Journal of Engineering and Applied Sciences 12 (5): 1250-1253, 2017
ISSN: 1816-949X

Containerized Microservice Architecture Performance Increasing by Using a Cache Subsystem and Multithreaded Application Server

Vladimir N. Solovyev, Andrey V. Prokofyev and Roman G. Chesov
Flexbby Solution's LLC Dolgoprudny, Moscow Region, Russia

Abstract: This study describes the perfomances of containerized multi-tier client-server architecture. The multi-tier architecture has a multithreaded application server and cache subsystem. Different types of the cache replacment politices were used such as LFU, LRU, PSEUDO LRU Cache, MRU, SLRU, 2-Way Cache.

Key words: SOA, containerization, LXC, application server, cache subsystem

INTRODUCTION

Currently, analytics and enterprise management systems experience a shift from virtualization (Rastogi and Sushil, 2015; Morabito *et al*., 2015) to managed containerized microservice architecture for application hosting (Dua *et al*., 2014). The primary driver for the use of containers was an active development of technologies such as Docker (Anonymous, 2017) and LXC (Linux Containers, 2014) started in 2014. The main advantage of containerization is the promise to reduce applications deployment time by 54%, labor costs by 40% and decrease total cost of ownership by 30% (Rastogi and Sushil, 2015).

However, the containerization is just a way to host and manage application lifecycle. On the other hand, an important parameter of analytics and enterprise management systems is high performance. It means that system response time should not degrade with growing volume of data and increasing number of user requests. It is possible to meet such a requirement of analytics and business applications with the use of multi-tier architecture based on the multithreaded application server (Tang *et al*., 2010; Jinjin and Zhaolin, 2013) and cache subsystem with various cache replacement policies (Chikhale and Shrawankar, 2014; Yin, 2011).

This study studies the performance of containerized multi-tier microservice architecture with the multithreaded application server and cache subsystem.

MATERIALS AND METHODS

Multi-tier client-server architecture with multithreaded application server, ORM and cache subsystem: The studied multi-tier client-server architecture is represented by multi-tier client-server application (Fig. 1). The main components of the architecture are: AJAX WebClient, WebServer, WebAPI, communication pluginslayer, application server, cache subsystem, storage server, database server.

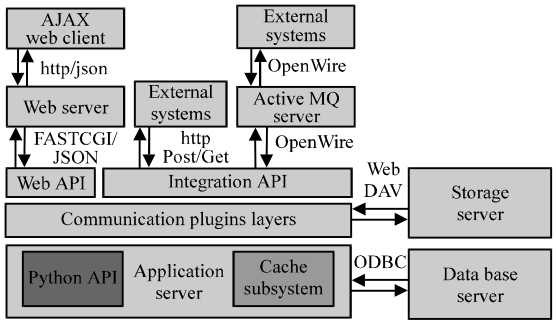

Fig. 1: Developed multi-tier client-server architecture

AJAX WebClient: Is a full-featured Java Script application compatible with all common browsers: Chrome, Firefox, Safari, Internet Explorer including mobile versions. WebClient interacts with WebAPI running on a WebServer sending JSON requests over HTTP/HTTPS emulating two-way connection. It means that both WebClient and WebAPI can initiate requests (integration bus component-communication plugins layers). Emulation is performed by GET long polling request. WebAPI responds either by timeout or when data needs to be transferred to the client appears. Data from WebClient to WebAPI is sent using short POST requests.

WebServer Nginx: Is used as a web server (runs on ports 80 and 443). WebServer works with the WebAPI adapter to transmit user requests to the application server through the communication plugin layer. In addition, application WebServer provides web access for AJAX WebClient, user authentication and dispatches inbound request form AJAX WebClients.

Corresponding Author: Vladimir N. Solovyev, Flexbby Solution's LLC Dolgoprudny, Moscow Region, Russia

Communication plugins layer and WebAPI: Provides the queue of external requests from the AJAX WebClient, external systems through a variety of protocols (HTTP POST/GET, STOMP/OpenWire), dispatching of requests between application servers and system scaling, DTO (Data Transfer Object) transformation of inbound/outbound requests. WebAPI provides interaction between web and application servers and queue management for requests.

Application server: Provides processing of DTO requests according to the business logic, API, supports cache subsystem, serializes the classes of business logic to the structure of database tables, communicates with the database through ODBC driver. It includes API for C++ and Python.

Cache subsystem: Provides data cache in the server's memory and replacement according to the chosen cache replacement policy. The following caching policies were developed for cache subsystem: LFU Cache; LRU Cache; PSEUDO LRU Cache; MRU Cache; SLRU (MRU+LRU); 2 way Cache (associative cache); AIC-intellectual selection of caching algorithm. The set of cache replacement policies may be easily extended with additional policies developed in C++ or Python.

Storage server: It is a server dedicated for processing application server requests related to file access (providing access, downloading, uploading, removing, getting file size). It interacts with the application server via WebDAV protocol.

Database server: PostgreSQL, MSSQL or other relational databases are used as a database. All the communications with the database server are performed only via application server using ODBC interface. The database stores data in tables.

The containerization of multi-tier client-server architecture: The LXC technology has been used as isolation at the operating system kernel level. The isolation layer is a set of libraries running on the top of the Linux kernel, enabling implementation of the logic of the application containers management (separated SOA nodes) and creation a computational cluster. One of the containers includes all the libraries required for the cluster management. Cluster management is performed using the administrator's interface. Hierarchy of isolated container management is shown in Fig. 2. Initsys library was developed to manage isolated containers. It provides

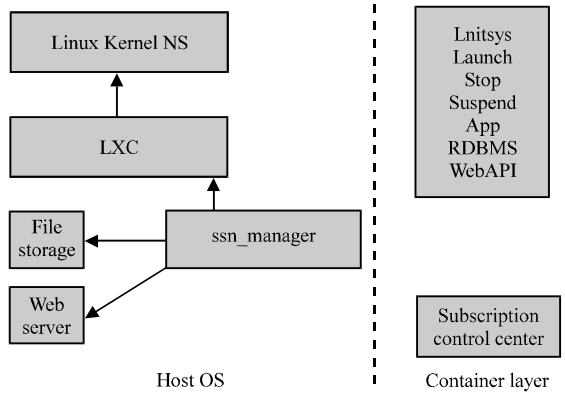

Fig. 2: Hierarchy of isolated container management

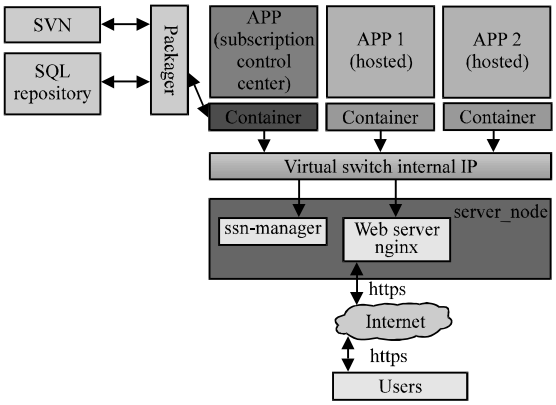

Fig. 3: Network hierarchy of container management

access to containers kernel for the managing application container. The ssn_manager component was developed for the host machine level. Using a high-level API, it provides to the external management server access to the Linux kernel function, the virtualization layer (LXC), the virtual container and its services via the initsys component. In addition, ssn_manager manager has interfaces to file storage servers and web servers for management from the administrator console. All the components of the cloud infrastructure interact via IP protocol. That allows placing the cloud components to physically and geographically distributed infrastructure (Fig. 3).

RESULTS AND DISCUSSION

The performance measurements of containerized architecture: The performance of the containerized architecture and the system response time were tested using the following hardware and software: Server Intel-E5-2650 (2.0 Ghz)×2 (16 Cores), 4×16Gb, HDD×300 Gb.

Software: OS-Ubuntu Server-Ubuntu 14.04.3 LTS. To provide experimental conditions, the module generating user requests (POST) with required frequency was started in one of the containers. The program for measuring the time of response to incoming user requests was installed in the tested application container. To maintain the experimental integrity, one 2.0 GHz processor core and 16 GB RAM were allocated for the tested application container.

Measuring performance of containerized architecture: Measured response time T consisted of two components, the response of the presentation tier (Nginx web server+WebAPI) and application server response time T_2. Application server processed the business logic of user requests:

$$T = T_1 + T_2$$

Presentation tier response time reflects maximal throughput. Presentation tier response time is calculated as time between sending POST request and receiving response that POST request is received and is queued for processing, POST Delay (PD). PD = T_0-T_1, where T_0-the time when the request was sent. The measured layers are shown in Fig. 4d.

The POST Delay parameter was measured as follows: Using request generator, the series of 1000, 2000, 5000 POST requests were sent at the maximum frequency. For each request, the time when it was put in the processing queue of the application server was measured. The experiment proved that the current configuration of the presentation tier provides throughput at least 900 requests per second at single CPU core. The important thing was that the presentation tier provides sufficient performance for testing the application server response time.

The next parameter to be measured was the overall system response time which consisted of presentation tier response time and application server response time. $T = T_2$-T_0 (where T_0-the time of sending the request to the web server, T_2-the time of receiving the response). The measurements of the system response time were conducted in the following settings: 1, 2, 4 threads and 10 threads. The system response time was measured at the frequency varying from 50 requests per second to 400 requests per second. The experimental research of performance of the containerized microservice architecture has shown that the system response time for a single thread reaches up to 160-170 requests per second. For two and more threads the system performance does not degrade with up to 400 requests per seconds.

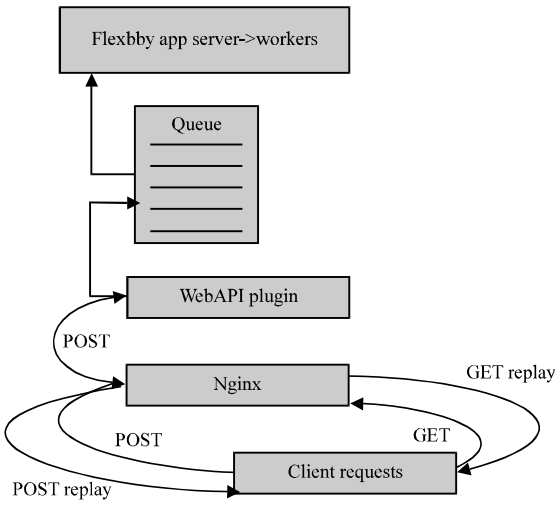

Fig. 4: Scheme of layers and requests

The test of cache subsystem: The caching subsystem is a set of replacement policies for LFU, LRU, PSEUDO LRU Cache, MRU, SLRU, 2-Way Cache and intelligent replacement policy selection algorithm enabling the use of different types of replacement policies depending on the requests. The conditions for measuring the response time of the application container with the caching subsystem were the same as for measuring the performance of a single application container. The main difference was that the application container stored randomly requested data in the database.

To measure system response time using caching algorithms and hit ratio, the following experimental conditions were used:

- Maximum data cache size: 200-5000 business objects
- Number of user sessions: 1000
- Number of unique objects requested: 3000 per single measurement
- Total number of requests: 20000 per single measurement
- Normal distribution of requests, sigma 500 objects
- Studied parameters
- Hit ratio (the probability of locating the requested item in the cache memory)
- System response time
- Total number of displacements of system data objects from the cache

The test results are shown in Table 1. The average system response time for the same dataset and user requests without caching algorithms is 0.45's. The minimal size of the data cache decreases system response time by almost 10 times. With the larger cache sizes, caching

Table 1: System response time for different replacement policies

	Algorithm, response time (sec)						
Cache size	LFU	LRU	MRU	PLRU	SLRU	2-way cache	AIC
200	0.052	0.050	0.052	0.051	0.052	0.055	0.050
400	0.049	0.049	0.050	0.049	0.050	0.050	0.049
600	0.046	0.046	0.048	0.047	0.048	0.048	0.046
1000	0.041	0.041	0.043	0.042	0.043	0.044	0.041
2000	0.029	0.029	0.033	0.032	0.034	0.036	0.029
3000	0.021	0.021	0.024	0.024	0.024	0.031	0.021
4000	0.015	0.015	0.016	0.018	0.016	0.028	0.015
5000	0.014	0.014	0.014	0.014	0.014	0.026	0.014

algorithms reduce system response time by >30 times. As for the size of the data cache which does not contain almost all objects, the algorithms LFU and LRU showed the best result, reducing the system response time by 20-25%. The system response time for the algorithm LFU and LRU is practically indistinguishable.

CONCLUSION

This study studies the performance of containerized multi-tier microservice architecture with the multithreaded application server and cache subsystem. As an isolation layer, LXC technology was used. Application container was tested at a single 2 GHz processor core. The experimental research of performance of the containerized microservice architecture has shown that the system response time for a single thread reaches up to 160-170 requests per second. For two and more threads the system performance does not degrade up to 400 requests per seconds.

The cache subsystem test has shown that cache replacement policies such as LFU, LRU, MRU, PLRU, SLRU reduce the application container response time by >30 times. The most promising algorithms for analytical and business systems are LFU and LRU, the 2-Way Cache algorithm showed the worst result among the studied replacement policies.

ACKNOWLEDGEMENT

This research was performed with the support of the Ministry of Education and Science of the Russian Federation (Reference No. RFMEFI57914X0069).

REFERENCES

Anonymous, 2017. Docker documentation. Docker Inc., San Francisco, USA. https://docs.docker.com/.

Chikhale, K. and U. Shrawankar, 2014. Hybrid multi-level cache management policy. Proceedings of the 2014 Fourth International Conference on Communication Systems and Network Technologies (CSNT), April 7-9, 2014, IEEE, Bhopal, India, ISBN:978-1-4799-3070-8, pp: 1119-1123.

Dua, R., A.R. Raja and D. Kakadia, 2014. Virtualization vs containerization to support paas. Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E), March 11-14, 2014, IEEE, Boston, Massachusetts, USA., ISBN:978-1-4799-3768-4, pp: 610-614.

Jinjin, H. and F. Zhaolin, 2013. The design ERP in the multi-tier architecture. Proceedings of the 2013 Fourth International Conference on Digital Manufacturing and Automation (ICDMA), June 29-30, 2013, IEEE, Qingdao, China, ISBN:978-1-4799-0325-2, pp: 1441-1444.

Linux Containers, 2014. LinuxContainers.org Infrastructure for container projects. Linux Containers Organization, USA. https://linuxcontainers.org/.

Morabito, R., J. Kjallman and M. Komu, 2015. Hypervisors vs. lightweight virtualization: A performance comparison. Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E), March 9-13, 2015, IEEE, Tempe, Arizona, USA., ISBN:978-1-4799-8219-6, pp: 386-393.

Rastogi, G. and R. Sushil, 2015. Cloud computing implementation: Key issues and solutions. Proceeings of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), March 11-13, 2015, IEEE, Dehradun, India, ISBN:978-9-3805-4415-1, pp: 320-324.

Tang, P., W. Fang, H.P. Si and Y.S. Cao, 2010. An enterprise flexible object-relational mapping framework based on metadata and property-seperation storage. Proceedings of the 2010 International Conference on Educational and Network Technology (ICENT), June 25-27, 2010, IEEE, Qinhuangdao, China, ISBN:978-1-4244-7660-2, pp: 263-266.

Yin, L., 2011. Design and implementation of multilayer cache strategy of web system based on seam. Proceedings of the 4th International Conference on Machine Vision (ICMV 11), December 9, 2011, International Society for Optics and Photonics, Singapore, pp: 83500L-83500L.