then be pushed into the buffer. The buffer itself gradually writes the data to the disk-based database.

Applications can interact with the system through API layer provided as a programming library. This layer has a responsibility to manage connections to the storage system and handle data transmissions. Those tasks are transparent to applications. They just need to describe their requests, then call corresponding services, and process the response.

Data Organization

Let be a set of GSP points within a given data space Ω. Each point ∈ is defined by a pair of real numbers representing its coordinates (�, �). Ω = ⟨ , � ⟩ is a rectangle defined by its bottom-left corner and upper-right one �. A range query � = ⟨� , �� ⟩ is also a rectangle where � , �� ∈ . Executing the query returns the result which is a set of points = (�, �) ∈ falling into �. In order to specify those points, it is necessary to scan through several points in . A good data management method would process such queries with the number of points to be checked as low as possible.

Current techniques split Ω into subspaces then query on them instead of checking Ω to minimize the number of tests to be done. Subspaces are usually organized in an order (hierarchically or equally) to help speed up the retrieval process. This solution significantly increases the performance but might hinder writing process since changes in data can make the structure of the dataset violated the rules set out in advance and that the system

must give a lot of time to restructure (e.g., a storage node on the R-Tree splits into two child nodes because of overload).

To achieve high performance in both writing and reading data, we use SIDI (Nguyen, et al., 2015) to manage data due to its simplicity, low cost in restructuring and in line with distributed systems. SIDI uses a grid � of n×n uniform rectangular cells to cover the whole space Ω. Each cell is identified by its coordinates (�, �) where � is the number of cells in the same row to the left and � is the number of cells below it. Low-density adjacent cells are combined to form a new unit, called "zone", specified by a 2-tuple ⟨ �, �⟩ where �is its lower-left cell and � is its upper-right one. The aim of using zone is to reduce the density gap between areas: SIDI tries to add more cells to zones in low density regions to equalize their density with those in high density areas. Hence, distributing data by zone will ensure even data distribution. Partitioning dataset into independent zones also guarantees fast fault recovery since the system only needs to reload data from missing zones in crashed data nodes instead of the entire index.

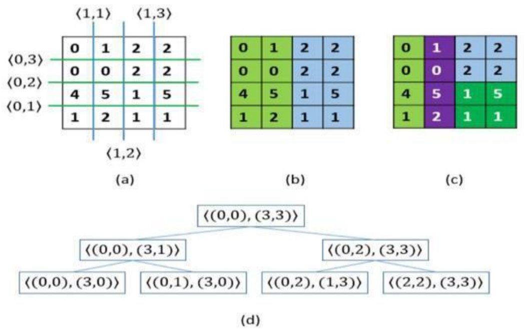

Figure 3. Data partitioning process.

How zones are constructed is not aligned with the definition. Denote � as the set of points falling into zone � then |� | is the sum of the density of cells forming it. Let �Ω = ⟨(0,0), (− 1, − 1)⟩ be the largest zone covering the whole space Ω, we build a zone structure over � by recursively splitting �Ω into two smaller ones using horizontal and vertical cuts until we have zones with the density not exceed a given Θ. At each stage, a zone is split into two halves by a cut such that the difference in data density of two new zones created by this cut is minimized. The set of created zones could be organized as a binary tree, whose root is �Ω, called partition tree.

Therefore, it is visible that produced zones (on leaf nodes of the partition tree) differ from each other within 1 level (i.e. the number of edges in the path from this node to the root) and the deviation in their density tends to be close to Θ.2

Figure 3 shows an example of how the algorithm works on zone � = ⟨(0,0), (3,3)⟩ with Θ = 7. The number in each cell represents its density. There are 6 possible cuts to be applied on �: ⟨0,1⟩, ⟨0,2⟩, ⟨0,3⟩ (horizontal ones) and ⟨1,1⟩, ⟨1,2⟩, ⟨1,3⟩ (vertical ones). The cut ⟨0,1⟩ divides � into two child zones � = ⟨(0,0), (0,3)⟩ and �� = ⟨(1,0), (3,3)⟩.

Hence, |� | = 1 + 2 + 1 + 1 = 5 and |� | = 24,infers that ||� | − |� || = |24 − 5| = 19. Similarly, differences in density of two child zones when applying the other cuts are respectively 11, 19, 19, 3, and 9. Among these, 3 is the smallest number so the cut ⟨1,2⟩ will be the chosen one as shown in Figure 3b. The same procedure is applied for both child zones resulting in Figure 3c. The algorithm ends as no zone having density that is greater than Θ = 7. The density gap is now decreased from 5 to 3. Figure 3d illustrates the partition tree of the whole process.

Indexes on time and device ID attributes are considered as sub-index layers under spatial partitioning. Since spatial constrains allow range queries to result in zones, it is necessary to put sub-indexes inside each zone to handle queries on other fields. Besides, spatial partitioning only specifies data position at data node level, hence, we need to choose either time attribute or device ID attribute to be the primary index to decide how data is stored physically on a node. Since time attribute is a numerical type, which consumes less time in processing than string type (device ID) does, it is suit to build the primary sub-index on this field and leaving device ID as a secondary index.

About constructing index structure, index on time attribute is constructed in an array arranged in written time order. Binary search will be used when processing queries on a certain range of time. For the device ID attribute, queries usually specify one or many different IDs, so hash table is a suitable structure for a string-based attribute like this. Inside each table's bucket, data is also stored in an array in order of written time to escalate processing of queries related to both device ID and time attribute.

Load distribution policy

GPS points ∈ are stored in a system � including � computer node �1, �2, ⋯ ��−1. Since RAM capacity is much less than that of secondary storage, we decided to keep just one copy of each record of in a node � ∈ �. And since data is partitioned in zones, distributing data to nodes will be implemented at zone level. Denote � as a result set of mapping from a set of zone to �: � = {� : � ∈ �} Where � is a set of zones assigned to a node � : � = {� ∈ : � : � ↦ � }. Denote as a total capacity of the whole system: = Σ ∈

The amount of data scattered to a data node should be according to its capacity compare to that of other nodes in the system. Particularly, the greater the value of is, the higher load is distributed to node � . This is based on the assumption that data nodes in the system have different configurations. If the load is distributed equally, low capacity nodes will become a bottleneck of the system. Thus, those with high capacity should receive the higher load to ensure global performance.

Since data is group by zones, we use zone density as the smallest unit to calculate the load on each data node. Thanks to the property of partitioning algorithm, the density of zones at the same level tend to be equal to each other and double that of its children. Let � be the highest level of zones in , every zone � at level ℎ will have load weight � determined as follows: � = 2 −ℎ

Zones at lower level (closer to �Ω) have the higher load weight. This weight reflects the relative proportion between levels of density.

Since zone's density is the unit of the load, and let zones at level � have the load of 1, the sum of load on � will be: � = Σ � ; ∈ Denote � as the sum of load on node � specified as follows: � = Σ � ; ∈� For the load on each node � equivalent to its capacity, then: � ≈ ; �

Thus, in order to achieve load balancing, we must distribute zones to data nodes in the way so that the above condition is satisfied on every � ∈ �. In addition, locality should also be reserved to avoid request to be sent to so many data nodes causing communication overhead and producing too much load. Therefore, load distribution process should scatter zones across storage system by groups of adjacent zones to guarantee locality.

EXPERIMENT RESULTS

All of our experiments are conducted on the GPS datasets collected from buses in Ho Chi Minh City. We implement our solution based on an existing implementation of S4STRD. Particularly, S4STRD will take care of storing data and transferring data between the system and applications. We just modify its data management mechanisms and request processing modules (including read and write) using the method we have described above. We conduct experiments on both the modified version of S4STRAD and MongoDB to evaluate the efficiency of the proposed method. We choose MongoDB because it utilizes Memory Mapping for data management. This technique is very similar to storing data in memory explicitly.

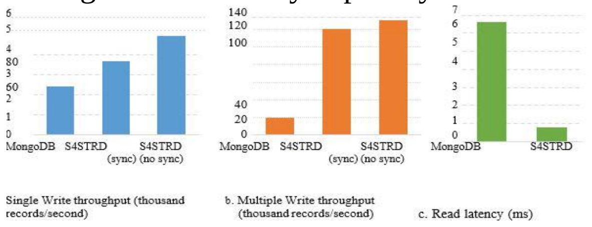

Figure4. Write andreadperformanceof S4STRD and MongoDB.

Figure 4 compares the performance of Single write (write GPS signals one by one), Multiple Write (write multiple GPS signal simultaneously), and read methods of S4STRD with those of MongoDB. Clearly, S4STRD outperforms MongoDB in all cases as its write throughput is about 6x times faster than that of MongoDB even when synchronization mode is on (keep synchronizing data from main memory to disk). It also reads data 7x times faster than MongoDB does.

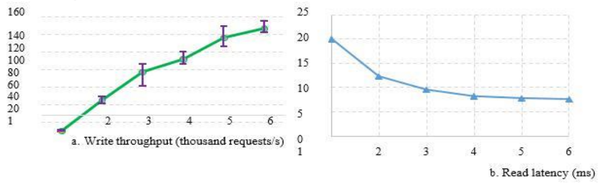

Figure 5. Read and write performance of S4STRD when changing the number of data nodes.

Figure 5 shows that the scalability of proposed method. Clearly, it scales relatively well as both write throughput and read latency are improved significantly when we add more data nodes to the system.

CONCLUSION

In this paper, we proposed a novel storage system which is dedicatedly constructed for spatio- temporal data (GPS for particular). The storage is designed to efficiently manage data in main memory of distributed storage systems. The most noticeable feature of this system is that it exploits the stable of spatial and temporal distribution of GPS data to build a multi-layer index structure which allows data to be read at very low latency and to be written at very high throughput. In addition, the proposed method also guarantees load balancing and scales well on distributed environment.

Although observation, analysis, and evaluation processes are carried out based on the dataset of buses moving around Ho Chi Minh City, we believe the proposed method could be applied to other areas. This is because the infrastructure and the behavior of moving objects are identical in cities in Vietnam.

In the future, we plan to improve the system by doing further research in data distribution since this is the primary factor that strongly affects other components of the design. In addition, we want to expand the research to using data collected from other sources such as mobile devices, camera, etc. in order to have a wider view of the traffic in the city.

Acknowledgement: This work is funded by Ho Chi Minh City University of Technology under grant number SVKSTN-2015-KH&KTMT-08 and Vietnam National University – Ho Chi Minh City under grant number B2014-20-07.

REFERENCES

AJI,A. et al., 2013. Hadoop-GIS: AHigh PerformanceSpatial DataWareshousingSystem over MapReduce.*Proceedings of theVLDBEndowment,*6(11), pp. 1009-1020.

GHIANI,G., GUERRIERO, F.,LAPORTE, G.& MUSMANNO, R., 2003. Real-time

GUTTMAN, A., 1984. *R-trees: a dynamicindexstructure for spatial searching.*New York, NY, USA, s.n., pp. 47-57.

MOUZA, C. & LITWIN,W., 2007.*SD-Rtree: Ascalabledistributed Rtree.*s.l., s.n. NECULA, E., 2014. *DynamicTraffic Flow Prediction Based on GPS Data.*Limassol, s.n., pp. 922-929.

NGUYEN, D.H., DOAN, K. &PHAM, T.V., 2015.*SIDI:A ScalableIn-MemoryDensity- based Indexfor Spatial Databases.*Kyoto, The7thInternational Workshop on Data IntensiveDistributed Computing.

PHAM, T. V., NGUYEN, D. H. &DOAN, K., 2015.*S4STRD: AScalablein MemoryStorage Systemfor Spatio-Temporal Real-timeData.* Chengdu, s.n.

SAMET, H., JUNE, 1984. TheQuadtree and Related Hierarchical Data Structures.*ACM Computing Surveys(CSUR),*16(2), pp. 187-260.