Table of Contents

1 Introduction

1.1 Concept

Although there is no absolute definition of Edge computing, a consensus is that edge computing is a distributed computing framework that directs computational data, applications, and services away from cloud servers to the edge (e.g., IoT devices or local edge servers [1]) of a network [2]. According to Karim Arabi [3], edge computing can be broadly defined as all computing outside the cloud happening at the edge of the network and, more specifically, in applications where real-time data processing is required.

As a concept often compared with edge computing, cloud computing is the delivery of computing services over the Internet [4]. Similarly, cloud and edge service providers furnish end users with an application, data computation, and storage services. However, the main difference between edge computing and cloud computing lies in the location of the servers. Services in edge computing are located in the edge network whereas the services in the cloud are located within the Internet. In Karim Arabi's definition [5], cloud computing operates on big data while edge computing operates on "instant data," that is, real-time data generated by sensors or users, which is attributed to their difference in distance from end users.

It is also worth mentioning that fog computing is a synonym of edge computing [6][7] in this report, although it is regarded as a branch of edge computing in some materials [8].

1.2 Classification

Mobile edge computing (MEC) [9] and cloudlets [10] are two major categories of edge computing discussed in this report, apart from which Intelligent Transport Systems Clouds (ITS-Clouds) [11] and VANET (Vehicular Ad Hoc NETworks)-Clouds [12] are also noticeable aspects of edge computing.

MEC is an open cloud platform that uses some end-user clients and is located at the mobile edge to carry out a substantial amount of storage (rather than stored primarily in cloud data centers) and computation (including edge analytics, rather than relying on cloud data centers) in real time, communication (rather than routed over backbone networks), and control, policy and management (rather than controlled primarily by network gateways such as those in the LTE core) [13].

Cloudlet is a proximate fixed cloud that consists of one or several resource-rich, multi-core, Gigabit Ethernet-connected computer aiming to augment neighboring mobile devices while minimizing security risks, offloading distance (one-hop migration from mobile to cloudlet), and communication latency. The mobile device plays the role of a thin client while the intensive computation is entirely migrated via Wi-Fi to the nearby cloudlet. Although cloudlet utilizes proximate resources, the distant fixed cloud infrastructures are also accessible as a backup in case of cloudlet scarcity. The authors employ a decentralized, self-managed, widely spread infrastructure built on hardware VM technology. Cloudlet is a VM-based offloading system that can significantly shrink the impact of hardware and OS heterogeneity between mobile and cloudlet infrastructures.

1.3 Objectives

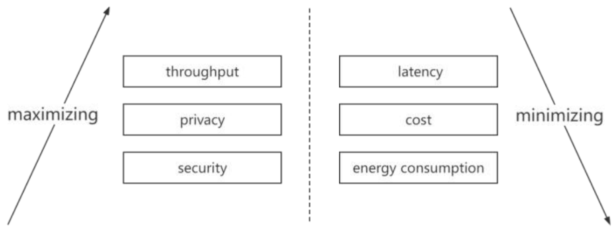

Fig. 1. Objectives of edge computing

Fig. 1 illustrates several prime objectives shared by edge computing of all kinds, yet extra factors would be taken into consideration in specific edge computing cases, e.g., increasing useful computations, optimizing placement when dealing with cloudlets.

2 Background

2.1 Existing Works

This section outlines a few solutions based on main objectives of edge computing.

Obtaining real-time data insights. J. He et al. [14] proposed a hierarchical fog computing model enabling large-scale data analytics for smart city applications. The model is comprised of ad hoc and dedicated fogs that mitigate the indispensable problems associated with dedicated computing infrastructure and cloud computing in terms of latency. The experiments were performed on Raspberry Pi computers based on a distributed computing engine, which helps measure the performance of the analytics tasks and creates easy-to-use workload models. In order to maximize the analytics service utilities, quality-ware admission control, offloading, and resource allocation approaches were recommended. The experimental results revealed that the proposed multitier Fog-based model significantly improves analytics services for smart city applications. However, it lacked a rigorous evaluation and validation of the model.

Minimizing cost. F. Hao et al. [15] proposed a two-layer multi-community framework (2L-MC3) for MEC that allocates tasks in community clouds or cloudlet-based environment by incorporating costs (e.g., monetary, access), energy consumption, security level and average trust level. The problem is optimized in two ways: horizontal optimization and vertical optimization. The horizontal optimization maximizes the Green Security-based Trust enabled Performance Price ratio whereas vertical optimization minimizes the communication cost between selected community clouds and cloudlets. The proposed framework is made complex due to the use of bilevel programming [16], yet it minimizes the accessing cost, optimizes the degree of trust and saves the energy.

Optimizing Cloudlet placement. A solution to address the problem of cloudlet placement was proposed by M. Jia et al. [17] in the form of two heuristic algorithms for optimal cloudlet placement in a user-dense environment of wireless metropolitan area network (WMAN). A simple heaviest-ap first (HAF) algorithm places a cloudlet at those APs with the heaviest workloads. However, the drawback of the HAF approach is that the APs with highest workload are not constantly nearest to their users. A density-based clustering (DBC) algorithm is proposed to overcome the shortcoming of the HAF algorithm by deploying cloudlets in dense users' places. The algorithm also assigns mobile users to the deployed cloudlets considering their workload. Despite the optimal cloudlet placement, the proposed solutions do not provide support for user mobility considered as a promising feature of mobile networks.

Reducing network latency. Q. Fan et al. [18] minimized the total response time of user equipment (UE) request within the network by proposing a scheme known as Application awaRE workload Allocation (AREA). The proposed approach works by allocating different user equipment (IoT based) requests among the least work loaded cloudlets with minimum network delay between the UE and the cloudlet. In order to minimize the delay, the UE requests are assigned to the nearest cloudlets. The proposed approach was simulated on a medium size cluster using the evaluation performance metrics of response time, computing delay, and network layer with significant improvement over existing approaches. However, in real scenario, the simulation set-up seems infeasible as within an area of 25 km, setting up 25 base stations (BS) is impractical. There is high probability of BS overshooting resulting in interference within neighboring BS, which would cause network delay.

Maximizing useful computations. K. Habak et al. [19] presented a dynamically self-configuring cloudlet system called FemtoClouds which coordinates multiple mobile devices to provide computational offloading services. Contrary to traditional cloudlet phenomena, FemtoClouds utilizes the surrounding mobile devices to perform computational offloading tasks, thereby reducing the network latency. In the proposed system, a device (e.g., laptop) acting as a cloudlet creates a Wi-Fi access point and controls the mobile devices (compute cluster) willing to share computation. Mobile devices can offload a task by giving proper information about the task (such as input and output data size and computational code) to the cloudlet. cloudlet first inspects the available computation resources in the compute cluster and the time required to execute the task. Then, the task is scheduled to the available devices using the greedy heuristic approached optimization model. The presented system creates a community cloud with minimal dependent on corporate equipment. The performance of the presented system depends on the number of mobile device participation in the compute cluster.

Strengthening security. S.J. Stolfo [20] proposed an approach to secure the cloud by utilizing decoy information technology. The authors introduced disinformation attacks against malicious insiders using the decoy information technology. Such attacks prevent the malicious insiders from differentiating the real sensitive data from customer fake data. The security of the cloud services is enhanced by implementing the two security features of user behavior profiling and decoys. User profiling constitutes a common technique used to detect when, how, and how much a user accesses the data in the cloud. The normal behavior of a user can be regularly monitored to identify abnormal access to a user ' s data. The decoy consists of worthless data generated to mislead and confuse an intruder into believing that they have accessed the valuable data. The decoy information can be generated in several forms, such as decoy documents, honeypots, and honeyfiles. The decoy approach may be used combined with user behavior profiling technology for ensuring the information security in the cloud. When abnormal access to the service is observed, fake information in the form of decoy is delivered to the illegitimate user in such a way that it seems completely normal and legitimate. The legitimate user immediately recognizes when decoy data is returned by the cloud. The inaccurate detection of unauthorized access is informed by varying the cloud responses using different means such as challenge response to the cloud security system. The proposed security mechanism for Fog computing still requires further study to analyze false alarms generated by the system and miss detections.

2.2 Related Software Tools

Docker. Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc. Docker can package an application and its dependencies in a virtual container that can run on any Linux, Windows, or macOS computer. This enables the application to run in a variety of locations, such as on-premises, in public (see decentralized computing, distributed computing, and cloud computing) or private cloud. When running on Linux, Docker uses the resource isolation features of the Linux kernel (such as cgroups and kernel namespaces) and a union-capable file system (such as OverlayFS) to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Docker on macOS uses a Linux virtual machine to run the containers. [21]

Kubernetes. Kubernetes is an open source container management tool. It is a Go-Lang based (https://golang.org), lightweight, and portable application. You can set up a Kubernetes cluster on a Linux-based OS to deploy, manage, and scale the Docker container applications on multiple hosts. [22]

3 The Project

3.1 Objective

This project aims to develop a system running on edge that converts a large amount of PDF files to other formats with high throughput.

3.2 System Design: Standalone Mode

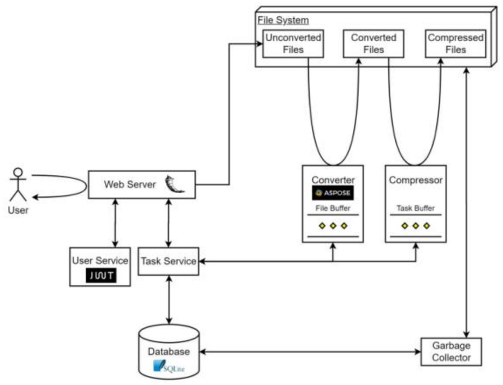

Fig. 2. Standalone mode architecture.

Fig. 2 shows the architecture of standalone mode. When user uploads a file, the **web server** (i.e. Flask [23]) creates a task record and store it in the **database** (i.e., SQLite [24]) through **task service** and save files onto the **file system**. The **converter** keeps polling the database and loading unconverted file information into its buffer. When a record is consumed from the buffer, the converter does file format conversion based on Aspose Words [25] and updates file status via task service. The **compressor** polls and compresses converted files and change their corresponded task status to finished before a user could download his files. Meanwhile, the **user service** is responsible for user identification and authorization based on JWT [26]. This token based approach provides both security and less workloads on edge devices. The **garbage collector** runs continuously to remove out-dated records and files.

3.3 System Design: Cluster Mode

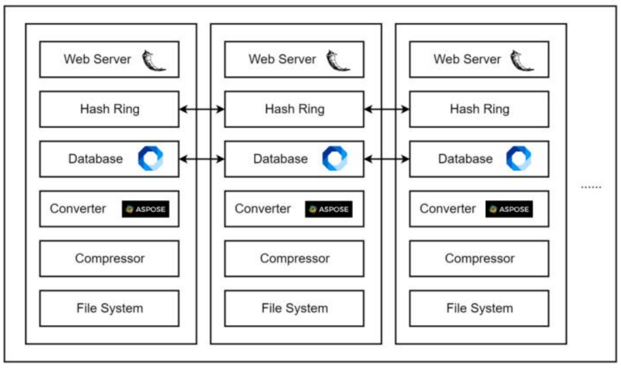

Fig. 3. Cluster mode architecture.

The working procedure inside one node in cluster mode is generally the same as in standalone mode, while the whole system introduces quantities of extra algorithms and techniques on account of the distributed design. Under this distributed context, some additional problems should be taken into consideration.

Database. Although SQLite in standalone mode is one of the most well-known lightweight databases, it is not capable of distributing itself, in which case a new alternative should be used. However, the replacement should not be heavy to avoid adding much workload of edge devices and support transactions to guarantee data consistency (as the database in this system would not be a bottleneck of performance but the status of files and tasks may update concurrently). In cluster mode, RQLite [27], a distributed implementation based on SQLite, is being used. Utilizing RQLite only increases a little workload but provides high availability and relational data modeling.

Load balancing. A major advantage of distributing is to improve the throughput, yet a bad load balancing strategy would eliminate the benefits to a large extent. In search of an optimal load balancing strategy, one thing we have to figure out in advance is the minimum unit for balancing, namely per request, per task, or per user.

Load balancing for every web request should be the most straightforward strategy and could be achieved by merely setting a load balance argument of Kubernetes (i.e., Round Robin). However, most requests in this system are not idempotent (i.e., uploads). Users cannot acquire files they uploaded if they are directed to another node unless we introduce file replications. Notice that we are not able to redirect users' download requests to the node where their converted files are stored either, because load balancing here is achieved by Kubernetes, which is ignorant about which nodes a file is on.

Load balancing for every user is also an option that can be achieved directly by features given by Kubernetes. With the system in standalone mode packaged in a docker image, we can make it distributed simply by orchestrating copies of it and setting the Kubernetes load balance argument to Ring Hash, an instance of consistent hashing. This allows a user (e.g., identified by IP address) to bind to a corresponding node and will keep visiting that node before changes in node topology affects the hash result so that a user can almost always find his conversion history. Although this approach is simple but practical, it also has obvious drawbacks. First, consider the occasion that some frequent users make up the most workloads, which will gather workloads to the nodes corresponding to those users through this strategy, leading to inefficient load balancing. Moreover, as this approach is merely a collection of single machine instances, it only provides limited high availability (e.g., data in databases may lose without replications if a node fails). Additionally, when a node leaves, files and data on it will be discarded, and users corresponding to that node will not be able to find their converted files and will go to the successor node next time.

Load balancing for every task is the method this system eventually adopts. Every time a user submits a task, the submit request is redirected to a node based on the hash result of a generated task ID by consistent hashing. Database records keep track of where tasks were redirected to so that users can always find their conversion history. This approach avoids the frequent-user problem mentioned above by regarding a task as a balancing unit. Actually, it is already the minimum unit appropriate for load balancing because if we distribute smaller units (e.g., files) to different nodes, the compressor needs to fetch files from other nodes rather than process them locally to compress files in a task into a zip file. Naturally, however, this approach could not be achieved by the mechanisms of Kubernetes. Details of its implementation will be discussed later.

Hash algorithm. Consistent hashing [28] is a special kind of hashing algorithm such that when a hash table is resized, only *n/m* keys need to be remapped on average where *n* is the number of keys and *m* is the number of slots. Canonical consistent hashing is achieved by a hash ring. When a node is added, keys kept by its predecessor will be moved to it, as well as corresponding values to the keys, and the key-value pair will go back to its successor both logically and physically if the node decides to leave. This is reasonable for a KVStore system, as it could also be regarded as a load balancing scenario where workloads are defined as storage (i.e., to store entries evenly among nodes). Back to this system, however, the major workload is not file storage anymore but file conversion tasks; that is, there's no need to transfer files between nodes to make files stored evenly on nodes, but the system has to distribute conversion tasks in a balanced way. Consequently, this system applies consistent hashing algorithm with changes.

Searching. Canonical consistent hashing follows the hash result strictly because entries are transferred between nodes when node topology changes so that one can always know the node where the key itself stores a key, while files will not always be transferred in this system. Fortunately, the node responsible for a task is recorded so that the system is able to find a file by querying the database.

Upon a node being added. When a node is added, the system only needs to tell other nodes information about the new node so that other nodes would add this node to their local hash ring, demonstrating a new node comes. Next time a user uploads files will hash based on the new hash ring.

Upon a node left. This is the only case we need file transfer between nodes to avoid converted files being lost. When dropping a node, the node first copies its files to its successor by FTP [29], then updates the database record about where its files will be stored later on. After finishing these operations, this node can announce its departure to other nodes, and they will update their hash ring. Note that the hash ring in this system is maintained by Raft [30] for consensus. This is mentioned here because consensus for the hash ring is actually not a vital problem in node addition; that is, a task could still find its destination even if a local hash ring does not add a new node in time, although the destination is not where it is supposed to go. However, if the hash ring does not reach consensus in node deletion, it will possibly direct tasks to nonexistent nodes, leading to severe problems.

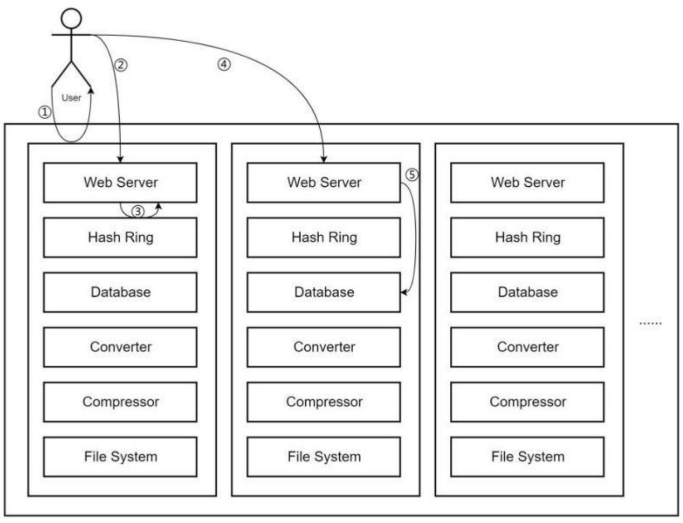

Fig. 4. Upload workflow.

Fig.4 illustrates the workflow of upload in cluster mode. The user first ask the web server for a task ID it used for the task (1). When an upload request comes (2), the web server will check the hash ring for its destination (3) and redirect the request to the right node (4). Then the database records the node responsible for this task (5), and file conversion and compression will be done on the new node.

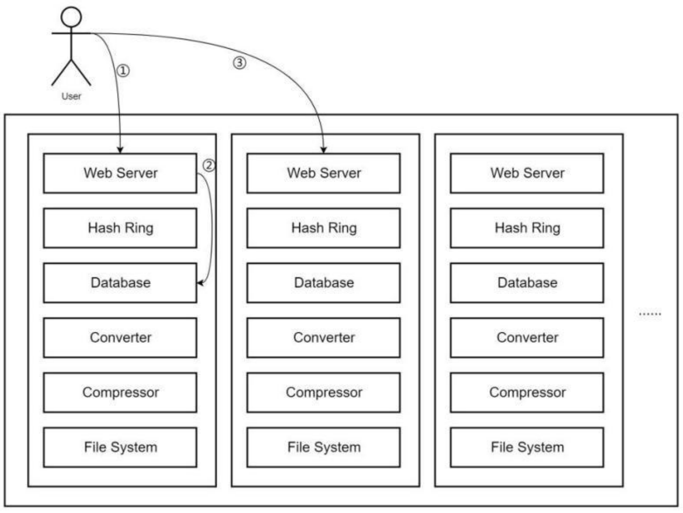

Fig. 5. Download workflow.

Fig. 5 shows the download procedure of the system. When a download request comes, the web server queries the database an redirect the request to its corresponding node.

4 Evaluation

This section compares this project running in cluster mode (pseudo with 5 instances on one machine), standalone mode and does the conversion by Aspose Words directly. Aspose Words all disabled multiprocessing during experiments for fair. The experiments were run on docker having each node with memory argument set to 2G and CPU argument set to 1 [31]. The docker containers were run on a machine equipped with Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz processors and 32GB memory.

4.1 Datasets and Set-up

Table 1. PDF dataset.

Dataset	Decompressed Size	File Amount
1000 .gov PDF	757 MB	1000

The experiments are carried out on 1000 .gov PDF dataset. Table 1 lists the statistic of the dataset.

As mentioned in 3.4 that task is the balancing unit in this system, datasets were divided randomly into tasks. **Task size** is defined as the amount of files contained in a task. In different experiments, task size were chosen to be 1, 10, 100, or a random number between 1 and 100, respectively to simulate different user behavior.

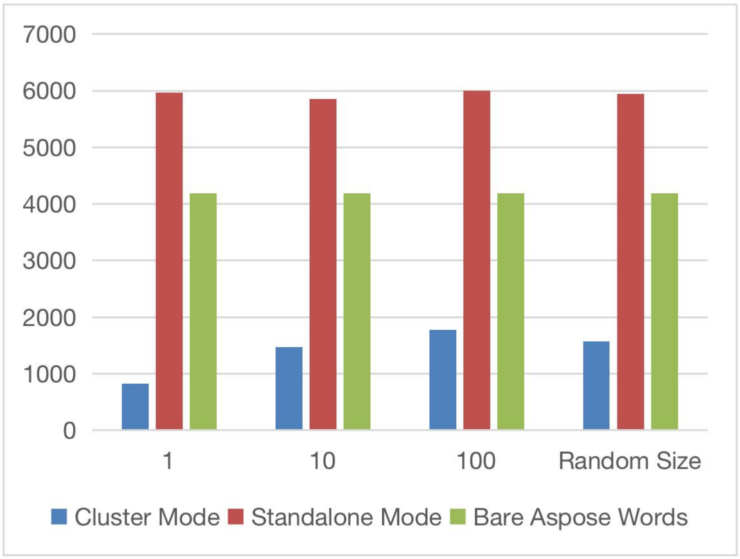

Fig. 6. Total conversion time for 1000 .gov PDF.

References

1. What Is Edge Computing | IBM, https://www.ibm.com/cloud/what-is-edge-computing, last accessed 2022/11/21.
2. Edge computing: A survey, Future Generation Computer Systems, Volume 97, 2019, Pages 219-235, ISSN 0167-739X, https://doi.org/10.1016/j.future.2019.02.050.
3. KEYNOTE: Mobile Computing Opportunities, Challenges and Technology Drivers. https://web.archive.org/web/20200730234708/http://www2.dac.com/events/videoarchive.aspx?confid=170&filter=keynote&id=170-103--0&#video, last accessed 2022/11/21.
4. What Is Cloud Computing? A Beginner ’ s Guide | Microsoft Azure, https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing, last accessed 2022/11/21.
5. Trends, Opportunities and Challenges Driving Architecture and Design of Next Generation Mobile Computing and IoT Devices. https://www.mtl.mit.edu/events-seminars/seminars/trends-opportunities-and-challenges-driving-architecture-and-design-next
6. What is edge computing? (redhat.com), https://www.redhat.com/en/topics/edge-computing/what-is-edge-computing, last accessed 2022/11/21.
7. What is a Fog Node? A Tutorial on Current Concepts towards a Common Definition. https://doi.org/10.48550/arXiv.1611.09193
8. What Is Edge Computing? - Cisco. https://www.cisco.com/c/en/us/solutions/computing/what-is-edge-computing.html, last accessed 2022/11/21.
9. Ejaz Ahmed, Mubashir Husain Rehmani, Mobile Edge Computing: Opportunities, solutions, and challenges, Future Generation Computer Systems, Volume 70, 2017, Pages 59-63, ISSN 0167-739X, https://doi.org/10.1016/j.future.2016.09.015
10. M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," in IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct.-Dec. 2009, doi: 10.1109/MPRV.2009.82.
11. S. Bitam and A. Mellouk, "ITS-cloud: Cloud computing for Intelligent transportation system," 2012 IEEE Global Communications Conference (GLOBECOM), 2012, pp. 2054-2059, doi: 10.1109/GLOCOM.2012.6503418.
12. Bitam S, Mellouk A, Zeadally S. VANET-cloud: a generic cloud computing model for vehicular Ad Hoc networks[J]. IEEE Wireless Communications, 2015, 22(1): 96-102.
13. Edge Definition and How It Fits with 5G Era Networks - IEEE Software Defined Networks. https://sdn.ieee.org/newsletter/march-2016/edge-definition-and-how-it-fits-with-5g-era-networks
14. J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou and Y. Zhang, "Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities," in IEEE Internet of Things Journal, vol. 5, no. 2, pp. 677-686, April 2018, doi: 10.1109/JIOT.2017.2724845.
15. F. Hao, D. -S. Park, J. Kang and G. Min, "2L-MC3: A Two-Layer Multi-Community-Cloud/Cloudlet Social Collaborative Paradigm for Mobile Edge Computing," in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4764-4773, June 2019, doi: 10.1109/JIOT.2018.2867351.
16. Bilevel optimization - Wikipedia. https://en.wikipedia.org/wiki/Bilevel_optimization
17. M. Jia, J. Cao and W. Liang, "Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks," in IEEE Transactions on Cloud Computing, vol. 5, no. 4, pp. 725-737, 1 Oct.-Dec. 2017, doi: 10.1109/TCC.2015.2449834.

18. Q. Fan and N. Ansari, "Workload Allocation in Hierarchical Cloudlet Networks," in IEEE Communications Letters, vol. 22, no. 4, pp. 820-823, April 2018, doi: 10.1109/LCOMM.2018.2801866.
19. K. Habak, M. Ammar, K. A. Harras and E. Zegura, "Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge," 2015 IEEE 8th International Conference on Cloud Computing, 2015, pp. 9-16, doi: 10.1109/CLOUD.2015.12.
20. S. J. Stolfo, M. B. Salem and A. D. Keromytis, "Fog Computing: Mitigating Insider Data Theft Attacks in the Cloud," 2012 IEEE Symposium on Security and Privacy Workshops, 2012, pp. 125-128, doi: 10.1109/SPW.2012.19.
21. Docker (software), https://en.wikipedia.org/wiki/Docker_(software), last accessed 2022/11/21.
22. Thomas Uphill, John Arundel, Neependra Khare, Ke-Jou Carol Hsu: DevOps Puppet, Docker, and Kubernetes: Practical recipes to make the most of DevOps with powerful tools, Packt Publishing (March 31, 2017).
23. Flask, https://flask.palletsprojects.com, last accessed 2022/11/21.
24. SQLite, https://www.sqlite.org, last accessed 2022/11/21.
25. Aspose, https://www.aspose.com/, last accessed 2022/11/21.
26. JWT, https://jwt.io/, last accessed 2022/11/21.
27. RQLite, www.rqlite.io, last accessed 2022/11/21.
28. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D., 1997, May. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (pp. 654-663).
29. FTP, https://en.wikipedia.org/wiki/File_Transfer_Protocol, last accessed 2022/11/21.
30. Ongaro, D. and Ousterhout, J., 2014. In search of an understandable consensus algorithm. In 2014 USENIX Annual Technical Conference (Usenix ATC 14) (pp. 305-319).
31. Runtime options with Memory, CPUs, and GPUs | Docker Documentation, https://docs.docker.com/config/containers/resource_constraints/, last accessed 2022/11/21.
32. Web scraping, https://en.wikipedia.org/wiki/Web_scraping, last accessed 2022/11/21.
33. Proxy server, https://en.wikipedia.org/wiki/Proxy_server, last accessed 2022/11/21.
34. What is a CDN? | How do CDNs work?, https://www.cloudflare.com/learning/cdn/what-is-a-cdn/, last accessed 2022/11/21.