International Journal of Grid and Distributed Computing
Vol. 9, No. 12 (2016), pp.127-142
http://dx.doi.org/10.14257/ijgdc.2016.9.12.12

Design and Evaluation of a High-concurrency Web Map Tile Service Framework on a High Performance Cluster

Bo Cheng[1], Xuefeng Guan[2*]

[1,2]*State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China*
[1]*chengbo@whu.edu.cn*, [2*]*guanxuefeng@whu.edu.cn*

Abstract

Web Map Tile Services (WMTS) have been widely used for quick and convenient sharing of geospatial information. In practice, when streaming requests to servers increase in scale, unacceptable response times and service unavailability might result. To address this scalability problem, we implemented a scalable WMTS framework on a high performance cluster (HPC), enabling the realization of elastic deployment as the client users grow in number. This scalable and high-concurrency WMTS is built totally with open-source software, including Nginx, GeoWebCache, and MongoDB. In this architecture, Nginx acts as a powerful load balancer for routing client requests; GeoWebCache is customized to publish the required WMTS and process client requests; while MongoDB is used to store the large volume of tile images in the HPC. Evaluation experiments were carried out to assess the efficiency and scalability of our WMTS system, using one synthetic workload. Experimental results illustrate that this distributed WMTS framework can achieve about 15% performance improvement when the service nodes are increased with a 0.5~2s reduction in the load time and a 5~10MB increase in network throughput.

Keywords*: Web Map Tile Service, High-Concurrency, Web server cluster, Load Balancer, NoSQL*

1. Introduction

Tile-based web mapping services are becoming increasingly popular for visualizing large-scale geospatial data on the Web. The Open Geospatial Consortium (OGC) has released a specification, the Web Map Tile Service (WMTS), for tile-based web mapping services [1]. Instead of creating a new image for each request, a WMTS returns small pre-generated images to users. WMTS provides an open-source alternative to proprietary web mapping services, such as Google Maps, or Microsoft Bing Maps.

WMTS has been adopted in many applications for online cartography, location-based services, and social networking because of its efficiency. There are a number of open source software packages, such as GeoWebCache, MapServer, TileCache that provide open-source implementations to publish spatial data as a WMTS. Current WMTS applications however, have scalability problems. WMTS servers often cannot handle massive concurrent user requests. When client users increase dramatically, the torrent of these requests places overwhelming pressure on the web server where the WMTS is deployed, causing significant response delays and serious performance degradation. Therefore, it is necessary to design an efficient high-concurrency WMTS architecture that can automatically scale to the volume of user requests.

This paper addresses the scalability problem of WMTS applications and introduces a powerful WMTS system on a high performance cluster (HPC). The architecture of this high-concurrency WMTS is segmented into three tiers, including a load balancer, the WMTS servers, and a distributed database, respectively. These three tiers are all built

ISSN: 2005-4262 IJGDC

with open-source software, including Nginx, GeoWebCache, and MongoDB. Three evaluation experiments were carried out to test the efficiency and scalability of our proposed high-concurrency WMTS with a synthetic workload named HELP (Hotspot/think-timE/Length/Path). This synthetic workload can simulate thousands of users simultaneously browsing a target WMTS map. Our experimental results demonstrate that the proposed WMTS framework can effectively improve system throughput and scalability by increasing the number of active middle-tier service nodes. When the number of the middle-tier service nodes increases, the WMTS framework reduces the page load time by 0.5~2s, and boosts network throughput by 5~10MB.

The rest of the paper is organized as follows. Section 2 presents work related to tile-based web mapping services and high-concurrency web systems. Section 3 introduces the high-concurrency WMTS framework design. Section 4 presents evaluation results and a discussion of the experiments. Section 5 closes the paper with our conclusions.

2. Background and Related Work

2.1. Tile-based Web Mapping Services

To solve the interoperability problem and facilitate better geospatial information sharing, several web mapping standards have been developed by a variety of organizations. The OGC is leading the development of several sequential standards to support geospatial interoperability, e.g. Web Map Service (WMS) [2], Web Feature Service (WFS) [3], and Web Coverage Service (WCS) [4]. Most of the standards leverage geospatial Web services (GWS) as building blocks to provide access to geospatial information through Hypertext Transfer Protocol (HTTP)-based queries.

Currently, the WMS standard has been widely accepted as an open standard for map visualization and is implemented by the majority of GIS software vendors. It standardizes the way in which web clients request maps. Clients can request maps from a WMS by specifying map layers and providing parameters such as the size of the returned map and the spatial reference system. When the WMS server receives the client requests, it will generate the requested map images on the fly to realize almost instant zoom and pan functions. However, this flexibility comes at a price: WMS cannot scale to requests due to inefficient on-the-fly generation.

To improve the performance of WMS, three additional types of web mapping services were proposed, WMS-Cache (WMS-C), Tile Map Service (TMS), and WMTS. The first two services were proposed by Open Source Geospatial Foundation (OSGeo), and the last one was proposed by OGC.

WMS-C optimizes the delivery of map imagery across the Internet. WMS-C defines a constrained profile for OGC WMS that permits servers to improve image generation, and allows tiles to be cached at hot map locations. A WMS-C service nonetheless can only directly deliver images for bounding boxes aligned to a given rectangular grid, and only at predetermined fixed scale levels. The WMS-C service is free to return an exception or a redirect if it receives a WMS request that is not WMS-C compliant.

The TMS specification was the work of a loose community of participants interested in client/server mapping solutions using multi-resolution image pyramids. The TMS renders spatial data into cartographic tiles at fixed scales. These predefined tiles are provided via a REST interface, starting with a root resource describing available layers, then map resources with a set of scales, then scales holding sets of tiles.

The OGC's WMTS follows the OSGeo's TMS specification with a tiling model to describe the predefined images. A tiling model divides the space into a fixed tile matrix set, illustrated by Figure 1. A tile matrix is a collection of tiles at a fixed map scale and usually occupies a specific level in the pyramid. The WMTS server can simply return the appropriate pre-generated image (e.g., PNG or JPEG) files to client users. WMTS

supports multiple service interfaces— KVP, REST and SOAP. The WMTS is backed by an official standards body, OGC, and more widely spread than the former two methods for internet mapping applications.

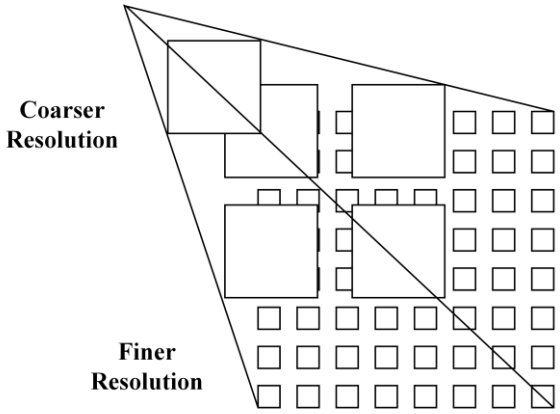

Figure 1. The Illustration of the Tiling Model in the WMTS Specification

2.2. High-concurrency Web Systems

High-concurrency web systems have been an active field of research for many years [5-11]. Higher scalability and more availability of Web servers are required as the traffic on the Internet has been increasing dramatically over the last few years. As a single server can only handle a limited amount of requests and cannot scale out with demand, the adoption of a cluster-based web server (web cluster) is one of the feasible solutions to build powerful web applications.

A web cluster is usually built on a cluster of commodity servers, in which one is front-end server and the others are back-end servers. The front-end server is a load balancer (a dispatcher), which routes inbound requests to the different back-end servers and makes the cluster to act as a virtual server. A complete survey of web cluster architectures has been compiled in [12-13]. The web cluster can be divided into two types, layer-4 or layer-7, according to the OSI protocol stack where the dispatcher works. The web cluster also can be divided into another two types, one-way or two-way, according to the flow direction of outbound responses. In one-way architectures, the responses for client requests flow through the direct connection between the selected back-end server and the client, while in two-way architectures, both requests and responses pass through the dispatcher.

In the layer-4 web cluster, the dispatcher determines the target back-end server when the client starts to establish the TCP connection, before sending out the HTTP request. The layer-4 dispatcher works at TCP/IP level. The dispatcher maintains a binding table to associate each client TCP connection with the target server. The layer-4 dispatcher forwards packets from clients to the back-end server without knowing the detailed request information. Thus the packet-forwarding algorithms adopted by the dispatcher are also called content-blind dispatching policies. There are many packet-forwarding mechanisms in the layer-4 dispatcher, such as Network Address Translation (NAT) and IP Tunneling [14]. Currently NAT and IP Tunneling techniques can be integrated into the Linux kernel through the Linux Virtual Server (LVS) Project [15].

Conversely, in a layer-7 web cluster the dispatcher examines the HTTP request content and selects the target server after it has established a complete TCP connection with the client. The layer-7 dispatcher usually forwards the incoming packets to the back-end server according to the client request information. Thus the algorithms used by the layer-7 dispatcher are also called content-aware dispatching policies. TCP gateway, TCP handoff

[16] and TCP splicing [17] are threes typical dispatching mechanisms that allow the front-end server to use content-aware distribution algorithms. In practice, a reverse proxy is widely used as a TCP gateway to implement layer-7 content dispatching.

The main advantage of layer-7 dispatching methods over layer-4 solutions is the possibility of using content-aware dispatching algorithms at the front-end dispatcher. The content-aware dispatching algorithm can achieve higher cache-hit rates and finer granularity of load-balance. However, the layer-7 dispatching mechanisms introduce additional processing overhead to the dispatcher and cause some inefficiency due to packet parsing work.

3. The Architecture of High-concurrency WMTS

The architecture of the proposed WMTS system is formed with three types of components, load balancer, WMTS server, and distributed database. All the components are built with open-source software, including Nginx, GeoWebCache, and MongoDB.

3.1. Reverse Proxy as a Load Balancer

Because of the uneven spatial and temporal request distribution of geospatial web services, a layer-7 dispatching solution, reverse proxy dispatching, is adopted to achieve better load-balance for the proposed WMTS. One reverse proxy is placed in front of a Web server or clustered servers to dispatch incoming requests. Reverse proxy parses the request parameters, routes incoming requests to the back-end web server, retrieves result information from the web server and then forwards it to the user. A reverse proxy can also be deployed to handle SSL acceleration, intelligent compression, and caching. Different from a traditional forward proxy located near clients, a reverse proxy acts as an intermediary for its associated servers. The function of one reverse proxy is illustrated in Figure 2.

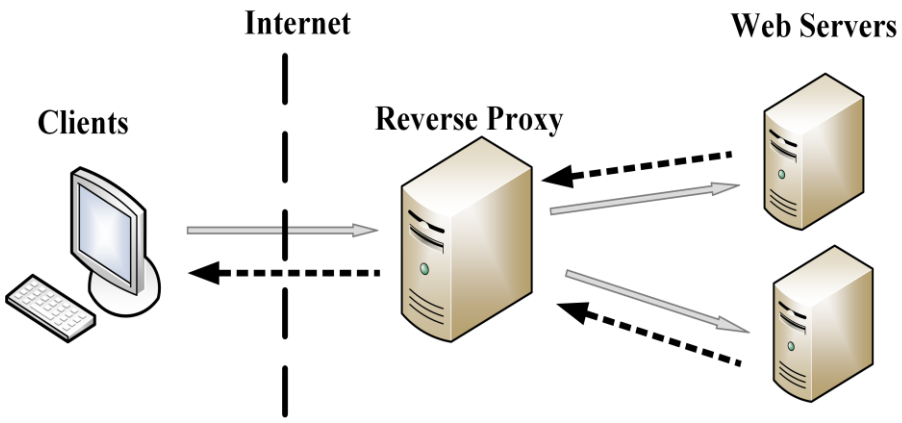

Figure 2. The Functionality Illustration of one Reverse Proxy

Many web servers can be deployed as a reverse proxy, such as Nginx [18], HAProxy [19], and Apache, etc. Among them, Nginx is one of the most popular open source web servers on the Internet. It can be also deployed as a reverse proxy or an IMAP/POP3 proxy server. Nginx can process 100k+ concurrent connections per server. In the proposed high-concurrency WMTS architecture, Nginx is deployed as the front-end server and acts as a layer-7 load balancer.

3.2. Web Map Tile Service

GeoWebCache is a Java web application used to cache map tiles coming from a variety of sources, from on-the-fly generation by WMS to pre-generated tile storage, e.g. WMTS. It implements various service interfaces (including WMS-C, WMTS, TMS, Google Maps KML, and Virtual Earth) in order to accelerate and optimize map image delivery. It can run as a standalone application or be integrated within a web map server, e.g. GeoServer. Figure 3 shows the process workflow of client requests in GeoWebCache.

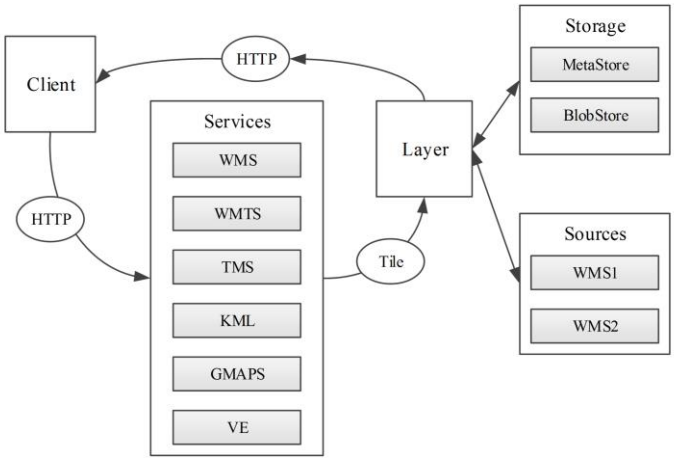

Figure 3. The Workflow of Processing Client Requests in GeoWebCache

In the proposed high-concurrency WMTS architecture, GeoWebCache is used to provide an implementation for the required WMTS. One GeoWebCache instance is deployed in each mid-tier server; all the instances work together to increase the whole tile throughput.

3.3. Tile storage with a NoSQL database

Unlike traditional relational databases, a NoSQL database provides a mechanism for storage and retrieval of data that use looser consistency models in order to achieve horizontal scaling and higher availability. The major difference between traditional relational and NoSQL databases is that relational databases have rigid schemas while NoSQL databases are schema-free. In a NoSQL database, data can be stored without defining a rigid database schema. The data stored are normally self-descriptive. This schema-free feature provides immense flexibility for users.

NoSQL database systems usually employ a distributed and fault-tolerant architecture. The database automatically stores data in a redundant manner across servers without user participation. In this way, the system can easily be scaled out by adding more servers, and subsequently the failure of one server can be tolerated. This type of database typically scales horizontally and can manage massive amounts of data.

A variety of NoSQL databases have been developed by industrial companies and open-source practitioners. Usually NoSQL databases are classified according to the data model they use to store the data, and can be categorized as wide column stores (HBase, Cassandra, etc), document stores (MongoDB, CouchDB, etc), key–value stores (DynamoDB, Redis, MemcacheDB, etc), and graph databases (Neo4J, InfiniteGraph, etc) [20].

MongoDB is one of the most widely used NoSQL databases currently available in the marketplace. Besides the common features of NoSQL databases, MongoDB stores data in the format of JSON-style documents and supports full geospatial indexing on the location

attribute of stored documents. The core components of one MongoDB database are: *mongos* and *mongod*, illustrated in Figure 4. The *mongos* is a routing service that processes query from the clients and determines the location of target data in the shared cluster. The *mongod* is the primary daemon process for the MongoDB database. It handles data requests, manages data format, and performs background management operations.

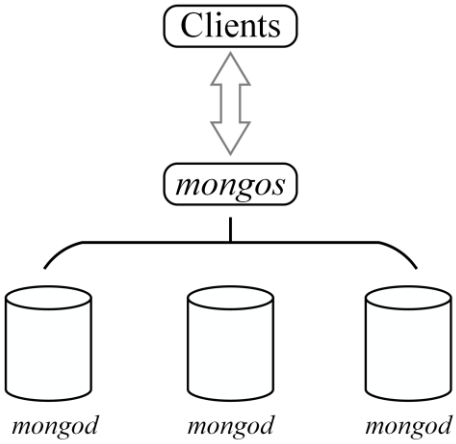

Figure 4. The Core Components of One Distributed MongoDB Database

In the proposed high-concurrency WMTS architecture, MongoDB provides storage space for large volume of WMTS tiles. One WMTS tile stored in the MongoDB JSON format is illustrated in Table 1. A 2D spatial index is ensured through the "location" attribute.

Table 1. The WMTS tile Stored in the MongoDB JSON Format

```
mongos> db.EPSG_900913_ 8.findOne()
{
"_id" : ObjectId("4f0008f8db7ecff1831b15f3"),
"name" : 10313,
"level" : 8,
"location" : {
            "col" : 56,
            "row" : 174
      },
"hitTimes" : 0,
"CacheTime" : null,
"content" : BinDta(0,"/9j/4AAQSkZA****WxgULUpOx//Z")
}
```

3.4. The Architecture of High-concurrency WMTS

The architecture of a high-concurrency WMTS system can be segmented into three tiers of services. The three tiers, from top to bottom, are load balancer, WMTS servers, distributed tile database. In this architecture, Nginx acts as a powerful load balancer for client requests; GeoWebCache is customized to publish the required WMTS and process user requests; while MongoDB is used to store massive map tiles in the HPC. The whole architecture is illustrated in Figure 5.

A prototype system was built following the proposed architecture to provide the high-concurrency WMTS. The prototype system was constructed on a Linux cluster running

CentOS 6.2. In this cluster, one node is configured as the load balancer running Nginx. A multi-node MongoDB databases were built to store the WMTS tiles. The other discrete nodes with GeoWebCaches were added to the WMTS service tier for processing client requests.

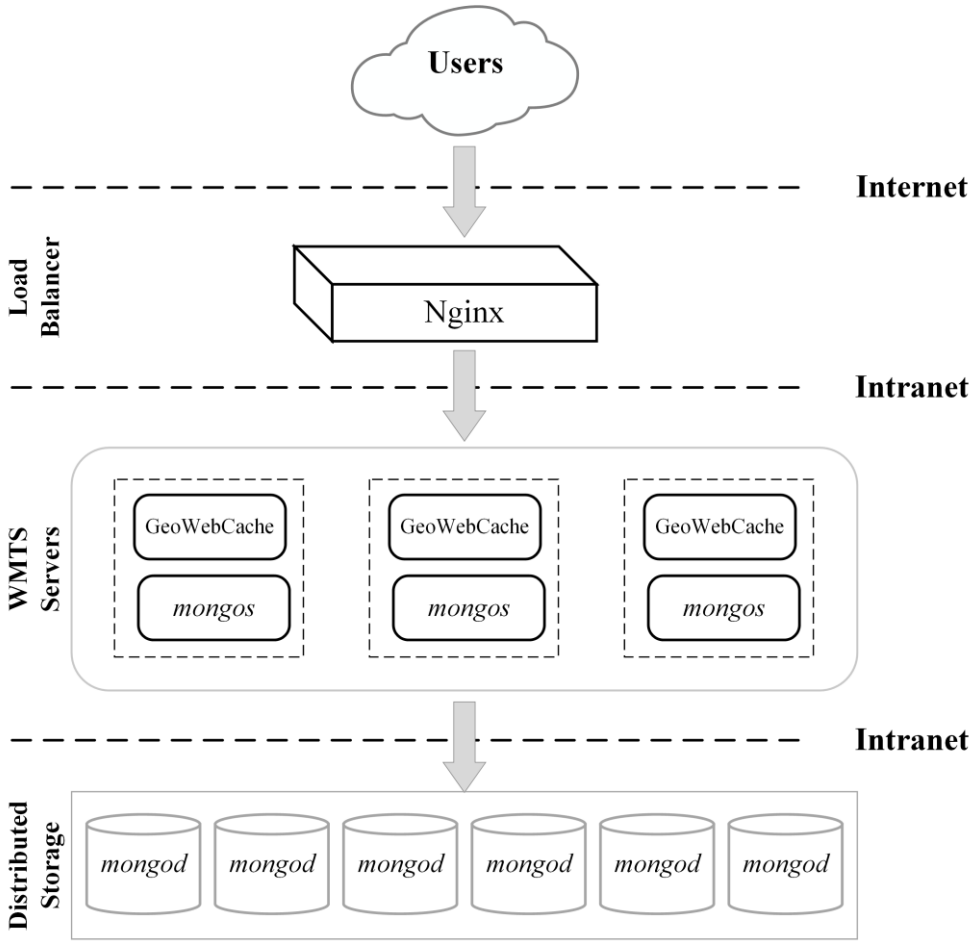

Figure 5. The Architecture of the Proposed High-concurrency WMTS System

In the proposed high-concurrency WMTS architecture, GeoWebCache is used to provide an implementation for the required WMTS. One GeoWebCache instance is deployed in each mid-tier server; all the instances work together to increase the whole tile throughput.

3.5. The Storage Schema of Massive Map Tiles

In the WMTS tiling model, these smaller image tiles are usually organized into a hierarchical tile pyramid, in which the resolution of one level is half of one upper level. For simplicity, the tiles of each pyramid level are originally designed to be stored in one MongoDB table. However, the number of image tiles increases dramatically as the pyramid level increases, especially for the bottom levels. If the whole set of tiles in the bottom level are all stored in one MongoDB table and the record number exceeds millions, the query performance will deteriorate very quickly.

To solve this problem, a modified storage schema is proposed as follows. The basic idea of this method is to decrease the tile number stored in one MongoDB table. For the upper levels(e.g. 1~12), the tile number of each level is small and can be stored in one table; for the bottom levels(e.g. 13~15), the tile number of each level is too big and then

this level is decomposed into several sub-levels with a quad-tree structure. The number of the decomposed sub-levels can be calculated with 4^(current level -12). The tiles that belong to one sub-level are organized into a MongoDB table. This decomposition method can control the tile number to a predefined number. The conversion from tile position (level, col, row) in the pyramid to record position (table_name, record_key) in the storage table is seen in Table 2.

Table 2. The Conversion from Tile Position in the Pyramid to Record Position in the Storage Table

```
Position_Convertion(level, col, row, table_name, record_key )
{
table_name = 'EPSG_900913_';
if (level < 13)
 table_name is appended with level num;
 record_key = hash(col, row);
else
 table_name is appended with level num;
 num_tiles = 4^ level;
 num_subs = 4^( level -12);
 sub_col = col / (2^12);
 sub _row = row / (2^12);
 table_name is appended with sub_col num;
 table_name is appended with sub_row num;
 record_key = hash(col, row)
end if
return table_name, record_key;
}
```

4. Experiments and Discussion

4.1. Experimental Environment and Datasets

All the experiments in this paper are conducted using a cluster of servers. The cluster consists of 13 machines configured as follows: one machine is used as the load balancer and connected with three WMTS servers; all the three WMTS servers are connected to a dedicated MongoDB database formed with six nodes; three additional servers are dynamically added to evaluate the framework scalability. The detailed configuration of the cluster is listed in Table 3. All the servers are directly connected by a dedicated 1Gb Ethernet link.

Table 3. The Detailed Configuration of the Cluster Servers

Type	**Load Balance Server**	**Web Severs**	**Database Servers**
Number	1	3	6
CPU	Intel Xeon E5-2665 (2*8 cores, 2.40GHz each core)	Intel Xeon E5-2620 (2*6 cores 2.00GHz each core)	Intel Xeon E5-2620 (2*6 cores 2.00GHz each core)
Memory	32 GB(1333 MHz)	32 GB(1333 MHz)	32 GB(1333 MHz)
Hard disk	250 GB(15000rpm, SAS)	500 GB(15000rpm, SAS)	500 GB(15000rpm, SAS)
Kernel	2.6.32-220.el6.x86_64	2.6.32-220.el6.x86_64	2.6.32-220.el6.x86_64

OS	CentOS 6.2	CentOS 6.2	CentOS 6.2
Software	Nginx-1.2.4	Tomcat-7.0.32	MongoDB -x86_64-2.2.1

The kernel parameters of each server's operating system are shown in Table 4, which are tuned from the Intel's web server benchmark work [21].

Table 4. The Kernel Parameters of Operating System

```
fs.file-max=5000000
net.core.netdev_max_backlog=400000
net.core.optmem_max=10000000
net.core.rmem_default=10000000
net.core.rmem_max=10000000
net.core.somaxconn=100000
net.core.wmem_default=10000000
net.core.wmem_max=10000000
net.ipv4.conf.all.rp_filter=1
net.ipv4.conf.default.rp_filter=1
net.ipv4.tcp_congestion_control=bic
net.ipv4.tcp_ecn=0
net.ipv4.tcp_max_syn_backlog=12000
net.ipv4.tcp_max_tw_buckets=2000000
net.ipv4.tcp_mem=30000000 30000000 30000000
net.ipv4.tcp_rmem=30000000 30000000 30000000
net.ipv4.tcp_sack=1
net.ipv4.tcp_syncookies=0
net.ipv4.tcp_timestamps=
net.ipv4.tcp_wmem=30000000 30000000 30000000
```

The request processing in Nginx is event-driven and Nginx can spawn several worker processes to increase the processing throughput. The important configuration parameters include the number of worker processes to spawn, and the maximum number of requests to dispatch to each child process. The core configuration of the Nginx in the proposed high concurrency WMTS architecture is listed in Table 5.

Table 5. The Core Configuration of the Nginx

```
worker_processes  16;
worker_rlimit_nofile 102400;
events {
    use epoll;
    worker_connections  102400;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    tcp_nopush     on;
    tcp_nodelay     on;
    keepalive_timeout  65;
    upstream upspoll {
     server 192.168.0.17:8080;
     server 192.168.0.18:8080;
```

```
    server 192.168.0.19:8080;
  }
  server {
    listen       80;
    server_name  localhost;
    location / {
      root   html;
      index  index.html index.htm;
    }
    location /geospeed/
    {
     proxy_pass http://upspoll;
     proxy_set_header Host $host;
    }
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
      root   html;
    }
  }
}
```

The maximum memory of the Java virtual machine (JVM) for Tomcat was increased from 256MB to 2048MB. The thread pool of Tomcat for incoming request connections was also opened and the pool size set to 1000.

The experimental tile dataset was from the Landsat image of the contiguous United States. It covers the extent from 20 to 48 degree in latitude and from 72 to 132 degree in longitude. The whole data has 15 levels and contains 91330776 tiles. The average size of one tile is about 12KB and the total data size is about 1TB. The detailed information about the tile dataset is listed in Table 6.

Table 6. The Tile Information in the Different Pyramid Levels

Level	Rows	Columns	Tiles	Size(GB)	10000-tile query time
1	1	2	2	2.1431E-05	52
2	2	2	4	4.3293E-05	67
3	3	3	9	9.76667E-05	76
4	4	6	24	0.000258568	82
5	7	12	84	0.000932785	88
6	13	22	286	0.003370382	91
7	26	43	1118	0.013629335	94
8	50	86	4300	0.051477337	92
9	99	172	17028	0.196899891	92
10	197	342	67374	0.805088253	95
11	393	683	268419	3.225658244	96
12	785	1366	1072310	13.07745007	97
13	1568	2732	4283776	46.14761879	133
14	3136	5462	17128832	192.7241897	173
15	6270	10923	68487210	720.786522	203

One example tile request URL is listed as follows; the WMTS map displayed by the Openlayers client can be seen in Figure 6.

http://192.168.0.30:8080/geospeed/service/wmts?request=GetTile&version=1.0.0&layer=usa&style=default&format=image/jpeg&TileMatrixSet=EPSG:900913&TileMatrix=EPSG:900913:3&TileRow=x&TileCol=y.

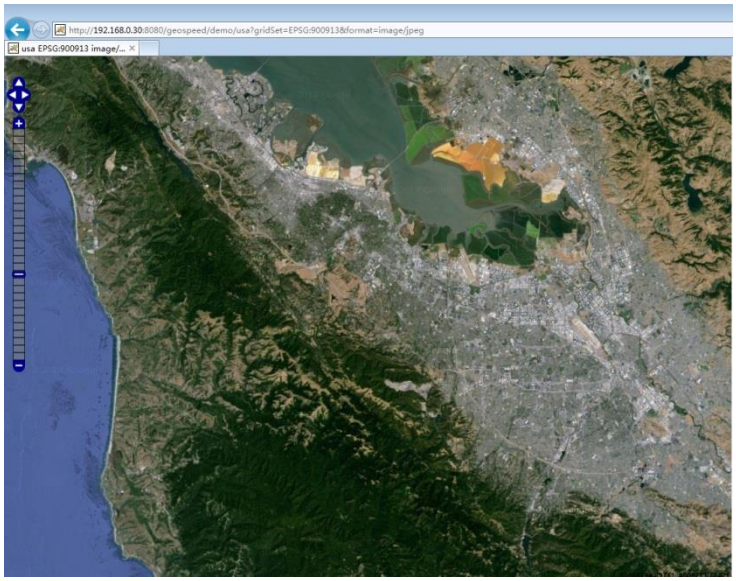

Figure 6. The WMTS Map Displayed by the Openlayers Client

4.2. Experimental Results and Discussion

Three experiments were carried out to evaluate the performance and scalability of our proposed high-concurrency WMTS with a synthetic HELP workload [22]. This HELP workload can simulate how users browse a WMTS map and statistically characterizes complete map navigation behaviors. The performance representation displayed in the HELP workload are more accurate than the traditional workloads for evaluating WMTSs such as repeated static URLs, randomized requests, and access log replay.

All experiments used a 20s ramp-up period and 10min stable period for measurement data collection. The metrics used to quantify performance include server-centric metrics and network-centric metrics. The server-centric metrics are page load time and tile throughput. Page load time is the whole time in which a single user issues a map page request; one map page usually contains up to 20~30 map tiles that are shown on a rectangular computer screen, the system processes the request tiles on the map page, and the client receives all the return tiles. Tile throughput is the number of map tiles the system can process in one second. The network-centric metric only includes network throughput.

4.2.1. Evaluation of the high-concurrency WMTS system performance: The first evaluation experiment was carried out to evaluate the performance of our implemented WMTS system. The evaluation results for the HELP workload are shown in Figure 7~9.

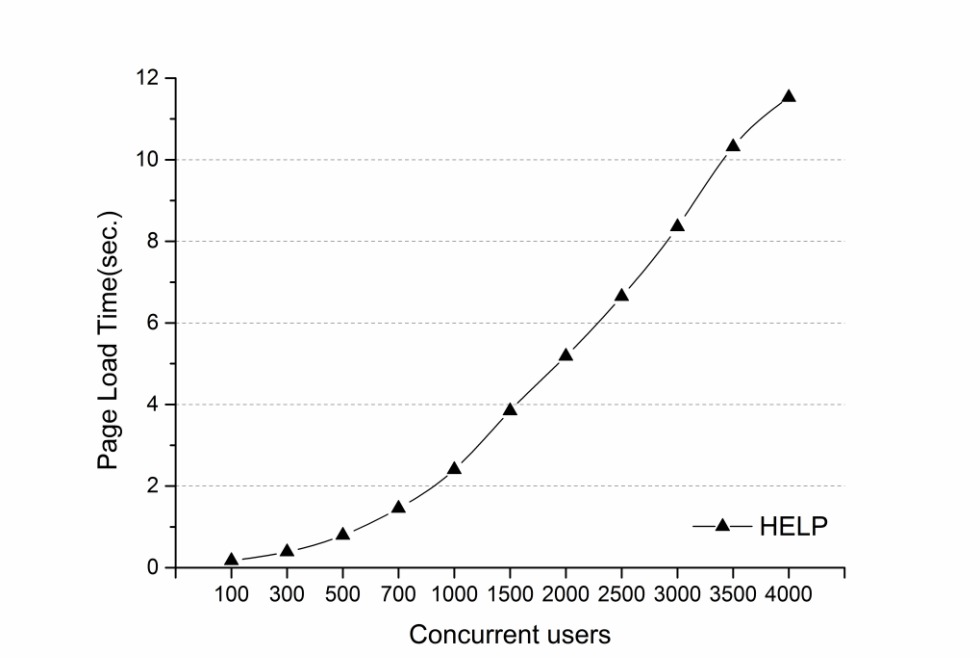

Figure 7. The Page Load Time as Represented by the HELP Workload

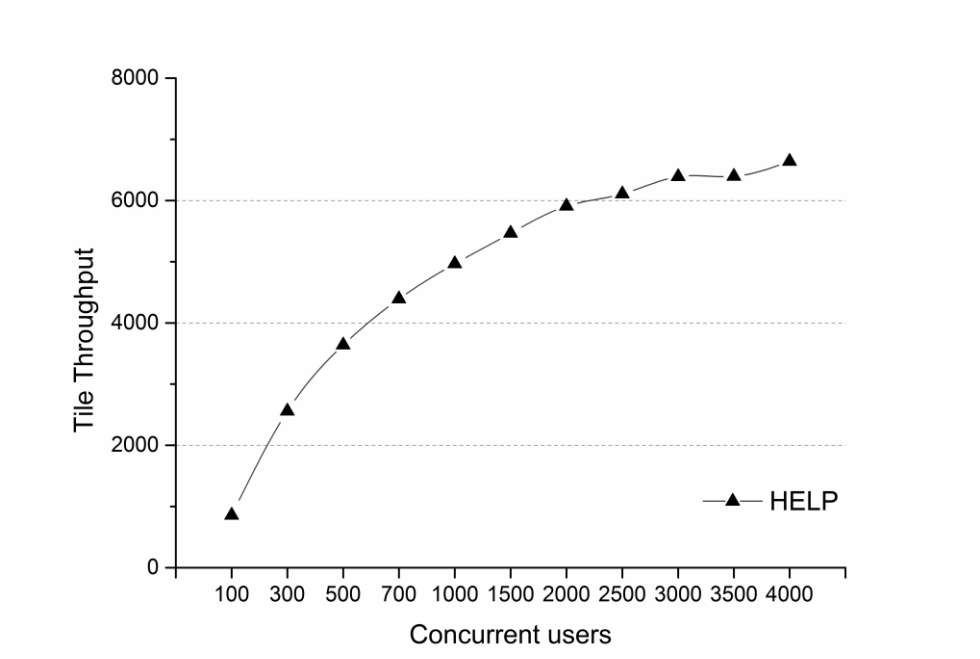

Figure 8. The Tile Throughput as Represented by the HELP Workload

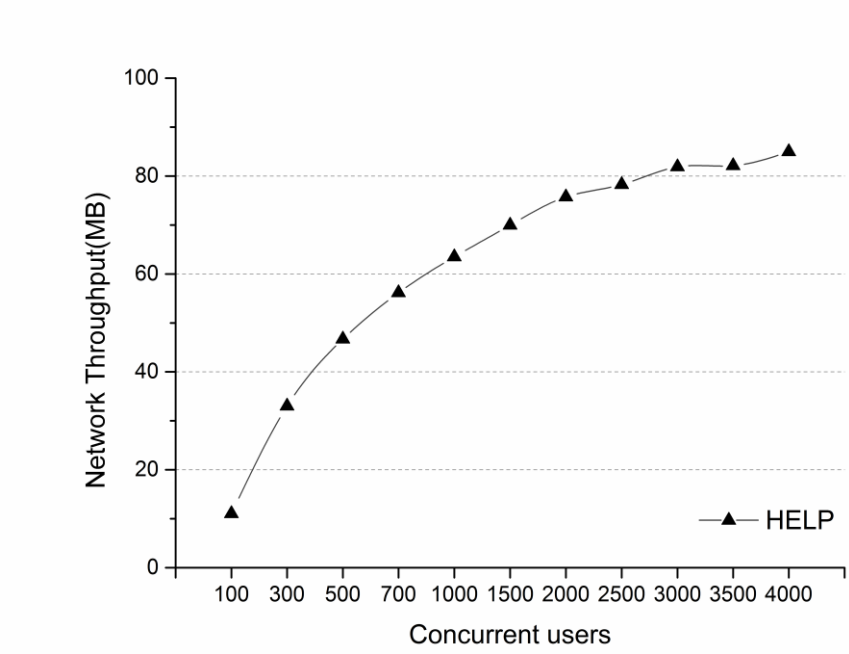

Figure 9. The Network Throughput as Represented by the HELP Workload

As illustrated in Figure 7, the page load time from the HELP workload increases very slowly when the user numbers are less than 1000. When there are less than 1000users, the response time of the HELP workload is nearly 2 seconds. When the number of users increases to 4000, the page load time climbs dramatically, to about 12 seconds.

The tile throughput and network throughput show a similar pattern. In the experimental dataset, the average tile size is about 12KB, and therefore the total network throughput nearly equals the tile throughput multiplied by the average tile size. In Figure 7, when the number of concurrent users exceeds 1000, the page load time for the HELP load increases dramatically, but the network throughput graph (Figure 9) shows that the maximum network throughput is about 80MB, and thus does not reach the upper bound of the network bandwidth; the effective network bandwidth of 1Gb Ethernet is about 100~110MB. This result shows that when the concurrent users exceed 1000, the accumulated requests nearly exhaust the capability of our implemented WMTS system.

4.2.2. Experiments with more WMTS nodes: In the web service layer of the implemented WMTS system, we increased the number of GeoWebCache nodes from three to six. This experiment was carried out to evaluate whether the performance of the high-concurrency WMTS could be boosted by increasing the service nodes. The results for page load time and network throughput are shown in Figure 10.

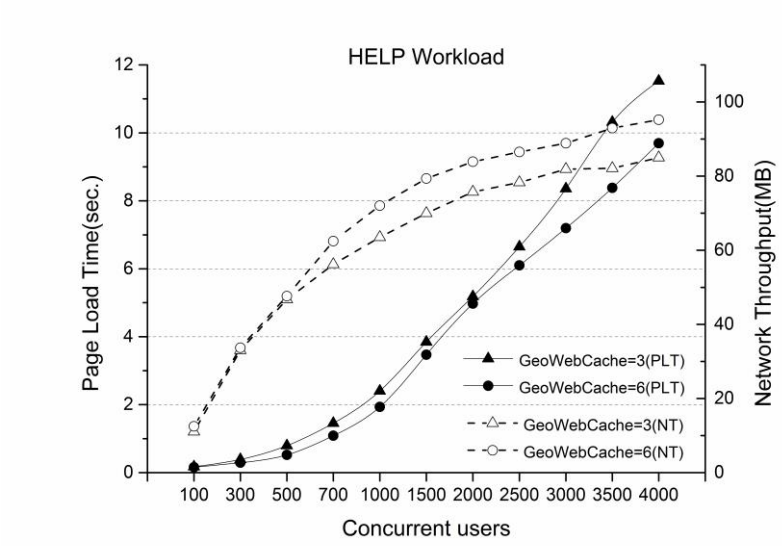

Figure 10. Performance as Represented by the HELP Workload with Different GeoWebCache Nodes

As illustrated in Figure 10, when the number of the service nodes was increased from three to six, there was an almost 15% performance improvement with about a 0.5~2 second reduction in the load time and a 5~10MB increase in network throughput. When the number of the GeoWebCache nodes was increased, the total capability for processing tile requests also increased resulting in a decreased page load time. This experimental result illustrates that our proposed WMTS framework can scale the number of middle-tier service nodes to deal with highly concurrent user requests.

4.2.3. Experiments with more *mongod* nodes: In the storage layer of our implemented WMTS system, the number of *mongod* nodes was increased from six to nine. This experiment was carried out to evaluate whether the performance of the high-concurrency WMTS could be boosted by increasing the database nodes. The results for page load time and network throughput are shown in Figure 11.

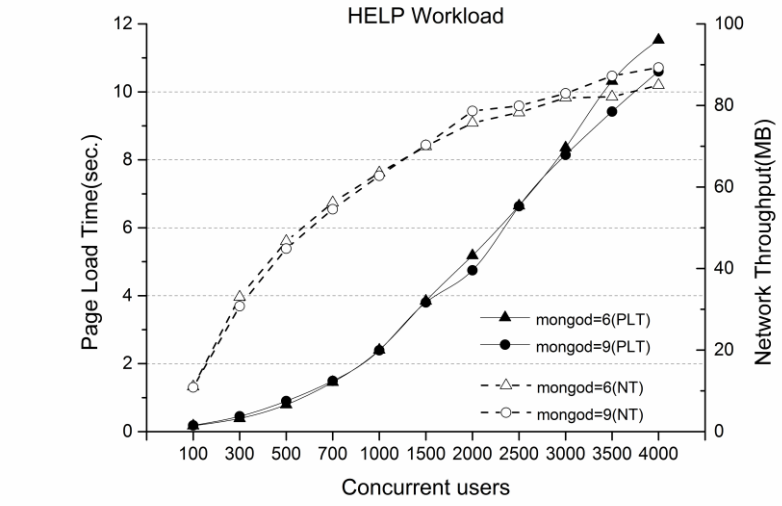

Figure 11. Performance as Represented by the HELP Workload with Different *Mongod* Nodes

As illustrated in Figure 11 when the number of the *mongod* nodes was increased from six to nine, the evaluation metrics of HELP workload exhibited little change. This means an increased number of *mongod* nodes could not deliver a discernable performance improvement. Thus, a bottleneck in the proposed WMTS framework appears in middle service tier, rather than in the bottom storage tier.

5. Conclusions

Web Map Tile Services are emerging as the preferred interface for large-scale geospatial data visualization over the Web. Traditional WMTS services are usually deployed on a single node and thus vulnerable to service oversaturation when faced with thousands of concurrent user requests. In order to tackle this scalability problem found in traditionally deployed WMTS, we designed a scalable cluster-based WMTS framework using several state-of-the-art information technologies, including a load balancer and the NoSQL database. We implemented a prototype system on a high performance cluster to validate our proposed architecture that was built totally on open-source software; Nginx, GeoWebCache, and MongoDB. Experimental results demonstrate that our proposed WMTS framework can effectively improve system throughput and can scale up the number of middle-tier service nodes.

Acknowledgments

This work is supported by the Natural Science Foundation of China (Grant No.: 41301411) and the Natural Science Foundation of Hubei Province (Grant No.: 2015CFB399).

References

[1] J. Maso, K. Pomakis and N. Julia, "OpenGIS Web Map Tile Service Implementation Standard", Open Geospatial Consortium, **(2010)**.
[2] J. de La Beaujardiere, "OpenGIS Web Map Service Implementation Specification", Open Geospatial Consortium, **(2006)**.
[3] P. A. Vretanos, "OpenGIS Web Feature Service 2.0 Interface Standard", Open Geospatial Consortium, **(2010)**.
[4] Whiteside and J. Evans, "OGC Web Coverage Service (WCS) 2.0 Interface Standard", The Open Geospatial Consortium, **(2010)**.
[5] T. N. K. Zatwarnicki, M. Płatek and A. Zatwarnicka, "A Cluster-Based Quality Aware Web System", In Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology–ISAT–Part II, Springer International Publishing, **(2016)**.
[6] E. Choi, "Performance test and analysis for an adaptive load balancing mechanism on distributed server cluster systems", Future Generation Computer Systems, vol. 20, no. 2, **(2004)**, pp. 237-247.
[7] Anitha and R. Balakrishna, "Content-Based Load Balancing Using Web Clusters and Service Rankings", Artificial Intelligence and Evolutionary Computations in Engineering Systems, Springer India, **(2016)**, pp. 1-10.
[8] H. Liu and S. Wee, "Web Server Farm in the Cloud: Performance Evaluation and Dynamic Architecture", Cloud Computing, Springer Berlin Heidelberg, **(2009)**, pp. 369-380.
[9] S. Sharifian, S. A. Motamedi and M. K. Akbari, "A predictive and probabilistic load-balancing algorithm for cluster-based web servers", Applied Soft Computing, vol. 11, no. 1, **(2011)**, pp. 970-981.
[10] F. Monteiro, M. V. Azevedo and A. Sztajnberg, "Virtualized Web server cluster self-configuration to optimize resource and power use", Journal of Systems and Software, vol. 86, no. 11, **(2013)**, pp. 2779-2796.
[11] S. Zhang, W. Wang, H. Wu, A. V. Vasilakos and P. Liu, "Towards transparent and distributed workload management for large scale web servers", Future Generation Computer Systems, vol. 29, no. 4, **(2013)**, pp. 913-925.
[12] V. Cardellini, E. Casalicchio, M. Colajanni and P. S. Yu, "The state of the art in locally distributed Web-server systems", ACM Computing Surveys, vol. 34, no. 2, **(2002)**, pp. 263-311.
[13] T. Schroeder, S. Goddard and B. Ramamurthy, "Scalable Web server clustering technologies", IEEE Network, vol. 14, no. 3, **(2000)**, pp. 38-45.

[14] J. Yang, D. Jin, Y. Li, K. S. Hielscher and R. German, "Modeling and simulation of performance analysis for a cluster-based Web server", Simulation Modelling Practice and Theory, vol. 14, no. 2, **(2006)**, pp. 188-200.
[15] W. Zhang and W. Zhang, "Linux virtual server clusters", Linux Magazine, vol. 5, no. 11, **(2003)**.
[16] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E. Nahum, "Locality-aware request distribution in cluster-based network servers", ACM Sigplan Notices, vol. 33, no. 11, **(1998)**, pp. 205-216.
[17] Cohen, S. Rangarajan and H. Slye, "On the performance of TCP splicing for URL-aware redirection", Proceedings of the 2nd conference on USENIX Symposium on Internet Technologies and Systems, **(1999)**; Boulder, Colorado.
[18] W. Reese, "Nginx: the high-performance web server and reverse proxy", Linux Journal, vol. 2008, no. 173, **(2008)**, p. 2.
[19] V. Kaushal and B. Anju, "Autonomic fault tolerance using HAProxy in cloud environment", International Journal of Advanced Engeneering Sciences and Technologies, vol. 7, no. 2, **(2010)**, pp. 222-227.
[20] B. G. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes", Proceedings of the 10th Roedunet International Conference (RoEduNet), **(2011)**.
[21] B. Veal and A. Foong, "Performance scalability of a multi-core web server", Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems, **(2007)**; Orlando, Florida, USA.
[22] X. Guan, B. Cheng, A. Song and H. Wu, "Modeling Users' Behavior for Testing the Performance of a Web Map Tile Service", Transactions in GIS, vol. 18, no. s1, **(2014)**, pp. 109-125.

Authors

Bo Cheng. He is a PhD student at the State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University. Since 2012, his research interests include distributed spatiotemporal data storage and management, real-time GIS, and task scheduling for massive spatial data processing.

Xuefeng Guan. He received his PhD from Wuhan University in 2011. He is currently working as a lecturer in State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, China. His primary interests include high performance GeoComputation, distributed geographic process simulation, spatial databases and real-time GIS.

