Resource Utilization of Distributed Databases in Edge–Cloud Environment

Yaser Mansouri, Victor Prokhorenko, Faheem Ullah, and Muhammad Ali Babar

Abstract—A benchmark study of modern distributed databases (DDBs) (e.g., Cassandra, MongoDB, Redis, and MySQL) is an important source of information for selecting the right technology for managing data in edge–cloud deployments. While most of the existing studies have investigated the performance and scalability of DDBs in cloud computing, there is a lack of focus on resource utilization (e.g., energy, bandwidth, and storage consumption) of workload offloading for DDBs deployed in edge–cloud environments. For this purpose, we conducted experiments on various physical and virtualized computing nodes, including variously powered servers, Raspberry Pi, and hybrid cloud (OpenStack and Azure). Our extensive experimental results reveal insights into which database under which offloading scenario is more efficient in terms of energy, bandwidth, and storage consumption.

Index Terms—Bandwidth, cloud computing, distributed databases (DDBs), edge computing, energy, storage.

1. INTRODUCTION

Harnessing the power of cloud computing can improve the usage of computing, storage, networking, and multitenant applications and databases over the Internet [1]. The centralization of cloud computing introduces delays for time-critical processing over wide area networks (WANs). This downside has led to the deployment of an edge computing paradigm that enables storing and processing data close to data sources rather than sending data to the cloud for processing. Such an approach aims to improve response times and reduce bandwidth consumption which may be critical for IoT applications. Relying on solely edge computing to deploy data-intensive applications, however, might not be always achievable due to limited resources in terms of computing, networking, data storage, and energy [2]. Therefore, the usage of a combined edge–cloud framework may be a viable solution in certain scenarios.

Running databases on the edge–cloud framework is challenging because it should be highly efficient in both performance and utilization of resource-constrained devices. This rises the main question relating to the applicability boundary of distributed databases (DDBs) deployment on the edge–cloud framework in terms of energy, bandwidth, and storage consumption per operations. We intend to fill this research gap and shed light on the efficiency hierarchy in terms of energy, storage, and bandwidth consumption for Cassandra, MongoDB, Redis, and MySQL. In addition, we share the experience related to the challenges we faced and the lessons we learned throughout resource measurement experiments.

Deployment of DDBs on edge–cloud framework enables task offloading from resource-constrained devices to powerful servers. The task offloading concept involves the questions of when, where, and what tasks should be offloaded (i.e., executed remotely).1

As an answer to where a task should be offloaded, we considered three options: 1) edge device; 2) adjacent server; and 3) remote server. In our work, a laptop and a cluster of Raspberry Pis (RPi) are considered as edge devices. A high-performance server with a distance of several meters from the edge devices is an adjacent server. VMs in the hybrid cloud are considered as remote servers. All edge devices, adjacent server, and remote servers have been connected through an overlay WireGuard2 network. These three offloading destinations allow us to investigate offloading workloads under different scenarios in which resource richness and the distance between database client running workloads and databases servers hosting data are varied.

Our main criteria to select NoSQL and relational databases in this study are popularity, usage, and commercialization by well-known cloud providers. Thus, we selected Cassandra,3 MongoDB,4 Redis,5 and MySQL.6 These databases are often evaluated only in terms of throughput, response time, and scalability in both private and public clouds [4], [5], [6], [7], [8]. These metrics are not enough for database selection because resource consumption is crucial for low-powered devices in edge–cloud scenarios. Therefore, we measured the resource utilization of these databases during workload offloading from edge nodes to powerful computing nodes. For the purpose of our study, we primarily focus on database client node resource utilization.

1The answer to “when” a task should be transferred is beyond the scope of this article. Interested readers are referred to [3].
2WireGuard: https://www.wireguard.com.
3Cassandra: https://cassandra.apache.org.
4MongoDB: https://www.mongodb.com.
5Redis: https://redis.io.
6MySQL: https://www.mysql.com.
The consumption of resources we focus on is energy, network bandwidth, and storage. Energy consumption is a key cost function in offloading because edge devices commonly have limited battery life, which depletes quicker under high load [9]. We measure the energy consumption of CPU, RAM, and the rest of the system (i.e., SSD, ports, screen, and so on). Bandwidth consumption of the database client node refers to the amount of data transferred during the task offloading [10]. The amount of bandwidth consumed impacts both response time and potential traffic costs. Storage cost is another essential metric in the edge–cloud framework due to increasing volumes of data generated by IoT devices. This metric refers to the data storage consumption of edge node or remote servers where the offloaded task is performed [11]. Therefore, we investigate how efficient is a database in terms of resource consumption (energy, bandwidth, and storage) for offloading various workloads under different scenarios that are different in resource richness, connection types, and distance between database client and servers.

To conduct the above investigation, we leveraged multiple RPi’s, a laptop (termed edge node hereafter), a high-performance adjacent server (termed edge server node henceforth), and a cluster of VMs in a hybrid cloud. We also considered both WiFi and cable connections between database client and servers. Our experimental scenarios are defined in two categories: 1) offloaded scenarios in which the client node is deployed in resource-constrained nodes and database servers are hosted at richer computing nodes and 2) nonoffloaded (local) scenarios in which database client and servers are residing on the same computing node. We evaluated these scenarios from a resource consumption perspective using different tools. To measure energy consumption, we relied on Intel’s running average power limit (RAPL) technology [12]. We also used iPerf3 and iftop network tools to measure the traffic between database client that runs the YCSB workloads [13] and database servers that host data. We used the standard df utility to measure storage consumption on database servers.

Our contributions are threefold.

1) We present a modular edge–cloud framework in which the whole process of cloud infrastructure deployment/destruction, database installation, and database cluster configuration is performed in a fully automated manner.

2) We evaluate resource usage in terms of energy, network bandwidth, and storage to explore the feasibility of workloads offloading for DDBs in the edge–cloud framework.

3) We finally discuss our experimental findings.

## II. RELATED WORK

To position the novelty of our work with respect to the state of the art, we divided the related studies into the following categories. Table I compares these notable studies.

### Performance Evaluation of Distributed Databases on Clouds: With the advent of NoSQL databases, researchers conducted a variety of experimental evaluations and achieved notable results from a performance perspective. Rabl et al. [4] presented a comprehensive performance evaluation in terms of throughput, latency, and disk usage for six modern databases on two different private clusters using the YCSB workloads. Kuhlenkamp et al. [5] evaluated the correlation between scaling speed and throughput for Cassandra and...

---

**TABLE I**

| Paper | Application | Infrastructure | Databases | E | R | B | S |
|-------|-------------|----------------|-----------|---|---|---|---|
| [4]   | DDB³        | Private cloud  | Cassandra, HBase, Redis, Voldemort, VoltDB, MySQL | ✗ | ✗ | ✓ | ✓ |
| [5]   | NDB²        | Public cloud   | Cassandra and HBase | ✗ | ✓ | ✗ | ✗ |
| [14]  | NDB²        | Public cloud   | MongoDB, Cassandra, Riak | ✗ | ✓ | ✓ | ✓ |
| [6]   | DDB²        | Private cloud  | MongoDB, RavenDB, CouchDB, MySQL, Cassandra, HyperTable, Couchbase | ✗ | ✓ | ✓ | ✓ |
| [7]   | DDB²        | Hybrid cloud   | Cassandra, MongoDB, Riak, CouchDB, Redis, MySQL | ✗ | ✓ | ✓ | ✓ |
| [17]  | NDB¹ Server(s) | Cassandra, MongoDB | ✓ | ✓ | × | × |
| [8]   | DDB² BD° Single node | Cassandra, HBase, HBase, Hadoop | ✓ | ✓ | × | × |
| [19]  | General Edge(RPis) | NA | ✓ | ✓ | × | × |
| [20]  | RDB³ Fog | PostgreSQL | ✓ | ✗ | ✓ | ✗ |
| [21]  | General RPis | Hadoop, Spark | ✓ | ✓ | ✓ | ✗ |
| [22]  | General RPis | Hadoop, Spark | ✓ | ✓ | ✓ | ✗ |
| [23]  | DDB² Rpis | MongoDB, SQLite, LevelDB | ✓ | ✓ | ✗ | ✓ |
| Our work | DDB² Edge-cloud | MongoDB, Cassandra, Redis, MySQL | ✓ | ✓ | ✓ | ✓ |

1) DDB stands for distributed database and includes both MySQL and NoSQL databases. 2) NDB stands for NoSQL database and includes only NoSQL databases. 3) BD stands for Big Data. 4) Storage I/O bandwidth has been measured.

---

³We also presented resource utilization in terms of per-operation efficiency. This simplifies comparing efficiencies of different databases in addition to raw performance.

⁸Iperf3: https://iperf.fr.

⁹iftop: https://linux.die.net/man/8/iftop.
and update transactions in PostgreSQL. Hajji and Tso [21] presented the performance of Hadoop Distributed File System (HDFS) on a single RPi and a 12-node RPi cluster and Scolati et al. [22] demonstrated CPU and RAM usage for the same frameworks, however, on a containerized RPiS cluster. We measured the resource consumption of DDBs to provide insight into the suitability of workload offloading in an edge–cloud environment.

Computation Offloading Toward Edge Computing: Offloading compute-intensive tasks has attracted researchers’ attention to optimize response time and reduce energy consumption through different optimization techniques. Vu et al. [30] provided a joint offloading and resource allocation framework for hierarchical cooperative fog computing nodes to optimize energy consumption using an improved branch-and-bound algorithm. Pei et al. [31] studied energy-efficient resource allocation through latency-sensitive tasks offloading in mobile-edge computing (MEC). Khan et al. [32] used integer-linear programming to offload tasks from a mobile node to MEC to optimize energy consumption and latency. Li [33] and Mehrabi et al. [34] studied numerical optimization approaches and Huynh et al. [35] combined such approaches with data caching to reduce battery energy and latency in MEC. In contrast to these studies, Cañete et al. [36] proposed the implementation of offloading decisions based on tasks and infrastructure for mobile IoT applications to reduce energy usage.

Recently, researchers proposed AI-based offloading approaches to optimize energy consumption and response time. La et al. [37] developed human- and device-driven intelligent algorithms for offloading tasks to reduce energy consumption and latency in edge computing. Zhou et al. [38] proposed ML-based dynamic offloading and resource scheduling to save energy in mobile edge nodes. Breitbach et al. [39] presented an ML-based code offloading to reduce energy for edge devices. Lan et al. [40] and Chen et al. [41] achieved a reduction in energy consumption and latency through a combination of ML-based tasks offloading and caching data. Reinforcement learning (DRL) [42], [43], [44], Markov-based [45], [46], partial code offloading [29], intelligent collaboration for computation offloading [47], and data synchronization and management via offloading techniques [48], [49] have proposed to improve response time, energy, and bandwidth consumption.

All the above studies investigated optimization techniques to make decisions on either partial or full offloading for computational tasks, including video rendering, gaming, etc. Also, these solutions have been evaluated through simulation in which tasks are defined based on the required CPU cycles, amount of memory, and bandwidth. In contrast, we investigated resource consumption for emerging NoSQL databases. The closest work to ours is [23], which differs in infrastructure scale and databases selected ([23] in Table I). The rest of the studies listed in Table I primarily focus on performance and scalability rather than measuring resource consumption in terms of energy, bandwidth, and storage as the key offloading factors. Our work is complementary to these works, as we evaluated the resource consumption of

\[10\] HBase: https://hbase.apache.org.
\[11\] Hive: https://hive.com.
\[12\] Hadoop: https://hadoop.apache.org.
\[13\] ODROIDC2: https://www.hardkernel.com.
DDBs for offloaded workloads from resource-constrained to resource-rich devices.

III. DESIGN AND IMPLEMENTATION OF EDGE–CLOUD FRAMEWORK

This section discusses the design and implementation of our edge–cloud framework.

A. Implementation of Edge–Cloud Framework

We designed a layered edge–cloud framework [Fig. 1(a)]. The bottom layer is hardware infrastructure that consists of edge nodes, RPis, and VMs in a hybrid cloud. The middle layer is network connection that includes WireGuard to build an overlay network across different nodes. The top layer of the framework is VMs and edge deployment in which we used Terraform to deploy VMs in a hybrid cloud. The output of this layer is a set of VM IPs, which enables the deployment of DDBs across computing nodes to measure resource utilization. We discuss two bottom layers in this section and the topmost layer in the next section.

The network connectivity topology and individual link throughput used within our experiment sets are detailed in Fig. 1(b). To have reproducible resource deployment in a hybrid cloud, we used Terraform.\(^{14}\) Despite the illusion of unlimited resources available in clouds, the increased network latency may negatively impact the real-time analysis of large amounts of data. In addition, potential costs increase as well as privacy challenges inherent to cloud environments need to be taken into account. Thus, we deployed RPis and edge nodes, where their computing and storage resources form a hierarchy with regard to resource richness [Fig. 1(b)]. To have richer resources in edge computing, we built a cluster of eight RPis connected through a Gigabit switch.

To make a network connection across all computing nodes in the edge–cloud framework, we leveraged WireGuard which is faster and more cost efficient compared to the VPNs provided by commercial cloud providers [7]. A key metric of network connection strength is the network throughput (measured in terms of data transferred and received per second). We leveraged Iperf3\(^{15}\) to measure the throughput between the two end nodes in both directions. We installed this tool on all nodes and ran it for 10 min to record the throughput between each pair of computing nodes, as labeled on links in Fig. 1(b). As can be seen, the network connection between VMs in the private cloud achieves the highest throughput of 3.57 Gb/s, whereas this value for VMs in the public cloud holds the second rank and is of 998 Mb/s. The reason behind such values is the VMs in the private cloud may reside on the same server, while in the public cloud, VMs might be provisioned in different servers or even different racks. In contrast, the lowest network throughput is observed across private and public clouds (24.6 Mb/s), and the master RPi and the broker VM in the private cloud (34.4 Mb/s).

B. Implementation of Resources Consumption Probes

We discuss the following resource consumption probes as summarized in Table II.

Energy Consumption Probes: These probes are implemented through both software and hardware tools, which depend on the facilities provided by the computing nodes. For the edge node and edge server node, we provided an edge-node-energy-consumption-probe that leverages energy RAPL to measure the energy consumption of CPU and RAM [12]. For the edge node, we implemented a battery-probe, which exploits Upower\(^{16}\) command to measure the battery depletion of the edge node. Based on these two probes, we measured the energy consumed by the rest of the system (i.e., storage, ports, screen, etc.) in the edge node. For the master RPi, we implemented a USB-energy-consumption-probe in which the energy consumption of the master RPi is recorded with the help of the USB power meter (UPM)—WEB U2 model. UPM can provide voltage readings down to 0.01 V and current to 0.001 A, which can be either displayed on the built-in LCD or recorded.

---

\(^{14}\)Terraform: https://www.terraform.io.

\(^{15}\)Iperf3: https://iperf.fr.

\(^{16}\)Upower: https://www.commandlinux.com/man-page/man1/upower.1.html.
in a file. We exploited energy cost meter (ECM) to implement power-socket-energy-consumption-probe to measure the energy of the whole cluster of RPis. ECM measures voltage and the current range of 200–276 V ac and 0.01–10 A, respectively. For the virtualized resources in the hybrid cloud, we did not provide energy measurement probes for two main reasons: 1) the depletion of energy resources of the edge computing nodes is crucial in the context of edge computing and 2) it is almost impossible to measure the energy consumption of a server for individual tasks in a cloud since each server provides multitenant services.

**Bandwidth Consumption Probe:** This probe captures the amount of data transferred and received between nodes. This measurement is implemented through iftop, which monitors the ingress and egress bandwidth of a network interface. This service is termed bandwidth-consumption-probe and we activated it on the network interfaces of computing nodes issuing and receiving operations.

**Storage Consumption Probe:** This probe measures the consumed storage during workloads execution against a particular database. We used the standard df to implement the storage-consumption-probe. We activated this service during experiments on the disk hosting the database.

Once the workload runs on the client node, the probes start to measure the consumed resources. Upon finishing the execution of a workload, the probes are stopped and results are collected for analysis.

### IV. DISTRIBUTED DATABASES AND WORKLOADS

We discuss databases under evaluation and workloads used.

**A. NoSQL and Relational Databases**

We evaluate Mongo and Cassandra as document-based NoSQL databases [50], [51] and MySQL as the most-used relational database in the industry sector. In addition, we chose Redis as an in-memory database for evaluation.

**B. Workloads**

We used YCSB workload (v0.15.0)\(^\text{17}\) to evaluate both NoSQL and relational databases. The YCSB workload facilitates a set of tunable parameters and acts on a loose schema including a string key assigned to a collection of fields, which themselves are the string to binary blob key–value pairs. The YCSB workload consists of elementary operations, such as read, write, and insert for a record based on a single key.

YCSB also supports a complicated “scan” operation, which refers to a paging operation starting from a particular key. Due to these advantages, our experiments targeted six core workloads as summarized in Table III.

We used the default YCSB workload configuration values except for two parameters: the number of records and operations. We adjusted them based on our hardware infrastructure support. For RPis and the edge node, we set up 10K records, while for the edge server node, which is more powerful, we set this parameter to a value of 10M records. Nevertheless, we used a variable value for the number of operations in each workload for RPi, edge node, and edge server node. The reason behind such a setting is that the information about battery depletion of the edge node is updated every 2 min. If we set the number of operations with a small value, then the implemented battery-depletion-probe might record zero for energy consumption. This implies that the workload runs out before updating data regarding to battery depletion. To avoid such an issue, we initially ran the workload for 10K operations and then the number of operations was calculated as throughput achieved for 10K operations multiplied by 1200 s (20 min). This duration time of 20 min for running the YCSB workload gives a good enough precision with respect to the battery depletion information.

### V. PERFORMANCE EVALUATION

In this section, we describe the setup of our edge–cloud framework and delineate our experimental results.

**A. Testbed Setup**

The edge–cloud framework consists of the following computing components as summarized in Table IV.

**Hybrid Cloud:** We built the hybrid cloud on the on-premises infrastructure virtualized through OpenStack at Adelaide University and Azure datacenter in the Sydney region [7]. We exploited clusters of VMs in the hybrid cloud with a size of

\(^{17}\text{YCSB Workload: https://github.com/brianfrankcooper/YCSB.}\)
Based on the cluster size, we considered three combinations of the hybrid cloud configuration settings: (8_0), (4_4), and (1_7). This allowed us to evaluate the hybrid cloud when 1) most nodes sit on either the private or public cloud or 2) nodes are equally distributed on each cloud side. Each VM in the private cloud has 2 vCPUs, 4-GiB RAM, and 40-GiB HDD, and the size of each VM in the public cloud is Standard B1m (1 vCPU, 2 GiB, and 30-GiB HDD).

**RPi Cluster:** We built a homogeneous cluster of 8 RPis 3 Model B+, where each RPi is equipped with a Quad-core CPU, 1-GiB RAM, and 16-GiB microSD storage.\(^{18}\)

**Edge Node:** We deployed two different edge nodes: 1) a laptop, referred to as edge node, has a Quad-core CPU, 16-GiB RAM, and 256-GiB SSD and 2) the high-performance edge server, referred to as edge server node, provides an 8-core CPU, 32-GiB RAM, and 1-TB SSD.

**Experimental Scenarios:** We considered three types of workers and four types of database servers (Table V). We generally offloaded data from the resource constrained to the more powerful computing resources (the database servers). This concept of offloading includes all scenarios except scenarios 1, 7, and 11 in which the worker and the server are the same computing node. Such scenarios termed nonoffloading (local) scenarios provide more insight into the databases in terms of energy consumption when databases are utilized locally. Furthermore, we considered different connection types for the RPi and edge node that give us insight into the effectiveness of databases from an energy consumption perspective as a faster connection (cable versus WiFi) is used (Scenarios 2, 3, 4, 5, 8, and 9). For simplicity of presentation, a scenario of \([A \rightarrow B \ (C/W)]\) indicates that A is a worker and B is a database server, and the connection between them is either Cable or WiFi. The extensive nature and high flexibility of our framework enable us to investigate, evaluate, and provide recommendations on resource utilization of distributed storage systems occurring in real-world scenarios. This includes examples, such as distributed monitoring, data logging for delayed processing, and predictive analytics in the context of smart farming, mobile field operations, and electric power grids.

To evaluate the experimental scenarios, we implemented a modular approach, including three components: 1) controller node; 2) worker/database client node; and 3) database server nodes (Fig. 2). The controller node initially receives IPs of computing nodes as input and then runs installation and cluster configuration of databases across those database servers if needed. At the same time, the controller node communicates with the worker node to set up the probes and runs the YCSB workload. Once the database workloads are sent to the DB server nodes, all resource consumption probes are activated to record the consumed energy, bandwidth, and storage of the worker and server(s). It should be noted that uploading the probes consumes energy, and we, thus, exclude it from the experimental results. We also ran all scenarios without running YCSB for 20 min and measured only the idle energy consumption. Then, this idle energy consumption is subtracted from the one for the corresponding scenario in which the YCSB workload was run.

### B. Experimental Results

This section explains energy, bandwidth, and storage consumption for the scenarios listed in Table V.

1) **Energy Consumption:** We investigate the energy usage [in Joules per Million Operations (J/MOPs)] of different databases.\(^{19}\)

   a) **Energy consumption of single RPi (scenarios 1–6):** Fig. 3(a) shows the energy consumption of Cassandra. For (RPi → RPi), the energy consumption is about 5000 J/MOPs for workloads (A, C, and D) and about 2 and 2.5 times this value for workloads B and F, respectively. As we move to (RPi → edge node (C)), the energy consumption for workloads A, B, and F, respectively, reduces by 28%, 46%, and 78% compared to the ones for (RPi → RPi). In contrast, in the same scenario with the WiFi (W) connection, the energy consumption increases by 190%–363% for all workloads compared to the ones for (RPi → RPi). This implies that faster connections

---

\(^{18}\)Note that a single VM on both private and public clouds possesses a CPU with fewer cores compared to the RPi, but a cluster of VMs provides more CPU cores. The impact of horizontal and vertical scalability of VMs on energy consumption remains as future work.

\(^{19}\)Note that there is a direct correlation between energy consumption and database throughput in all experiments. Though, we did not plot database throughput here due to space constraints.
cause less energy consumption. For \( \text{RPi} \rightarrow \text{edge server node (C/W)} \), Cassandra requires less energy to serve workloads as compared to both discussed scenarios. The value for this scenario decreases between 50\% (Workload A—Cable) and 82\% (Workload B—WiFi) in contrast to the values for (\text{RPi} \rightarrow \text{RPi}). This indicates that more powerful computing resources at a close distance from the worker allow for saving energy.

For (\text{RPi} \rightarrow \text{hybrid cloud}), as the number of VMs in the public cloud increases, the energy consumption of all workloads raises from 10 kJ/MOPs for (8_0) to 25 kJ/MOPs for (1_7). This means the worker requires more time to receive responses from database servers due to a longer distance. All workloads (except F) have the same energy consumption (more than 15 kJ/MOPs) for (1_7), which implies that the energy consumption is dominated by the distance between nodes regardless of the workload. In summary, two out of six scenarios are energy efficient in offloading for Cassandra (Table VI).

Fig. 3(b) illustrates the energy consumption of Mongo. For (\text{RPi} \rightarrow \text{RPi}), RPi consumed energy between 2800 J/MOPs (workload C) and 4500 J/MOPs (workload F), which is 39\%–65\% less than the consumed energy for Cassandra. This fact can be explained by memory swapping occurring on RPi to operate Cassandra due to RAM constraints. In contrast, the relaxation of this constraint through hosting Mongo on the edge node [i.e., \text{RPi} \rightarrow \text{edge node (C)}] with more memory capacity, the energy consumption grows by a factor of (1.05–2.41) against Cassandra. This is because Cassandra utilizes the CPU effectively compared to Mongo, which results from the internal design and implementation of these databases [52].

For WiFi connection, there is no obvious supremacy of Mongo and Cassandra over each other because the network fluctuations have an impact on the execution time of databases, which leads to the increment/decrement of energy consumption. Similarly, we observe the same trend for (\text{RPi} \rightarrow \text{edge server node (C)}) in which Mongo requires more energy by a factor of (at most) 1.44 for workload C in comparison with Cassandra. For (\text{RPi} \rightarrow \text{hybrid cloud}), Mongo, compared to Cassandra, increases energy usage by (30\%–80\%) for (8_0) and by (9\%–47\%) for (1_7). This is because Cassandra is balancing data placement, while Mongo is not.\(^{20}\) In summary, Mongo consumes more energy than Cassandra on average except for the scenario with a memory shortage. Also, the hierarchy of scenarios for Mongo has changed slightly in comparison to Cassandra (Table VI).

Fig. 3(c) depicts the energy usage of Redis, which is significantly less than Cassandra and Mongo for all scenarios (except for [\text{RPi} \rightarrow \text{edge node (W)}]). For (\text{RPi} \rightarrow \text{RPi}), Redis

\(^{20}\)Due to space constraint, we did not present the bandwidth usage across participant nodes for (\text{RPi} \rightarrow \text{hybrid cloud}).
decreases energy consumption by (55%–92%) and (75%–90%) with respect to Cassandra and Mongo, respectively. We can also see the same trend for [RPi→ edge node (C/W)]. Redis consumes (88%–90%) and (55%–92%) less energy than Mongo and Cassandra for [RPi→ edge node (C)]; likewise, (66%–73%) and (60%–63%) for [RPi→ edge server node (C)]. For the WiFi setting, Redis also outperforms Cassandra and Mongo except for workloads B and D [Fig. 3(c)], where we observed instability in connection. In fact, Redis reduces energy consumption by (15%–30%) and (5%–38%) compared to Mongo and Cassandra, respectively, as it is hosted on the edge node (W). Likewise, (28%–76%) and (17%–25%) reduction in energy compared to Mongo and Cassandra when Redis is deployed on the edge server (W). As data is offloaded to the hybrid cloud, Redis outperforms Cassandra and Mongo in energy consumption. For example, the maximum energy consumption by (1.1–1.9) is slightly more than 10 kJ/MOPs for all workloads except F, while for Cassandra and Mongo, this value grows to 15 and 20 kJ/MOPs, respectively. In summary, apart from [RPi→ edge node (W)], Redis outperforms Mongo, which in turn, outweighs Cassandra in energy consumption. The hierarchy of scenarios for Redis is different from the one for Cassandra and Mongo (Table VI).

Fig. 3(d) illustrates the energy usage of MySQL. Results show that for write-related workloads (A and F) under the (RPi → RPi) scenario, MySQL consumes more energy than Redis by (1.5–1.9) times, while less energy than Cassandra and Mongo by (3.74–6.97) and (2.27–2.37) times, respectively. This can be explained by the fact that Redis outperforms MySQL in response time due to its RAM-based nature. In contrast, for the same scenarios, Mongo outperforms MySQL by (1.1–1.9) times in energy consumption. This shows the superiority of Mongo over MySQL in response time.

The [RPi→ edge node (C)] and [RPi→ edge server node (C)] scenarios, respectively, are more energy efficient by 40%–67% and 11%–18% in offloading versus nonoffloading due to the fast CPU and network connection. In contrast, using the same computing nodes with the WiFi connection makes offloading noneffective. Under the same scenarios, MySQL performs worse than Redis, and these scenarios consume (4.05–13.5) and (2.56–3.12) times more energy, respectively. This is because more memory allows Redis to run faster. However, on average, MySQL saves 9% (resp., 18%) energy compared to the deployment of Mongo (resp., Cassandra) on the edge node (resp., the edge server node).

For (RPi → hybrid cloud), unlike the other databases, the configuration of the hybrid cloud does not impact the energy consumption of MySQL. The results show that MySQL consumes 6–10 kJ/MOPs for workloads (A–D) and around 12.5 kJ/MOPs for workload F, which is less than the ones for Cassandra and Mongo and stays competitive with the energy consumption of Redis. The reason behind such results is that MySQL supports strong consistency in a data node group (i.e., two replicas on the private cloud) and then the updated data is asynchronously propagated to other data node groups. Thus, the only latency between RPi and the VMs in the private cloud is reflected in the energy consumption. In summary, Redis outperforms MySQL in almost all scenarios in terms of energy consumption, while MySQL is relatively effective in energy consumption compared to Mongo and Cassandra for (RPi → hybrid cloud). Furthermore, Table VI summarizes the energy consumption of different scenarios from lowest to highest, where the rank of [RPi→ edge server node (W)] and (RPi→ hybrid cloud) is exchangeable based on the workload.

Fig. 4 depicts the energy consumption of workload E for scenarios 1–6. Results show that the energy usage of workload E is higher than the one for the other workloads. This is because workload E is expensive in terms of operations. Cassandra and Mongo, respectively, consume the highest (48.15 kJ/MOPs) and the lowest (13.5 kJ/MOPs) energy under the nonoffloading scenario. When other edge computing nodes host databases, only the edge server node (C) provides promising offloading, where its energy consumption decreases by 73% for Cassandra, 6% for Redis, and 28% for MySQL. Mongo suffers a 15% energy usage increase under the same conditions. This shows that more capacity of RAM accelerates the response time of Cassandra and Redis, which results in energy consumption reduction. For (RPi → hybrid cloud), MySQL operates the best and Redis acts the worst in the case of energy usage with a value of (300–600) kJ/MOPs and (520–600) kJ/MOPs, respectively. This is because Redis transmits more data to the public, while MySQL requires the least.
In the same scenario, Cassandra (at most 140 kJ/MOPs) and Mongo (at most 185 kJ/MOPs) achieve middle ranks in energy consumption.

b) Energy consumption of the edge node (scenarios 7–10): We make the following observations from Fig. 5.
1) Locally running YCSB and databases on the edge node exhibits the lowest energy consumption compared to the nonlocal running. As expected, Redis outperforms all databases in energy consumption (203–312 J/MOPs), while MySQL has the worst performance in energy consumption (480–3650 J/MOPs). This is because Redis is RAM-based, while MySQL has to update data and logs to disk regularly compared to NoSQL databases.
2) As databases are deployed on the edge server node (C/W), Redis and MySQL still exhibit the lowest (301–521 J/MOPs for cable and 4170–7610 J/MOPs for WiFi) and the highest (382–556 J/MOPs for cable and 7880–13110 J/MOPs for WiFi) energy consumption, respectively. The ratio of energy consumption for WiFi to Cable is (10–25) times for Cassandra, (1.5–25) times for Mongo, (10–16.5) times for Redis, and (1.5–10) times for MySQL. These values exhibit that the faster connection between the worker and the database servers is, the less energy consumption.
3) The more nodes reside on the private cloud, the less energy is consumed for all databases except MySQL. The energy consumption of Cassandra, Mongo, and Redis, respectively, is at the level of 60, 80, and (80–100) kJ/MOPs for workloads (A–D) with the configuration of (7_0). With the same condition and workloads, the energy consumption for (8_0) drops by (38%–76%) for Cassandra, (25%–37%) for Mongo, and (27%–34%) for Redis. These results show that the highest reduction happens for Cassandra since it requires reading and writing data on a quorum of replicas. The energy consumption of Workload F is more than workloads A–D so the ratio is (1.3–2.5) times for Cassandra, (3–4.8) times for Mongo, (1.36–1.73) times for Redis, and (1.96–2.7) times for MySQL. As more VMs are used on the public cloud, this factor drops significantly, which means running all workloads across WAN is expensive.

In summary, as databases are deployed on the edge and edge server nodes (C/W), Redis and MySQL consume the lowest and highest energy, respectively, followed by Mongo and Cassandra. In contrast, for the edge server node (W), there is no preference between Mongo and Cassandra in energy consumption. Furthermore, only under the [edge node → edge server node (C)] scenario, offloading is effective for MySQL (Table VII).

Table VII: Sorted List of the Lowest to the Highest Energy Consumption for Scenarios 7–10

| Database | Energy Consumption (kJ/MOPs) |
|----------|-------------------------------|
| Cassandra | 60–80 (A–D), 38–76 (8_0) |
| Mongo | 60–80 (A–D), 25–37 (8_0) |
| Redis | 60–80 (A–D), 27–34 (8_0) |
| MySQL | 60–80 (A–D), 1.96–2.7 (8_0) |

There is no particular hierarchy among hybrid cluster configurations for MySQL, and we denoted Hybrid cloud (all) in the table.
Fig. 6 shows the energy consumption of workload E for scenarios 7–10. Running workload E on the edge node consumes the lowest energy for Mongo (1980 J/MOPs) and the highest for Redis (11,420 J/MOPs). For offloading data to other computing resources, Redis still needs the highest energy (19,246 kJ/MOPs) and even more (1361–2455 kJ/MOPs) as it is deployed on the edge server (C/W) and hybrid cloud, respectively. Cassandra has the lowest energy consumption on the edge server [Fig. 6(a)] because it transmits fewer data across WAN to satisfy quorum consistency (Appendix B, Table 1, workload E in the supplementary material).

c) Energy consumption of the edge server node (scenarios 11 and 12): We evaluated the energy consumption of databases for scenarios 11 and 12, where the edge server node is the database worker. We observed the same trend of energy consumption for different databases so that the more computing nodes are close to the worker, the less energy is consumed. Similarly, workload E is the most expensive workload for all databases (see Appendix A in the supplementary material).

d) Energy consumption of RPi cluster (scenario 13): Fig. 7 plots the energy consumption of worker and database servers running on a cluster of 8-RPis. Fig. 7(a) shows that Cassandra’s energy usage is the highest compared to Mongo and Redis. Cassandra requires 80–97 kJ/MOPs to run write-related workloads. This value drops by 2–6 kJ/MOPs for read-related workloads. This is likely due to the memory swapping required by Cassandra (2 GiB), which reduces the speed of writing operations. In contrast, Redis is the most energy efficient (74–214 J/MOPs), while Mongo is in the middle position (390–918 J/MOPs) for all workloads, except E. For workload E, the position of Redis and Mongo changes, since Redis generally requires longer data transfer between computing nodes to serve workload E, which leads to longer execution time, which causes, higher energy consumption [see 7(b)]. Table VIII summarizes the above discussion.

e) Breakdown of the energy consumption of edge node: Fig. 8 breaks down the energy consumption of the edge node including CPU, RAM, and the rest of the system (monitor, peripheral devices, ports, etc.)—termed by REST—for workload E. Simply, the energy consumption of “REST” is the energy measured through Upower utility for battery depletion minus the one through RAPL for CPU and RAM. Results show that the energy consumption of RAM was the lowest (<7%) for most of the scenarios and databases. Thus, CPU and REST have the most contribution to the energy consumption of the edge node. Interestingly, when databases are hosted locally, the energy consumption of the CPU made a significant contribution of ≈60% to the whole consumed energy, while the energy consumption of REST is 25%–39%. As databases are moved into the edge server node and hybrid cloud, this percentage of energy consumption decreases for CPU and increases for REST. This is because the worker spends energy even during waiting to receive a response from database servers. For example, to run Cassandra on edge server node (C), CPU and REST, respectively, are 35% and 57% of the whole energy consumption, while these values, respectively, changed to 14% and 77% as the edge server

---

21 MySQL results are skipped (see Section VI for details).

22 Results for other workloads are skipped due to space constraints.
node (W) was deployed. This is because the worker waits longer to receive a response from the edge server node through WiFi compared to cable. This waiting time increases the energy consumption of REST, while CPU is idle without using significant energy. We also see the same trend with other cluster configurations, where more VMs in the public cloud wait for longer, thus, leading to increased REST energy consumption.

2) Bandwidth Consumption: This section presents the amount of data transferred (TX) and received (RX) between the private and public clouds (Fig. 1), where a worker is the edge node.\textsuperscript{23} Fig. 9 depicts TX and RX in bytes per operation for the OpenStack broker subnet only since these metrics are symmetric for both subnets. Clearly, the values of TX and RX are zero for (8_0) due to all nodes being in the same cloud.

Based on Fig. 9, we can make several observations.

1) As expected, TX and RX for workload E are at the level of several KB per operation, while for the other workloads, the values are at the level of several hundred bytes per operation. This confirms that workload

\textsuperscript{23} Due to space constraints, we only consider edge node as a worker.
E is the most expensive in execution time and energy usage.

2) The values of TX and RX for (4_4) are less than the ones for (1_7), which implies that more nodes in the public cloud cause higher TX and RX values. This is another confirmation of higher energy consumption for (1_7) compared to (4_4) and (8_0).

3) The TX values are less than TX values for most databases and cluster configurations since TX includes the request issued from the worker and RX is the response returning from the database server. Thus, we focus on RX for (1_7) and (4_4).

MySQL has the lowest RX values compared to the other databases for all workloads for (1_7). This is because, as summarized in Tables in Appendix B in the supplementary material, the worker mostly exchanges data with the nodes in the private cloud. Hence, we have fewer data transferred across clouds for MySQL. For the same configuration, Mongo possesses the highest RX values, followed by Redis and Cassandra. This is because, as summarized in Appendix B, Table 1 in the supplementary material, Mongo mostly sends data to the public cloud (node 7) while Redis and Cassandra almost equally spread data across nodes in the hybrid cloud.

For (4_4), Mongo and MySQL serve the read-related workloads through the private cloud since they should satisfy eventual and strong consistency, respectively. Strong consistency is provided by MySQL because the default replica number for MySQL is two (with both in the private cloud). For workloads A and F, Mongo and Redis obtain the highest RX values. For workload E, Cassandra and Redis generate the most traffic on the WAN while MySQL and Mongo transmit less. This is because MySQL and Mongo serve workload E locally, while Cassandra and Redis spread data across nodes (Appendix B, Table 2, workload E in the supplementary material).

3) Storage Consumption: This set of experiments plots the storage consumption (measured in bytes/operation), where the edge node is a worker and the hybrid cloud is a database server. Due to space constraints, we only report results of write-related and scan workloads. For (1_7) and (8_0), Fig. 10(a) and (c) shows that Mongo is the worst in terms of storage consumption compared to other databases for write-related workloads, where workload A uses storage space more than workload F. This is because Mongo uses a document-based data model with full replication as the default setting. In contrast, for the same configuration, MySQL is the most efficient database in storage consumption (35–57 Bytes/Ops for workload A versus 29–43 Bytes/Ops for workload F) due to using two replicas rather than full replication for Mongo and three replicas for Cassandra. Fig. 10(b) exhibits the storage consumption of databases for workload E, which is more than the one for the write-related workloads. Redis uses the largest amount of storage, followed by Mongo with a 20% reduction. This correlates with a high RX value of workload E for Redis. Cassandra and MySQL stay close to each other with the lowest storage usage. Table IX summarizes discussed results.

VI. DISCUSSION

We discuss findings, practical experiences, and technical challenges that we encountered during experimentation.

Research Findings: From the discussed evaluated experiments, it is a challenging problem to select a specific database solution that incurs the lowest resource consumption (energy, bandwidth, and storage) in an edge–cloud framework for all workloads. However, from the results, we have extracted several insights as follows.

1) In terms of offloading, a few scenarios make data offloading profitable in terms of energy usage. Indeed, if database operations are offloaded from source-constrained edge nodes to powerful computing nodes with high bandwidth and low latency connection, then we expect to save energy for edge devices [e.g., RPi → edge server (C)].

2) Connection bandwidth and latency have a direct impact on the energy usage of data offloading. Hence, all databases exhibit less energy consumption with a faster connection between workers and data servers.

3) The limitation of memory can increase the energy consumption of disk-based databases such as...
Cassandra because memory swapping further increases the response time, which directly impacts energy consumption.

4) The distance between worker and database nodes, and the spread of data across computing nodes in a cluster of VMs in a hybrid cloud are two key factors that affect the response time, which results in energy consumption increment. In other words, the greater is the distance between worker and database servers, the more is energy consumption. The more data is distributed among nodes in the hybrid cloud, the less energy is consumed. This is because more operations can be served through the private cloud, as seen in the case of Cassandra and Redis.

5) The energy consumption of CPU and RAM has the highest and lowest contribution, respectively, in the total energy consumption. This is likely the reason why Redis is superior to disk-based databases in terms of energy consumption in most cases.

With respect to the superiority of databases to each other, Redis consumes the least amount of energy followed by Cassandra if an edge computing node supports a high amount of memory capacity. This superiority is also valid when we run workloads A–F locally (i.e., on RPi, edge node, and edge server node), and offload these workloads from the database worker to the edge node and edge server nodes. In contrast, for workload E, Redis performs the worst in energy consumption, while MySQL requires the least energy on average. For offloading data from the database worker to the hybrid cloud, MySQL consumes the lowest energy, followed by Redis particularly when more nodes are deployed on the public cloud. With regard to bandwidth usage across clouds, MySQL transmits the least amount of data irrespective of a cloud configuration. This correlates with MySQL using less energy in hybrid cloud scenarios compared to other databases. We can also see that MySQL and Cassandra require the lowest storage capacity.

Practical Experiences: While we automated the installation and configuration of the databases across cloud and edge use cases, the ARM architecture of RPi caused some issues with MySQL. The default MySQL server package provided by Ubuntu 20.04.1 does not come with clustering components included. Thus, we had to compile our own version with the clustering explicitly enabled. Compiling on RPi node itself was failing as more than 12 GB of RAM was required to complete the build process. Enabling a swap file allowed us to proceed, however, the resulting build performance was acceptably slow, requiring several days to complete. Thus, we also attempted to cross-compile ARM binaries on a high-end x86_64 server, which was significantly faster. Unfortunately, both produced packages crashed upon execution on RPi nodes due to the lack of L3 cache. Upon a brief MySQL source code inspection and assessing the time constraints, we skipped MySQL test for RPi nodes. Further investigation of this issue and related code changes might be useful in the future. This is a prime example of the reasons behind DDBs being unsuitable in the context of resource-constrained devices. This can motivate further database development geared toward lightweight deployments.

We used RAPL which exploits a software power model to estimate energy usage of the edge node and edge server node through hardware performance. The main issue with this utility is the maximum energy range of 65 Billion Micro-joules for its counter. This imposes constraints on the duration of the experiment for each workload because when the energy consumption reaches this value, the counter resets, and consequently the energy consumption probe records a wrong value. Hence, we had to take extra care to adjust the counter values to compensate for this limitation.

VII. Conclusion

Selecting a suitable DDB to deploy across the edge–cloud framework is not a trivial task as overall performance and energy efficiency highly depend on a multitude of factors. To disclose these factors, we conducted an extensive evaluation of DDBs through a variety of scenarios in which operations are issued from resource-constrained computing nodes to more powerful ones via cable and WiFi connections. We implemented these scenarios through a modular framework to achieve flexibility and accuracy in experimental data. Our evaluation quantified the impact of connection speed, latency, and the computational power of database servers on various types of resource utilization. Notably, our results exhibit that the distance (and hence latency) between the database client issuing operations and the database servers hosting databases is a major factor that should be considered. Similarly, the bandwidth usage in the edge–cloud framework greatly impacts the client’s energy consumption. We see that Redis generally consumes the least amount of energy for most workloads in local and edge-offloaded processing due to being RAM-based. For offloading data to the hybrid cloud (higher latency), MySQL is the most efficient in energy consumption for most workloads on average since it transmits fewer data across private and public clouds. Mongo and Cassandra hold a rank after MySQL and Redis in terms of energy usage, where Cassandra commonly outperforms Mongo when more nodes reside on the public cloud.

Future Work: We conducted our experiments for particular physical and virtualized resources in the edge–cloud framework. However, repeating these experiments for all existing and new flavors of physical and virtual resources is daunting work and to a large extent is impossible. To tackle this challenge, we can leverage AI and ML to discover patterns of resource utilization in the edge–cloud landscape based on the data collected in our experimental scenarios [53]. This can aid in predicting whether full offloading of database workloads from edge to cloud should be conducted. While we empirically measured resource utilization of databases under non/full-offloading, partial offloading, and optimal resource management might save energy consumption of DDBs in the edge–cloud framework [46], [54]. Furthermore, we can also exploit ML models to find a correlation between resource consumption in a wide range of computing devices and custom database parameter settings, such as replication number, consistency model, and data size in order to analyze offloading possibility more precisely. Ultimately, we can create ML
models to predict resource utilization given a combination of hardware, database parameters, and distance between database client and servers. This can enable determining when and where to offload database workloads in a given configuration. Finally, we can evaluate big data frameworks (e.g., Spark and Flink) using our experimental framework to find potential correlations between parameter settings and resource utilization. This may provide further insights into the feasibility of data processing on edge devices compared to sending and processing data in a centralized cloud.

REFERENCES

[1] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in Proc. 5th Int. Joint Conf. INC IMS IDC, Aug. 2009, pp. 44–51.

[2] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, “ENORM: A framework for edge node resource management,” IEEE Trans. Services Comput., vol. 13, no. 6, pp. 1086–1099, Nov./Dec. 2020.

[3] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, “Toward computation offloading in edge computing: A survey,” IEEE Access, vol. 7, pp. 131543–131558, 2019.

[4] B. Rahil, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, “Solving big data challenges for enterprise application performance management,” Proc. VLDB Endowment, vol. 5, no. 12, pp. 1724–1735, Aug. 2012.

[5] J. Kuhlenkamp, M. Klems, and O. Röss, “Benchmarking scalability and elasticity of distributed database systems,” Proc. VLDB Endowment, vol. 7, no. 12, pp. 1219–1230, Aug. 2014.

[6] Y. Li and S. Manoharan, “A performance comparison of SQL and NoSQL databases,” in Proc. IEEE Pac. Rim Conf. Comput. Commun. Signal Process., Aug. 2013, pp. 15–19.

[7] Y. Mансови, В. Прокхоренков, and M. А. Бабар, “An automated implementation of hybrid cloud for performance evaluation of distributed databases,” J. Netw. Comput. Appl., vol. 167, Oct. 2020, Art. no. 102740.

[8] T. Li, G. Yu, X. Liu, and J. Song, “Analyzing the waiting energy consumption of NoSQL databases,” in Proc. IEEE 12th Int. Conf. Depend. Auton. Secure Comput., 2014, pp. 277–282.

[9] W. Chen, D. Wang, and K. Li, “Multi-user multi-task computation offloading in green mobile edge cloud computing,” IEEE Trans. Services Comput., vol. 12, no. 5, pp. 726–738, Sep./Oct. 2019.

[10] W. Yu et al., “A survey on the edge computing for the Internet of Things,” IEEE Access, vol. 6, pp. 6900–6919, 2018.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

[12] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, “RAPL in action: Experiences in using RAFL for power measurements,” ACM Trans. Model. Perform. Eval. Comput. Syst., vol. 3, no. 2, pp. 1–26, 2018.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Rakamrishnan, and R. Sears, “Benchmarking cloud serving systems with YCSB,” in Proc. 1st ACM Symp. Cloud Comput., 2010, pp. 143–154.

[14] J. Klein, I. Gorton, N. Ernst, P. Donoho, K. Pham, and C. Matser, “Performance evaluation of NoSQL databases: A case study,” in Proc. 1st Workshop Perform. Anal. Big Data Syst. Proc., 2013, pp. 5–10.

[15] V. Abramova and J. Bernardino, “NoSQL databases: MongoDB vs cassandra,” in Proc. Int. Conf. Comput. Sci. Softw. Eng., 2013, pp. 14–22.

[16] Y. Mансови and M. A. Бабар, “The impact of distance on performance and scalability of distributed database systems in hybrid clouds,” 2020, arXiv:2007.15826.

[17] D. Mahajan and Z. Zong, “Energy efficiency analysis of query optimizations on MongoDB and cassandra,” in Proc. 5th Int. Green Sustain. Comput. Conf. (IGSC), 2017, pp. 1–6.

[18] B. Bani, “Understanding the impact of databases on the energy efficiency of cloud applications,” Ph.D. dissertation, Dept. Comput. Eng. Softw. Eng., Univ. Montreal, Montreal, QC, Canada, 2016.

24Spark: https://spark.apache.org.
25link: https://hlink.apache.org.
[42] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, “Fast adaptive task offloading in edge computing based on meta reinforcement learning,” IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 1, pp. 242–253, Jan. 2021.

[43] C. Qu et al., “DroneCOCONet: Learning-based edge computation offloading and control networking for drone video analytics,” Future Gener. Comput. Syst., vol. 125, pp. 247–262, Dec. 2021.

[44] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, “Collaborate edge and cloud computing with distributed deep learning for smart city Internet of Things,” IEEE Internet Things J., vol. 7, no. 9, pp. 8099–8110, Sep. 2020.

[45] C. Chen, Y. Zhang, Z. Wang, S. Wan, and Q. Pei, “Distributed computation offloading method based on deep reinforcement learning in ICV,” Appl. Soft Comput., vol. 103, May 2021, Art. no. 107108.

[46] S. Liu, Y. Yu, L. Guo, P. L. Yeoh, B. Vucetic, and Y. Li, “Adaptive delay-energy balanced partial offloading strategy in mobile edge computing networks,” Digit. Commun. Netw., to be published.

[47] J. Kang, S. Kim, J. Kim, N. Sung, and Y. Yoon, “Dynamic offloading model for distributed collaboration in edge computing: A use case on forest fires management,” Appl. Sci., vol. 10, no. 7, p. 2334, 2020.

[48] T. Wang, J. Zhou, A. Liu, M. Z. A. Bhuiyan, G. Wang, and W. Jia, “Fog-based computing and storage offloading for data synchronization in IoT,” IEEE Internet Things J., vol. 6, no. 3, pp. 4272–4282, Jun. 2019.

[49] Y. Qiu, G. Liu, B. A. Muthu, and C. B. Sivaparthipan, “Design of an energy-efficient IoT device with optimized data management in sports person health monitoring application,” Trans. Emerg. Telecommun. Technol., vol. 33, no. 10, p. e4258, 2022.

[50] Y. Mansouri, A. N. Toosi, and R. Buyya, “Data storage management in cloud environments: Taxonomy, survey, and future directions,” ACM Comput. Surv., vol. 50, no. 6, pp. 1–51, 2017.

[51] J. Han, H. E. G. Le, and J. Du, “Survey on NoSQL database,” in Proc. 6th Int. Conf. Pervasive Comput. Appl., Oct. 2011, pp. 363–366.

[52] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system,” ACM SIGOPS Oper. Syst. Rev., vol. 44, no. 2, pp. 35–40, 2010.

[53] S. S. Gill et al., “AI for next generation computing: Emerging trends and future directions,” Internet Things, vol. 19, Aug. 2022, Art. no. 100514.

[54] S. S. Gill, P. Garraghan, and R. Buyya, “ROUTER: Fog enabled cloud based intelligent resource management approach for smart home IoT devices,” J. Syst. Softw., vol. 154, pp. 125–138, Aug. 2019.

Yaser Mansouri received the Ph.D. degree from the Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Melbourne, VIC, Australia, in 2017. He is a Researcher with the Centre for Research on Engineering Software Technologies, The University of Adelaide, Adelaide, SA, Australia. His research interests cover the broad area of data management in cloud storage services. Dr. Mansouri was awarded a First-Class Scholarship supporting his Ph.D. studies.

Victor Prokorenko received the Ph.D. degree in computer science from The University of South Australia, Adelaide, SA, Australia, in 2017. He is a Researcher with the Centre for Research on Engineering Software Technologies, The University of Adelaide, Adelaide. He has more than 14 years of experience in software engineering with main areas of expertise, including the investigation of technologies related to software resilience, trust management, and big data solutions hosted within OpenStack and Microsoft Azure cloud platforms.

Faheem Ullah received the Ph.D. degree from The University of Adelaide, Adelaide, SA, Australia, in 2020. He was a Postdoctoral Fellow with The University of Adelaide, where he is a Lecturer and a Cyber Security Program Coordinator. He is also a member of the Centre for Research on Engineering Software Technologies, The University of Adelaide. His research and teaching interests include big data analytics, cyber security, software engineering, and cloud computing.

Dr. Ullah is a two times Gold Medalist and a one-time Silver Medalist, and a receiver of six academic distinctions.

Muhammad Ali Babar received the M.Sc. degree in computing sciences from the University of Technology Sydney, Ultimo, NSW, Australia, in 1999, and the Ph.D. degree in computer science and engineering from the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW, Australia, in 2007. He is a Professor with the School of Computer Science, The University of Adelaide, Adelaide, SA, Australia. He leads a theme on architecture and platform for security as a service in Cyber Security Cooperative Research Centre (CSCRC), a large initiative funded by the Australian government, industry, and research institutes. He is the Technical Project Lead of one of the largest projects on “Software Security” in ANZEC region funded by the CSCRC. SOCRATES brings more than 75 researchers and practitioners from six research providers and four industry partners for developing and evaluating novel knowledge and AI-based platforms, methods, and tools for software security. After joining The University of Adelaide, he established an interdisciplinary research centre called CREST, Centre for Research on Engineering Software Technologies, where he directs the research, development, and education activities of more than 25 researchers and engineers in the areas of software systems engineering, security and privacy, and social computing. His research team draws a significant amount of funding and in-kind resources from governmental and industrial organizations. He has authored/coauthored more than 275 peer-reviewed research papers at premier software journals and conferences. More information can be found at http://malibabar.wordpress.com.