UDC 004.75

A.I. Shvayka, post-graduate
Pukhov Institute for Problems of Modeling in Energy Engineering
(15, General Naumov St., Kiev, 03164 Ukraine,
(044) 4249165, e-mail: andrew.shvayka@gmail.com)

Load Balancing in IoT Applications Using Consistent Hashing

The paper gives a systematic analysis of the load balancing methods within the scope of the most common cases of the Internet of Things use and network protocols. It also presents the architecture of a load balancing system that supports HTTP, MQTT and CoAP protocols and utilizes a consistent hashing algorithm.

Представлен анализ методов балансировки нагрузки в наиболее популярных сценариях использования и протоколах интернета вещей. Представлен дизайн системы балансировки нагрузки, основанной на использовании консистентного хеширования, которая поддерживает протоколы HTTP, MQTT и CoAP.

K e y w o r d s: load balancing, consistent hashing, IoT, telemetry, HTTP, CoAP, MQTT.

Introduction. The relevance of the load balancing problem in distributed systems is proportional to the amount of physical objects that use these systems to exchange data. The most common objects of the Internet of Things (IoT) are devices, vehicles, and buildings with various sensors embedded into them. The amount of connected devices has grown by 30 % (up to 4 billion, which is the lowest estimate) in 2015, and it is planned to reach 20-40 billion in 2020 according to various forecasts [1].

The main goals of the load balancing process are as follows [2]: to optimize consumption of the server-side resources; to minimize the request processing time; to enable horizontal scalability of the distributed system; to enable resilience of the distributed system.

The capability to load-balance requests between multiple components of a distributed system significantly improves the system's reliability and availability compared to when a single server side component is used. If one or more components of a distributed system break down, the load is distributed between the remaining healthy components.

The load balancing can be performed on various layers of the model of Open Systems Interconnection (OSI). The most popular load balancing solutions operate at the transport and application layers. At the transport layer, the load balancing algorithms can make limited routing decisions based on the information in protocol headers without regard to the actual content of the messages. At the application layer, the load balancing algorithms operate with the actual content of each message [3]. Content-specific load balancing methods are most promising due to their capability to inspect information exchanged in the network and thus make highly intelligent load balancing decisions.

The most common load balancing methods that work well for web servers may also produce good results for IoT applications. However, much better results can be achieved for the IoT applications load balancing, if typical IoT use cases as well as specific network protocols used for the IoT are taken into account when selecting or developing the load balancing algorithms.

IoT use cases. Various use cases of IoT applications represent different load scenarios and have different footprint on server-side resources: Central Processing Unit (CPU), Random Accessory Memory (RAM), disk and network. Understanding the load profile of a particular use case may provide hints and impact applicability of certain load balancing strategies.

Telemetry. Connected devices with built-in temperature, gas and other energy-consumption meters, enable huge amount of use cases in home or enterprise monitoring and control. Main scenario of this use case is to deliver telemetry data to the cloud for further processing and analytics. Delivery and storage of telemetry data mostly impact network and disk. In case of secure communication channel, this operation also affects CPU and RAM (cache to lookup security credentials for a particular device).

Messaging. Connected devices may exchange messages between each other. Messaging capabilities of IoT applications can be implemented using peer-to-peer communication or through the cloud. Peer-to-peer communication usually has no particular impact on the server side of distributed system. On the other hand, communication through the cloud usually impact CPU and RAM. CPU resources are usually spent for encryption and decryption of the messages due to common requirement of secure communication channel. RAM resources are spent to cache security credentials, undelivered messages, session state, etc.

Push notifications. Connected devices may receive notifications that are pushed from the cloud with configuration, alert and other information. In order to deliver notification to device, distributed system should either lookup particular node that manages communication session with device or broadcast of this message to all nodes. Message broadcast may introduce performance bottleneck in case of huge message load. Thus, the ability to localize processing of particular device requests on a particular node of distributed system may be a key requirement for load balancing system. In case of successful device localization, RAM resources are spent to cache undelivered messages and session state.

IoT protocols. Despite the overwhelming variety of powerful Internet-connected devices, some of the devices are very constrained in terms of energy consumption, RAM and flash storage size. This constraints the impact possible choice of network protocols that are used to communicate with application servers [4]. This paper also covers a review of such protocols [5] and their possible impact on load balancing strategies.

Con*strained Application Protocol* (CoAP) works over User Datagram Protocol (UDP) and is used for resource constrained, low-power sensors and devices connected via Lossy networks. This low-power devices often have 8-bit microcontrollers and are related to Class 1 (C1) of constrained devices [4]. CoAP with Observe extension enables publish/subscribe functionality that allows supporting messaging and notifications use cases [6]. Since the protocol is UDP based, message delivery confirmation is optional and is controlled on the application layer. CoAP supports secure extension based on Datagram Transport Layer Security (DTLS) [7]. In the scope of load balancing, CoAP is similar to Hypertext Transfer Protocol (HTTP), however, each request is identified using UDP source and destination host/port pair and protocol specific request token. It is quite effective to embed and parse device identifier from Uniform Resource Request (URI) request.

Message Queuing Telemetry Transport (MQTT) is a client-server publish/subscribe messaging protocol. MQTT client is able to subscribe, publish and un-subscribe from multiple topics. MQTT is lightweight and simple which make it useful in many situations, including constrained devices where a small code footprint is required and/or network bandwidth is at a premium. The protocol runs over Transmission Control Protocol/Internet Protocol (TCP/IP), or over other network protocols (e.g. WebSockets) that provide ordered, lossless, bi-directional connections. Since the protocol is based on TCP, it is not supported by majority of C1 constrained devices. MQTT supports secure communication based on Transport Layer Security (TLS). In the scope of load balancing, MQTT session is identified using TCP source and destination host/port pair. It is quite effective to embed and parse device identifier from MQTT topic or identify device based on corresponding client security certificate.

Hypertext Transfer Protocol. Restful HTTP is a client-server request-response protocol that received wide adoption in IoT applications that are hosted on quite powerful devices. HTTP security protocol (HTTPS) supports secure communication based on HTTP and TLS. The majority of load-balancers and cloud providers support HTTP protocol load balancing with various options

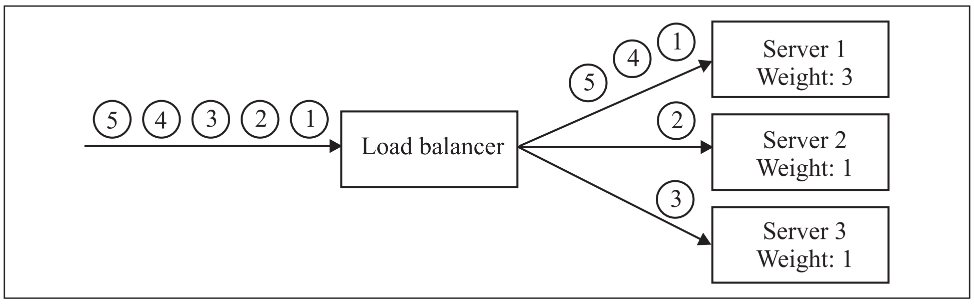

Fig. 1. Load balancing using Weighted Round Robin method

(e.g. sticky sessions) [8]. Despite all advantages, the request-response model makes this protocol unsuitable for real-time messaging and frequent push notifications.

WebSockets. WebSockets protocol allows raising bi-directional communication over HTTP connection. This protocol has also received wide adoption in Web applications due to low latency bi-directional messaging capabilities. However, it requires more resources than MQTT and thus much less popular on constrained devices. Although the WebSockets protocol is not actively used for device-server communication, it is actively used to deliver information from the server to Web and mobile client applications. WebSockets protocol supports security based on TLS.

Load balancing methods are available as the separate specialized software or directly integrated into distributed systems. The existing methods can be divided into static, semi-dynamic and dynamic categories based on the principle of taking into account the dynamics of load profile [9, 10]. When using static methods of load balancing plan is fixed and known before the system startup. Semi-dynamic methods define load balancing plan during system initialization. Dynamic methods adapt load distribution plan, based on specific events, periodically or on a specific schedule. The dynamic methods are the most complex ones in terms of implementation and consume more computing resources. In most cases, their use is justified and can lead to a significant increase of overall system performance. However, in the case of distributed systems with static load profile the use of these methods can cause system performance downgrade.

Round Robin (RR) is one of the simplest methods for distributing client requests across a group of servers. This method forwards a client request to each server in turn. There are several modifications of this method. In case of weighted RR, a weight is assigned to each server based on server hardware configuration (Fig. 1). In case of dynamic RR, the weight is assigned to each server dynamically on the basis of real-time data about the server's current load and

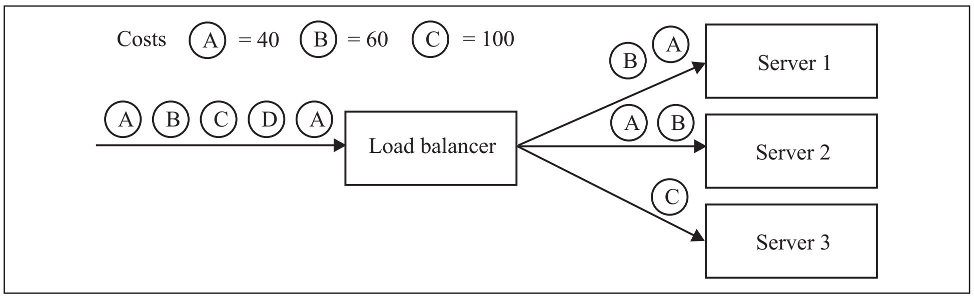

Fig. 2. Load balancing using CAP method

idle capacity. The higher the weight, the larger the proportion of client requests received by the server [11].

In the scope of IoT applications, RR method has a significant disadvantage. Multiple requests from the same device, especially devices that use the request-response network protocols (CoAP, HTTP), will most likely target different servers. This will cause the increase of the cache size that holds device metadata (security credentials, state info, etc.) and eventually will cause low cache hit rate.

Client aware policy (CAP) is a static method that operates on application layer of OSI model and makes routing decision based on the content of request. This method classifies a request based on its impact on server resources: CPU, memory, disk and network. This method uses only the request information and does not attempt to predict time spent to process each specific request (Fig. 2). The method is the most productive in the case of servers with heterogeneous and unpredictable load [12].

In the scope of IoT applications, CAP method requires fine tuning and extra knowledge about behaviour of particular application. This method also does not guarantee locality of requests from a particular device. Thus it may cause similar performance issues to the ones described earlier in RR algorithm description.

Locality aware request distribution (LARD) is a dynamic method that operates on application layer of OSI model and makes routing decision based on the content of request and corresponding server states. The LARD strategy yields scalable performance by achieving both load balancing and cache locality at the back-end servers. This method uses several configuration parameters: Tlow (defines a value below which a back-end node is potentially underutilized) and Thigh (defines a value above which a back-end node is potentially overloaded) (Fig. 3). If one of the servers, which is responsible for handling requests of a specific type is overloaded, load balancer selects one of the underutilized servers or the least downloaded one [13].

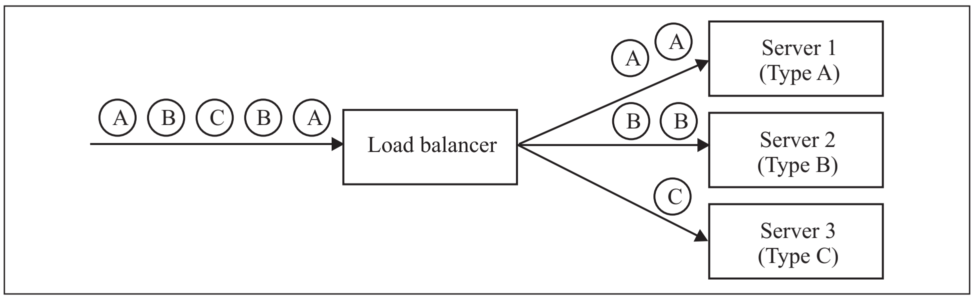

Fig. 3. Load balancing using LARD method

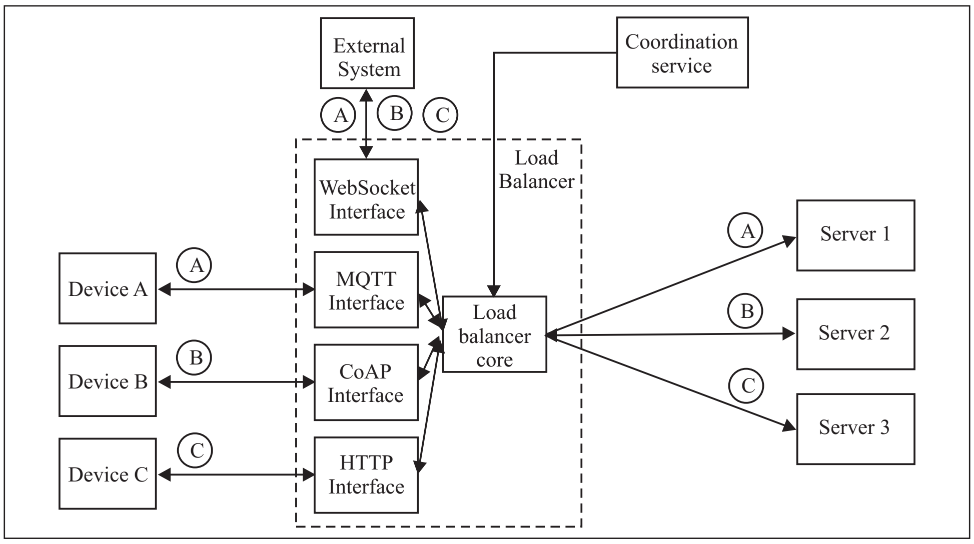

Fig. 4. Proposed system architecture

In the scope of IoT applications, LARD method partially solves a problem with localization of requests based on some request identifier. Assuming that the request type will be determined as a hash function of an application or device identifier, this load balancing strategy can provide good cache hit rate and overall performance. However, requests for a particular device can still hit different servers within one group.

Consistent Hashing. In traditional hash tables, a change in the number of array slots causes nearly all keys to be remapped. Thus, if existing server fails or new server is added, new device sessions most likely will hit different servers. Consistent hashing is a special kind of hashing such that when a hash table is resized, only K/n keys need to be remapped on the average, where K is the number of keys, and n is the number of slots [14].

Consistent hashing technique is actively used in industry standard NoSQL databases and is one of the key strategies to partition load. The concept of virtual nodes allows us to evenly disparse load in case of particular node failure and achieve incremental scalability by adding new nodes [15].

Load balancing system architecture. Consistent hashing algorithm provides ability to address all requests from particular device to specific server based on well-defined device identifier and up-to-date information about list of servers in the cluster. This allows achieving the best possible cache hit rate and localize other memory related resources. The ability to address particular entity and related session information also reduces delays in notification or other message delivery. The proposed architecture assumes that all device requests and related application programming interfaces calls will contain information about device identifier. For example, the device identifier may be a device token or de-

Features	Load balancing methods			
	Weighted RR	CAP	LARD	Consistent Hashing
General Features				
Cache Hit Rate	Low (requests from one device may hit multiple servers)	Low (requests from one device may hit multiple servers)	Medium (requests from one device hit certain group of servers)	High (requests from one device hit the same server)
Sticky Sessions	No	No	No	Yes
Cluster Resizing	Semi-automatic (requires manual configuration)	Semi-automatic (requires manual configuration)	Semi-automatic (requires manual configuration)	Automatic (automatic out-of-the-box)
Inhomogeneity of Servers Inside Cluster (hardware configuration)	High (based on weight parameter)	Medium (easy to improve)	High (based on server type parameter)	Medium (easy to improve)
IoT Use Cases Support				
Telemetry (no security)	High	High	High	High
Telemetry (security enabled)	Low (low cache hit rate on security credentails and session information lookup)	Low (low cache hit rate on security credentails and session information lookup)	Medium (medium cache hit rate on security credentails and session information lookup)	High (best cache hit rate on security credentails and session information lookup)
Messaging	Low (addressing particular device session requires continious persistance of session information)	Low (addressing particular device session requires continious persistance of session information)	Medium (addressing particular device session requires broadcast to certain group of servers)	High (addressing particular device session is based on hash of device id)
Push Notifications	Low (the same as for messaging)	Low (the same as for messaging)	Medium (the same as for messaging)	High (the same as for messaging)

vice security credentials. Analysis of various existing IoT platforms allows assuming that this is a common practice [16, 17] (Fig. 4).

System components. Load balancing system consist of several interface components that enable the support of particular network protocols and core component that should be either aware of coordination service or implement capabilities of this service. Coordination service should provide information about "worker" servers and ability to subscribe for updates about the state of these servers. The preferred implementation of coordination service should be a dedicated distributed system to provide failover and high availability support [18].

Supported protocols. Load balancing system may support pluggable implementations of network protocols, however in this article we have reviewed MQTT, CoAP, HTTP and WebSocket implementations. The basic implementation can parse the device identifier from the URI in case of CoAP, HTTP and WebSocket implementations. The topic name can be used to extract the device identifier in MQTT protocol. The other option is to use credentials based on one of the authentication methods: access tokens, hash of the client certificate, etc.

In order to compare load balancing methods listed above we will use the comparison table. The comparison will be based on general load balancing features and level of IoT use cases support.

Conclusion. The proposed architecture assumes that all worker servers have identical or very similar hardware configuration. However, the system can be improved to support various hardware configurations by introducing the weight for a particular server node. The capability to coordinate several load balancer instances and support balancing between servers with different hardware configurations is the field of the future research.

Наведено аналіз методів балансування навантаження в найбільш популярних сценаріях використання і протоколах інтернету речей. Надано дизайн системи балансування навантаження, базованої на використанні консистентного хеширувания, яка підтримує протоколи HTTP, MQTT і CoAP.

REFERENCES

1. Lueth, K.L. (2014), IoT market – forecasts at a glance, *IoT Analytics*, available at: https://iot-analytics.com/iot-market-forecasts-overview/ (accessed March, 2016).
2. Zhang Lin, Li Xiao-ping and Su Yuan (2010), A content-based dynamic load-balancing algorithm for heterogeneous Web server cluster, *ComSIS*, Vol. 7, no. 1, Special issue, pp. 153-162, available at:http://www.doiserbia.nbrs/img/doi/1820-0214/2010/1820-02141001153Z.pdf. (accessed March, 2016).
3. What is layer 7 load balancing?, *Nginx Documentation*, available at: https://www.nginx.com/ resources/glossary/layer-7-load-balancing/ (accessed: March, 2016).
4. Bormann, C., Ersue, M. and Keranen, A. (2014), Terminology for constrained-node networks, *Internet Engineering Task Force (IETF) Documents*, available at: https://tools.ietf.org/html/rfc7228#section-3 (accessed March, 2016).

5. Duffy, P. (2013), Beyond MQTT: A Cisco view on IoT protocols, *Cisco Blogs, Digital Transformation*, available at: http://blogs.cisco.com/digital/beyond-mqtt-a-cisco-view-on-iot-protocols (accessed March, 2016).
6. Hartke, K. (2015), Observing resources in the Constrained Application Protocol (CoAP), *Internet Engineering Task Force (IETF) Documents,* available at: https://tools.ietf.org/html/rfc7641 (accessed March, 2016).
7. Shelby, Z., Hartke, K. and Bormann, C. (2014), The Constrained Application Protocol (CoAP), *Internet Engineering Task Force (IETF) Documents*, available at: https://tools.ietf.org/html/rfc7252 (accessed March, 2016).
8. Elastic load balancing product details, *Amazon Web Services Documentation*, available at: https://aws.amazon.com/ru/elasticloadbalancing/details (accessed March, 2016).
9. Anicas, M. (2014), An Introduction to HAProxy and load balancing concepts, *DigitalOcean, HAProxy*, available at: https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts (accessed March, 2016).
10. Best practices in evaluating elastic load balancing. (2014), *Amazon Web Services Documentation*, available at: https://aws.amazon.com/articles/1636185810492479 (accessed March, 2016).
11. What is Round-Robin load balancing?, *Nginx Documentation*, available at: https://www.nginx.com/resources/glossary/round-robin-load-balancing/ (accessed March, 2016).
12. Casalicchio, E. and Colajanni, M. (2001), A client-aware dispatching algorithm for Web clusters providing multiple services, *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong, May 1-5, 2001, pp. 535-544, available at: http://www.survey.ethz.ch/CDstore/www10/papers/pdf/p434.pdf (accessed March, 2016).
13. Druschel, P. (1999), The LARD strategy, *USENIX Board of Directors Election Results. – April*, available at: https://www.usenix.org/legacy/publications/library/proceedings/usenix99/full_papers/aron/aron_html/node13.html (accessed March, 2016).
14. Karger, D. et al. (1997), Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, *STOC '97 Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 654-663, available at: https://www.usenix.org/legacy/publications/library/proceedings/usenix99/full_papers/aron/aron_html/node13.html. TODO: NEW LINK (accessed August, 2016).
15. DeCandia, G. et al. (2007), Dynamo: Amazon's highly available key-value store, *SOSP'07 Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, Vol. 41, Iss. 6, pp. 205-220, available at: ://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf (accessed March, 2016).
16. Basics of MQTT, *Thethings.iO Internet of Things Platform*, available at: https://developers.thethings.io/makers-mqtt.html (accessed March, 2016).
17. Security and identity for AWS IoT, *Amazon Web Services Documentation*, available at: http://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html (accessed March, 2016).
18. Hunt, P., Konar, M., Paiva, F., Junqueira, P.F. and Reed, B.C. (2010), "ZooKeeper: Wait-free coordination for Internet-scale systems", *USENIXATC'10 Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, pp. 1-14, available at: https://www.usenix.org/legacy/event/usenix10/tech/full_papers/Hunt.pdf (accessed March, 2016).

Received 06.06.16;
after revision 18.08.16

SHVAYKA Andrei Igorevich, post-graduate of the Pukhov Institute for Problems of Modeling in Energy Engineering. In 2001 A. Shvaika graduated from the National Technical University of Ukraine Kyiv Polytechnic Institute. A field of research: load balancing, distributed systems, high-load systems.