VegaCache: Efficient and Progressive Spatio-Temporal Data Caching Scheme for Online Geospatial Applications

Yunqin Zhong[1,2]*, Jinyun Fang[1], Xiaofang Zhao[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
*Corresponding author, e-mail: zhongyunqin@ict.ac.cn

Abstract—**With the emergence of online geospatial applications such as WebGIS services, the large spatio-temporal data and numerous concurrent users have posed grand challenges on the state-of-the-art spatial databases due to their intolerable disk I/O latency. Hence, the clients cannot be served with real-time response. In this paper, we propose VegaCache, a distributed in-memory caching scheme, to provide low-latency access for time-critical geospatial workloads. VegaCache has three significant characteristics. First of all, it is composed of tunable memory pools maintained by commodity cluster and thus its capacity can be increased linearly with more nodes. Moreover, a data replacement model is designed to determine loading hotspot data into and swapping out victim data from the distributed cache. In addition, we design a localized caching model to reduce data transfer delay with consideration of geographic proximity and geospatial access patterns. Comprehensive experimental results show that VegaCache is able to provide sustained I/O throughput and high efficiency access and thus could meet the requirements of geospatial applications.**

Keywords-spatio-temporal data management; spatio-temporal database; distributed cache; WebGIS; geospatial application; spatial cloud computing

I. INTRODUCTION

With the development earth observation technologies and advancement of data capture techniques, the collected spatio-temporal data is growing rapidly with an unprecedented speed. Moreover, with the prevalence of online geospatial applications such as Web-based Geographic Information System (WebGIS) and Location-based Services (LBS), a large number of common amateurs alike to professionals become both users and producers. In turn they have generated large amounts of volunteered geographic data [1] and user-generated geospatial datasets [2]. The desiderata for the backend data management infrastructure used in geospatial applications include high I/O throughput, dynamic scalability and efficient spatio-temporal data access.

The Spatial Database Management Systems (SDBMS) are served as the state-of-the-art data infrastructure for geospatial applications. The big spatio-temporal data and numerous concurrent accesses have posed grand challenges on backend SDBMS due to its intolerant disk I/O latency. Firstly, web-based geospatial applications are characterized as data-intensive and access-intensive. The SDBMS cannot provide high I/O throughput due to the limited storage performance of the underlying DBMS. Secondly, most of the existing methods are based on single-node SDBMS; as a result, their access efficiency cannot be improved as the data and concurrent users increase, due to poor scalability. Thirdly, most of the geospatial applications are read- and write-heavy access patterns, and the I/O access latency becomes the performance bottleneck while processing geospatial computation. The spatial indexes and data retrieval patterns of SDBMS are tuned for disk rather than memory cache. Since the disk seek operation is costly, I/O overload is unavoidable in SDBMS while there are numerous concurrent accesses for geospatial data.

Motivated by the above deficiencies, in this paper, we propose VegaCache, a novel distributed in-memory spatio-temporal data caching scheme, to meet the real-time access requirements of online geospatial applications. VegaCache takes advantage of elastic physical resources of shared-nothing commodity cluster, and it serves as a “big memory” from applications’ perspective. Each node contributes a tunable memory pools, and VegaCache composes a larger memory by combining these pools together via a network switch. VegaCache provides elastic memory resource that varies with data volume by adding or removing cluster nodes, and its memory capacity can be increased linearly with larger clusters. The active geospatial data are resident in VegaCache for serving online applications, while the disk-based SDBMS is worked only for data backup and recovery.

Moreover, we designed an Object Replacement Model (ORM) to determine which data should be loaded into or swapped out from the distributed memory cache. The access frequency of data object is counted periodically and the top-ranked objects are marked as “hotspot” data with respective ratings. Once an object is determined as hotspot data, its geographically adjacent objects are dynamically loaded into VegaCache and then clients are requesting data from memory cache rather than disk, which can be greatly improv the access efficiency without disk I/O latency. Since the memory capacity is much smaller than the data volume, the effective use of memory resource is of significance. Thus, the low-rating data are marked as “victim” data and will be swapped out of memory.

This work is supported by the National High Technology Research and Development Program (863 Program) of China (Grant No.2011AA12030 and No. 2011AA120302), the Knowledge Innovative Program of The Chinese Academy of Sciences (Grant No. KGCX2-YW-174), and The CAS Special Grant for Postgraduate Research, Innovation and Practice.

Besides, to reduce data transfer overhead, we design a Localized Caching Model (LCM) with consideration of geographic proximity and geospatial data access patterns. The adjacent geospatial objects are persistent in the sequential disk pages of cluster node in terms of their spatio-temporal proximity. LCM guarantees that the data in the local disk are firstly resident in the same memory pool maintained by VegaCache. If the local memory is used up, the data will be transferred to a remote memory pool. The spatio-temporal data objects are grouped together in terms of geographic location and each object is assigned a unique identifier as the *key*. The data objects belonging to the same region will be organized into the same cache pool. VegaCache is transparent to geospatial applications, and it provides unified interfaces for clients.

In addition, we implement VegaCache middleware on top of the cloud platform, and Memcached is adopted as the sever-side daemon into which the data is loaded. The data objects are distributed and manipulated from client-side, and the ORM and LCM implementation details are encapsulated into VegaCache runtime system and served clients as the client library. As memory becomes cheaper and network capacity increases, VegaCache could be scaled out with larger cluster and faster hardware. It serves requests by fetching data from the memory pool rather than disk pages, and hence it could eliminate I/O latency while there are numerous concurrent accesses.

We have conducted comprehensive experiments to evaluate the performance of VegaCache with various benchmarks. With the help of cluster technology, VegaCache can easily handle millions of requests per minute and respond to clients in less than ten milliseconds, which is imperative for reducing access latency in online geospatial applications that require intensive I/O operations. Moreover, the cache hit rate will be improved with more physical memory resources and a larger cluster. The experimental results show that the geospatial processing efficiency could be improved by several orders of magnitude and the access efficiency is increased by more than one hundred times as well.

In summary, our contributions could be described as follows. Above all, we present a distributed caching scheme called VegaCache for geospatial applications that involves big spatio-temporal data and numerous concurrent users. The scheme is characterized as distributed, scalable, cost-effective and has high throughput. Besides, we design ORM and LCM models to improve access efficiency according to geospatial access patterns. In addition, we implement VegaCache with cloud technology based on clusters and deploy real applications to confirm its feasibility and effectiveness.

The following of the paper is organized as follows. Section 2 presents the details of our scheme. Section 3 discusses the experimental results and analysis. Section 4 describes the related work and we conclude this paper in Section 5.

II. DESIGN AND IMPLEMENTATION DETAILS OF VEGACACHE

In this section we present the details of system architecture, ORM and LCM models, and the usage and optimization of VegaCache will be described as well. We begin by reviewing access patterns of geospatial applications, and then explain how VegaCache meet the requirements and its operation principle.

A. VegaCache System Architecture

Our scheme bridges the gap between access patterns of geospatial applications and provisions of cloud computing cluster. It takes advantage of storage and computing capabilities of cluster nodes. Likewise, VegaCache is composed of memory pools and its capacity is the total size of all pools. The online geospatial applications are served by VegaCache buffer manager, and geospatial data are stored in memory represented as binary *key-value* objects via distributed caching middleware. Furthermore, since memory is no longer a scarce resource and its capacity can be extended with more nodes, all of the online geospatial data could be resident in the big memory of VegaCache. To guarantee the data integrity and fault-tolerance, we have designed spatio-temporal data backup and recovery function by using the cloud storage system. The disk-based SDBMS and NoSQL database (i.e., HBase) are adopted as the long-term data preservation system for geospatial applications.

Since the geospatial access pattern is *read many write once*, the frequently accessed data in VegaCache have three replicas in the backend storage system. The new data are updated into memory pool with write-ahead-logging (WAL) mode, in other words, the data is written into the memory pool after logging process behaviors. Once the VegaCache cluster node has failure problems such as network partition and power off, the updated data will be recovered by redoing the operation logs, and then sends the latest data to the nodes where the replicas are preserved. Finally, the newest data will be reloaded into VegaCache for serving requests.

VegaCache has characteristics of being distributed, layered and loosely coupled. Fig.1 shows the system architecture of VegaCache,; it consists of two modules, e.g., online spatio-temporal data buffer manager, and data backup and recovery sub system. VegaCache buffer manager serves the I/O-heavy workloads that require real-time response and low latency, whereas the data backup and recovery module is served for offline access and fault-tolerance.

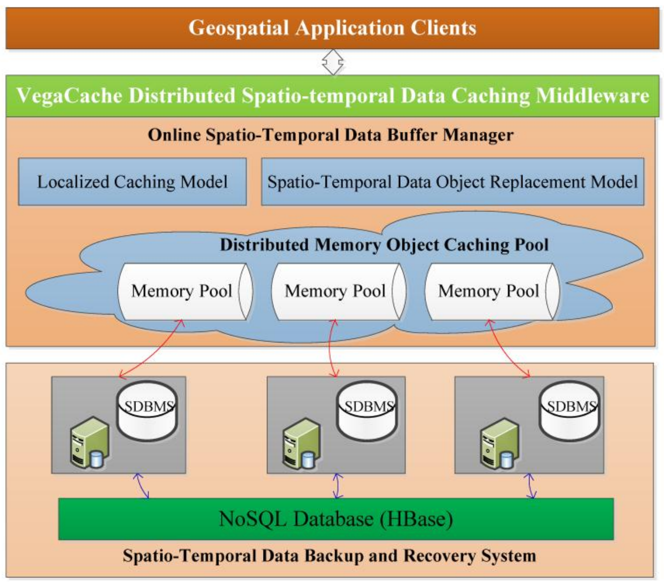
Geospatial Application Clients
VegaCache Distributed Spatio-temporal Data Caching Middleware
Online Spatio-Temporal Data Buffer Manager
Localized Caching Model
Spatio-Temporal Data Object Replacement Model
Distributed Memory Object Caching Pool
Memory Pool
Memory Pool
Memory Pool
SDBMS
SDBMS
SDBMS
NoSQL Database (HBase)
Spatio-Temporal Data Backup and Recovery System

Figure 1. System Architecture of VegaCache

As shown in Fig.1, the data backup and recovery subsystem is built on top of cloud data management infrastructure, which contains NoSQL database and SDBMS. We use HBase, a leading NoSQL database based on the Hadoop cluster, to preserve the big spatio-temporal data and semi-structured data such as imagery data and textural Point of Interest (POI) data. Since HBase only has support for key-based query predicate, it cannot process spatio-temporal queries, which limits its capability. To meet the requirements of geospatial applications, we have designed a *bidirectional transmission tunnel* to exchange data between HBase and SDBMS. The spatio-temporal query requests are redirected to SDBMS for further spatial computation and the query processing results are transferred to the VegaCache buffer. The Memcached server is widely used in web applications. Each VegaCache node manages one or more Memcached server and a Memcached daemon maintains one memory pool, i.e., VegaCache is composed of many memory pools, and the ORM and LCM models are integrated into the buffer manager module. From the applications' perspective, VegaCache is working as the distributed caching middleware that manages the memory pools and bridges the gap between Memcached and spatio-temporal access patterns.

The geospatial application clients interact with VegaCache middleware via OGC-compatible spatial access interfaces. VegaCache provides a unified global view for geospatial applications, and the details of data distribution, request scheduling, remote memory management and storage organization that are transparent to applications. The clients can utilize the big distributed cache just they call the local memory cache. With the data residing in the memory, applications are not necessary to access disk, and the access latency can be greatly reduced without disk I/O. Hence our scheme could guarantee real-time response to meet the requirements of online geospatial application services.

B. Spatio-Temporal Data Object Replacement Model

VegaCache is a cost-effective solution to improve access efficiency by leveraging distributed in-memory object caching technique. To make full use of physical memory resource, data objects that will not be accessed in recently should be swapped out of VegaCache. We present a data replacement model specifically for geospatial application, and it takes characteristics of spatio-temporal data and geospatial access patterns into account. The traditional Memcached caching methods are designed for web applications with simple data models such as textual and structured records, whereas the spatio-temporal data are semi-structured, high dimensional and complex, our scheme should bridge the gap between key-value object caching model and geospatial applications.

Since the Memcached server provides simple key-value data models, the spatial and temporal information is encrypted into the KEY by computing the Hilbert curve value of data objects, while the VALUE object is the real data content of raster image data and vector-based feature represented by Well Known Binary (WKB) and Well Known Text (WKT) format. Table I shows a reference of the basic notations used in this paper, and they will be explained at their first appearance.

TABLE I. BASIC NOTATIONS

Notation	Definition
Ψ	The total capacity of distributed memory cache pools in VegaCache.
N	Number of cluster nodes.
M	Total Number of spatio-temporal objects in the distributed cache.
P	The number of memory pools of cluster.
i	The node indicator *i-th* node ($1 \le i \le N$).
j	The *j-th* memory pool on cluster.
R_r	The geographic area that maintained by the *r-th* memory pool ($1 \le r \le j$)
k	The *k-th* spatio-temporal object ($1 \le k \le \mathrm{M}$).
m_j	The size of memory pool on *j-th* node.
n_i	The number of memory pools on *i-th* node.
s_k	The size of data object ($\min \le s_k \le \max$).
$\mathrm{H}_d(x,y,t)$	The value of *d-order* Hilbert curve, an object is indicated by two-dimensional spatial location (*x, y*) and timestamp (*t*).
B	The network bandwidth of cluster, which is a constant determined by communication switch.
ξ	A constant of threshold size of sub region data.

The access behavior of clients will be traced and recoded into logging files. VegaCache daemon threads will analyze the log and periodically count the access frequency of data objects, and hence determine the hotspot geographic region and sort the ratings of accessed data objects dynamically. The VegaCache capacity is depended on the number of cluster nodes and number of respective memory pools on each node. It is formulated as (1):

$$\Psi = \sum_{j=1}^{P} m_j \text{, where } P = \sum_{i=1}^{N} n_i \quad (1)$$

The memory cache capacity of VegaCache is the total size of all memory pools on cluster nodes. A memory pool maintains one geographic region. In our proposal, the whole map layer is partitioned by quadripartition strategy, i.e., a region is divided into four sub regions until their size reached to a given threshold. The threshold is an empirical value, which is determined by data volume size and memory pools of cluster. The spatio-temporal objects are sorted with space filling curve, and Hilbert curve is chose for its preservation of spatial proximity. The Hilbert value of objects is used as the KEY in VegaCache memory pool, and it is computed by $\mathrm{H}_d(x,y,t)$, where (x,y,t) represents the spatial location and timestamp. The d represents order value of Hilbert function, and its numeric range is formulated as (2):

$$d = \sqrt{\frac{M}{P}} \text{, where } M = \Psi / s_k \quad (2)$$

Thus, $d = \left(\sum_{j=1}^{P} m_j \Big/ \left(s_k \cdot \sum_{i=1}^{N} n_i\right)\right)^{\frac{1}{2}}$, where $\min \le s_k \le \max$. In practical applications, the d value is computed while the data are imported into cloud storage systems. The spatio-temporal data are stored as <KEY, VALUE> objects in the data blocks. Typically, the block size is determined by trade-off of geographical area (R_r) and memory size (Ψ), and the larger block is beneficial to disk-based distributed file system, whereas VegaCache is discriminated in favor of smaller block. According to Tobler's first law of geography, the clients are vulnerable to access the geographically adjacent data objects and different clients may access different areas. If more sub regions are loaded into memory cache, more concurrent users will be served by VegaCache, and hence it could improve the whole access efficiency with most of the accessed data are pre-fetched into distributed cache.

Let W denote the whole map layer, and its geographic region is represented by its Minimum Bounding Rectangle (MBR), i.e., $MBR(W)$. The spatio-temporal data are partitioned into smaller data fragment, and their size is close to the threshold (i.e., ξ). However, the geographic area is not split evenly, and the split process is recursively executed until all sub regions are satisfied the threshold. The data objects are evenly distributed across VegaCache memory pools in terms of their geographic regions, and each memory pool maintains objects belonging to a region. The dynamical spatio-temporal data replacement algorithm is described as Algorithm 1:

Algorithm 1 Spatio-temporal data replacement procedure

```
1: Compute the access frequency ratings of objects and return the Hash Map list represented by $HM\langle objID, ratings\rangle$
2: Choose $\mathrm{M}$ objects whose ratings are ranked at top $\mathrm{M}$ from access log list.
3: Determine the hotspot regions in terms of the total access frequency of their containing objects. The returned regions are denoted by $R^h$ , where the $h$ value is computed by $h = MBR(W)/d = \frac{MBR(W)}{\left(\sum_{j=1}^{P} m_j \Big/ \left(s_k \cdot \sum_{i=1}^{N} n_i\right)\right)^{\frac{1}{2}}}$ . The region size is not more than $\xi$ .
4: Compute Hilbert value of hot regions, which is computed by central location $(x_c, y_c)$ of region's MBR, i.e., value of $\mathrm{H}_d(x_c, y_c, t)$ .
5: Distribute the hot regions across memory pools in VegaCache with consistent hashing policy. More specifically, the regions are distributed into memory pool by the consistent hashing value of their Hilbert order, which is computed by function $C\mathrm{H}ash\left(H_d(x_c, y_c, t)\right)\mathrm{mod}\left(\sum_{i=1}^{N} n_i\right)$.
6: The top $\mathrm{M}$ spatio-temporal objects are divided into $h$ regions, and send them to $P$ memory pools that computed by their geographic regions in step5. The memory pool size is $m_j$ and the size of object is $s_k$ ( $\min \le s_k \le \max$ ), thus, the number of objects that resident in a cache pool is ranged from $(m_j/\max)$ to $(m_j/\min)$ . The data objects in memory pool are organized as key-value representation, and the KEY is the Hilbert value of spatial location of object which is computed by function $\mathrm{H}_d(x,y,t)$ .
7: Reevaluate the access frequency ratings periodically, and the in-memory objects will be swapped out while their ratings are decreased to the last $1/N$ . Then select the objects ranked at top $\mathrm{M}/N$ and load them into distributed memory pools of VegaCache, and return to Step 3 for further processing.
8: VegaCache dynamically replaces the caching objects in it and maintains high availability to serve heavy concurrent request workloads. The VegaCache daemons are still on running state to guarantee real-time response.
```

C. Localized Caching Model of VegaCache

The access latency is only dependent on the network bandwidth if the data are resident in VegaCache caching pools. Thus, to reduce the latency, the effective way is to avoid data transfer via network. We propose a localized caching model for VegaCache to balance the tradeoff among access efficiency, data transmission and cache hit rate. Above all, since the access speed of memory is in general orders of magnitude faster than that of disk, the system could provide lower latency access for concurrent requests if there are more data objects are resident in the memory cache. However, the memory pool capacity on a single node is very limited, and the data transfer via network is a time-consuming process because the network latency is even more than disk I/O latency, which may counteract the faster access benefits from the memory cache. Hence, VegaCache should transfer data objects to memory pools of remote nodes as less as possible. We tuned the localized caching models from two facets, which include writing data objects into distributed cache and reading data from memory pools.

As shown in Fig.2, the data objects are firstly written into the local memory pool. If all local memory pools are full, the data will be sent to the remote memory cache by executing data replacement procedure as in Algo.1. Typically, a cluster node has n_i (i indicates node ID) memory pools, and each of them maintains data within a geographically adjacent region. Thus, the concurrent clients who access the adjacent objects could be served by memory pools on the same node, which reduces access latency for it needs not to transfer data from remote nodes over the network. To cope with a single point of failure problem caused by too many clients requesting data from one node, VegaCache distributes the requested data in terms of geographical proximity to avoid single-node overload. Furthermore, if the remote memory pools are not enough, the data items should be swapped out from local memory pool with Least Recently Used (LRU) algorithm, and the latest data objects will be written into local memory pools.

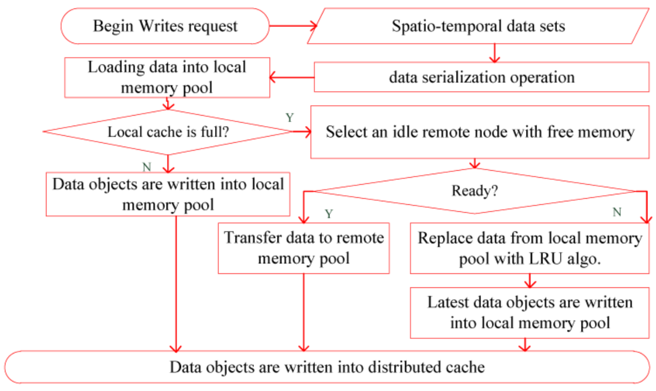
Begin Writes request
Spatio-temporal data sets
Loading data into local memory pool
data serialization operation
Local cache is full?
Y
Select an idle remote node with free memory
N
Data objects are written into local memory pool
Ready?
Y
N
Transfer data to remote memory pool
Replace data from local memory pool with LRU algo.
Latest data objects are written into local memory pool
Data objects are written into distributed cache

Figure 2. The workflow diagram of writing data into VegaCache

As shown in Fig.3, VegaCache improves the read efficiency and guarantees real-time response by retrieving data from the memory cache. It firstly determines the node location and memory pool by computing the *CHash* function. If the data objects are in any memory pool, then return the requested data to clients. Otherwise; if a cache hit miss occurs, VegaCache will load data objects from the underlying disk-based storage system and pre-fetch the adjacent data into distributed cache buffer for later access, finally return data objects to clients.

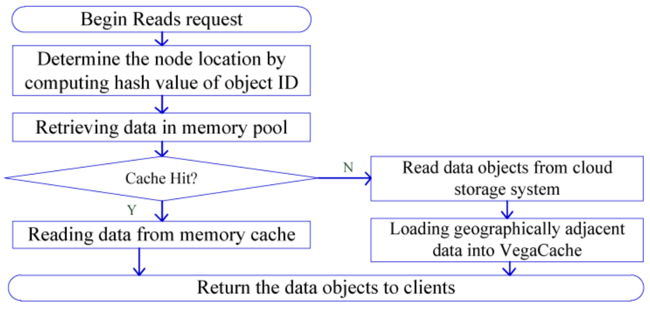
Begin Reads request
Determine the node location by computing hash value of object ID
Retrieving data in memory pool
Cache Hit?
N
Y
Read data objects from cloud storage system
Reading data from memory cache
Loading geographically adjacent data into VegaCache
Return the data objects to clients

Figure 3. The workflow diagram of reading data from VegaCache

Let D be the total dataset, and let S_{mem}, S_{disk} denote the access speed of distributed memory cache and I/O bandwidth of cluster. The access latency can be reduced progressively with localized caching model. If the dataset is smaller than the capacity of VegaCache and the cache hit, the access efficiency is D/S_{mem}. Otherwise, it should access disk I/O and network transfer, and the *Latency Cost* (*LC*) is formulated as (3):

$$LC = (\Psi / S_{mem}) + (D - \Psi)\cdot\left(\frac{1}{S_{disk}} + \frac{1}{B}\right) \qquad (3)$$

Hence, the access performance can be accelerated by $(D-\Psi)\cdot\left(\frac{1}{S_{disk}} + \frac{1}{B} - \frac{1}{S_{mem}}\right)$ with localized caching model. $\because S_{mem} << S_{disk}$, $\therefore$ the network latency is the primary performance bottleneck. Therefore, our scheme could improve the access efficiency greatly with the high cache hit ratio.

III. Performance Evaluation

A. *Experiment Environment*

The experiments are conducted on a 8-node cluster, and each of them has two quad-core Intel CPU 2.13GHZ, 4GB DDR3 RAM, 15000r/min SAS 300GB hard disk. All nodes are deployed with CentOS 5.5, Memcached-1.4.15, SDBMS (i.e., PostGIS-2.0), Hadoop-1.1, HBase-0.94 and LoadRunner. The spatio-temproal dataset is about 450.7 GB, which contains real raster data and vector data.

B. *Cache Hit Ratio of VegaCache*

We evaluated the cache hit ratio of VegaCache with different memory capacity, and each node is allocated 512MB, 1GB, 1.5GB, 2GB, 2.5GB and 3GB memory, i.e., the memory capacity of VegaCache ranges from 4GB to 24GB. The benchmark sends data requests randomly across whole geographic region. As shown in Table II, the cache hit ratio of VegaCache could be improved with larger memory capacity, and it increases from 52.87% to 98.53% when the distributed cache size is enlarged from 4GB to 24GB. Additionally, with the data volume becoming larger, our scheme could improve the cache hit ratio by adding more physical memory and cluster nodes, which guarantees most of data objects resident in the distributed cache and hence reduce disk I/O latency.

TABLE II. Cache Hit Ratio of VegaCache

Test item	**Memory Capacity of VegaCache (GB)**					
	4GB	***8GB***	***12GB***	***16GB***	***20GB***	***24GB***
Cache Hit Ratio (%)	52.87	68.92	79.21	87.33	93.06	98.53

C. *Access Throughput Performance*

To confirm the effectiveness of supporting concurrent users, we evaluate the access throughput in terms of Transactions per Second (TPS) measurements and use Loadrunner to emulate two hundred users requesting for geospatial data service. The access patterns are determined by read and write ratio, which ranges from 10%-100%. As shown in Fig.4, the throughput performance of SDBMS with the local cache is about 1.58-2.37 times better than that of SDBMS without cache, and the TPS of VegaCache is 12.78-17.39 times more than that of SDBMS with local cache. Moreover, the throughput of VegaCache increases more progressively with larger memory capacity. Its average performance is improved by 1.78-2.98 times better when the memory capacity is enlarged from 4GB to 24GB.

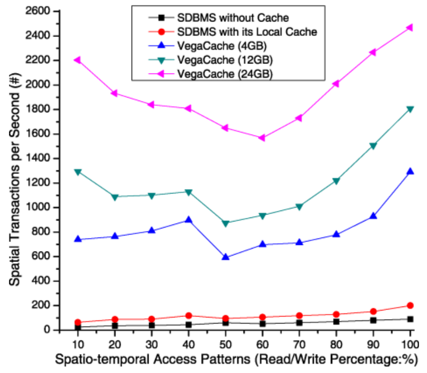

Figure 4. Access throughput performance

D. *Response Performance*

We evaluated the response performance with different numbers of concurrent users, and VegaCache is compared to SDBMS with and without cache during experiments. With real geospatial application workloads, the average response time is logged under different circumstances.

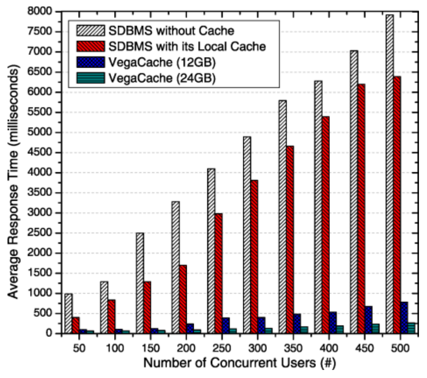

Figure 5. Average response performance

As shown in Fig.5, the average response time becomes longer as the number of concurrent users increases. The response performance of SDBMS is very low whether there is a cache or not, and the average response time of SDBMS with cache is 23.8%-137.2% less than that of without cache, which is about 400-6389 milliseconds. On the contrary, the average response time of VegaCache is kept stable at about 68-580ms, and the response performance is even better with a larger memory, which is improved by 1.5 orders of magnitude than SDBMS. Thus, VegaCache could respond to a large number of concurrent clients in milliseconds, which provides real-time access efficiency for online geospatial applications.

E. *Data Object Retrieval Efficiency*

We evaluated the data retrieval performance by retrieving ten groups of data within different geographic regions. The regions are equal to $5k\left(k=1,2,3,\cdots,10\right)$ *percent* of MBR of map layer, which are denoted as $R_k\left(k=1,2,\cdots,10\right)$. As shown in Fig.6, VegaCache outperforms SDBMS by about 4.47-9.76 times in all test cases. Moreover, since the cache hit ratio will be improved with larger memory pool, the average data retrieval performance of VegaCache is improved by about 13.6%-66.9% with the cache size increased by 4GB, e.g., VegaCache has achieved the retrieval efficiency by 2.35-3.71 times better when its capacity increases from 12GB to 24GB. VegaCache performs at a relatively stable level no matter what size of the requested regions. In contrast, the retrieval performance of SDBMS is reduced heavily while accessing larger region data. Additionally, the experiment results show that VegaCache has good scalability, and its retrieval performance can be improved by adding more cluster nodes and physical memory resources.

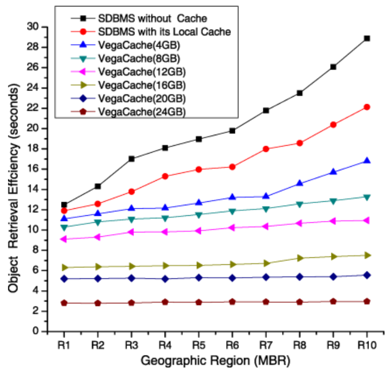

Figure 6. Data Retrieval Efficiency

IV. RELATED WORK

With the rapid growth of data volume and users, the real-time access is a significant challenging problem in online geospatial applications. There are various prior works on spatial data management. HiSbase is a histogram-based peer-to-peer data organization approach with in-memory dynamic hashing table for e-science communities [3]. Canim et al. propose a SSD buffer pool extension for database systems to improve the random I/O access performance [4]. TiMR [5] is a framework of online temporal analytics processing over big data for web advertising. Geocrowd [6] solves the maximum task assignments problem to answering location-based mobile queries with a spatial crowdsourcing platform. MOBB [7] is a synchronous parallel processing of big data analytics method that can reduce data transfer delay in federated clouds. Panda [8] targets long-term query predication for efficient support of a wide variety of predictive spatial queries, and it achieves a low response time. Do, et al, presents systematical designs of using Solid State Disk (SSD) technique to improve the performance of DBMS buffer manager [9], which gains great performance improvements over hard disk configurations. CloST is a scalable Hadoop-based storage system for big spatio-temporal data, and it can avoid scanning the whole dataset while processing range query [10]. Most of these spatio-temporal data management methods are tuned for disk-based storage system, whereas they failed to consider the memory caching scheme for real-time access.

V. CONCLUSION

We have designed an efficient and progressive spatio-temporal data caching scheme named VegaCache to meet the requirements of online geospatial applications. Our scheme takes advantage of the distributed memory cache and cloud platform. It consists of three key techniques to guarantee real-time access, i.e., scalable architecture of caching systems, data replacement model and localized caching models. The capacity of VegaCache could be enlarged by simply adding more cluster nodes and physical memory resources. Moreover, VegaCache could improve its access efficiency when all of the online data are resident in the memory pool and minimize data transmission across the cluster. Besides, we confirm the effectiveness of our scheme with comprehensive experiments on real-deployed geospatial applications. The results show that VegaCache has good scalability, high availability, real-time response and low-latency access efficiency. In the future, we will focus on real-time spatio-temporal queries with hybrid storage technologies of memory and SSD.

References

[1] D. J. Coleman, Y. Georgiadou, and J. Labonte, "Volunteered Geographic Information: The Nature and Motivation of Produsers", International Journal of Spatial Data Infrastructure Research, vol.4, 2009, pp.332-358.

[2] L. George, P. Dieter, "Collaborative Geospatial Feature Search", Proc. Int. Conf. on Advances in Geographic Information Systems, SIGSPATIAL'12, pp.169-178, 2012.

[3] T. Scholl, B. Bauer, B. Gufler and et al. "HiSbase: Histogram-based P2P Main Memory Data Management", Proc. Int. Conf. on Very Large Data Bases, VLDB'07, pp.1394-1397, 2007.

[4] M. Canim, G. A. Mihaila and et al. "SSD Bufferpool Extensions for Database Systems", VLDB Journal, vol.3(2), 2010, pp.1435-1446.

[5] B. Chandramouli, J. Goldstein, S. Duan. "Temporal Analytics on Big Data for Web Advertising", Proc. Int. Conf. on Data Engineering, ICDE'12, pp.90-101, 2012.

[6] L. Kazemi, C. Shahabi. "Geocrowd: Enabling Query Answering with Spatial Crowdsourcing", Proc. Int. Conf. on Advances in Geographic Information Systems, SIGSPATIAL'12, pp.189-198, 2012.

[7] G. Jung, G. Nathan, T. Mukherjee, "Synchronous Parallel Processing of Big-Data Analytics Services to Optimize Performance in Federated Clouds", Proc. Int. Conf. on Cloud Computing, pp.811-818, 2012.

[8] A. M. Hendawi, M. F. Mokbel, "Panda: A Predictive Spatio-Temporal Query Processor", Proc. Int. Conf. on Advances in Geographic Information Systems, SIGSPATIAL'12, pp.13-22, 2012.

[9] J. Do, D. Zhang, J. M. Patel and et al. "Turbocharging DBMS Buffer Pool Using SSDs", Proc. Int. Conf. on Management of Data, SIGMOD'11, pp. 1113-1124, 2011.

[10] H. Tan, W. Luo, L. M. Ni, "CloST: A Hadoop-based Storage System for Big Spatio-Temporal Data Analytics", Proc. Int. Conf. on Information and Knowledge Management, CIKM'12, pp.2139-2143, 2012.