applied sciences

MDPI

Article

EDCrammer: An Efficient Caching Rate-Control Algorithm for Streaming Data on Resource-Limited Edge Nodes

Yunkon Kim and Eui-Nam Huh *

Department of Computer Science and Engineering, Kyung Hee University, 1732, Deogyeong-daero, Giheung-gu, Yongin-si, Gyeonggi-do 17104, Korea; ykkim@khu.ac.kr

* Correspondence: johnhuh@khu.ac.kr; Tel.: +82-31-201-3778

Received: 3 May 2019; Accepted: 20 June 2019; Published: 23 June 2019

Abstract: This paper explores data caching as a key factor of edge computing. State-of-the-art research of data caching on edge nodes mainly considers reactive and proactive caching, and machine learning based caching, which could be a heavy task for edge nodes. However, edge nodes usually have relatively lower computing resources than cloud datacenters as those are geo-distributed from the administrator. Therefore, a caching algorithm should be lightweight for saving computing resources on edge nodes. In addition, the data caching should be agile because it has to support high-quality services on edge nodes. Accordingly, this paper proposes a lightweight, agile caching algorithm, EDCrammer (Efficient Data Crammer), which performs agile operations to control caching rate for streaming data by using the enhanced PID (Proportional-Integral-Differential) controller. Experimental results using this lightweight, agile caching algorithm show its significant value in each scenario. In four common scenarios, the desired cache utilization was reached in 1.1 s on average and then maintained within a 4–7% deviation. The cache hit ratio is about 96%, and the optimal cache capacity is around 1.5 MB. Thus, EDCrammer can help distribute the streaming data traffic to the edge nodes, mitigate the uplink load on the central cloud, and ultimately provide users with high-quality video services. We also hope that EDCrammer can improve overall service quality in 5G environment, Augmented Reality/Virtual Reality (AR/VR), Intelligent Transportation System (ITS), Internet of Things (IoT), etc.

Keywords: resource-limited edge node; lightweight and agile caching; advanced PID controller; qualified caching-as-a-service

1. Introduction

The computing paradigm is headed toward the edge of the network to address the limitations of cloud computing, such as latency between a central datacenter and a user device and load concentrated on a central datacenter. Gartner, the world's leading research and advisory company, mentioned "Cloud to the Edge" as one of its Top 10 Strategic Technology Trends for 2018 and included "Empowered edge" in its 2019 list. The emergence of the edge computing paradigm has generated active movements to redesign the network, increase coverage, boost network capacity, and cost-effectively bring content closer to users. Cloud and network experts forecast that this paradigm will contribute significantly to the emerging 5G environment. Within those active movements, caching techniques could be a key factor since the advantages of caching can fulfill requirements of edge computing. For example, caching can reduce unnecessary data transfers, thereby reducing the cost of network charges; reduce network bottlenecks, thus allowing pages to load quickly without increasing bandwidth; reduce requests to the origin server, thus reducing the load on the server and allowing it to respond faster; and reduce delays

caused by distance. Therefore, caching as a service should be considered so that caching techniques for edge nodes can be used more easily by service developers, end-users, and even between cloud service providers (CSPs). From a multi-cloud perspective, CSPs can collaborate and provide cloud services through caching. To do that, CSPs demand clear metering and billing methods for caching as a service.

Edge nodes has its own limitations, and the most important of those is that edge nodes have relatively lower computing resources than a cloud datacenter. The reason is that it is difficult to allocate a lot of resources to the edge of the network due to cost effectiveness which has been an important since the introduction of cloud computing. That is why edge nodes deployed in multiple locations need to be miniaturized, energy efficient, and cost-effective. From the viewpoint of improving cost efficiency by taking advantage of existing resources, base stations or home WiFi routers could be improved and thereby become edge nodes. However, current research has not adequately considered the performance of edge nodes. Instead, it has studied applying relatively heavy tasks to edge nodes, such as machine learning [1–6], high-quality video streaming [2–5,7–14], and AR/VR [1,3,7]. Furthermore, existing caching research has focused on pre-deploying data to an edge node without fully considering the dynamic scale-up and -down of the cache [3,15–19], which are related to the pay-as-you-go pricing policy of cloud computing [20,21].

As the computational limitation of edge nodes, the lightweight, agile, and optimized data caching algorithm for edge nodes is highly necessary to meet the needs of cloud service developers, end-users, and CSPs. We here propose an efficient caching rate-control algorithm, EDCrammer (Efficient Data Crammer), that considers the performance of edge nodes and the optimal cost of the cache used. EDCrammer performs agile operations by using the enhanced PID (Proportional-Integral-Differential) controller, which includes three techniques, namely overshooting prevention, min-max regulation of PID output, and variation control of PID output, to tune PID controller for data caching. Accordingly, EDCrammer can control the caching rate for streaming data on resource-limited edge nodes by transferring appropriate amounts of data. The main contribution of this algorithm is that EDCrammer requires less computing resources than machine learning based caching, and reaches the desired cache utilization quickly. Therefore, this efficient data caching algorithm is suitable to apply for resource-limited edge nodes. Furthermore, we hope that EDCrammer will be well integrated with the service routing capability that efficiently deploys cloud services to users, and can be useful to better meet the quality of service (QoS) required by cloud services, such as AR and VR, autonomous vehicle, ITS, and IoT.

2. Related Works

This section provides a comprehensive review of the field, describes a consideration for resource-limited edge nodes, and illustrates the analysis of state-of-the-art edge caching techniques. Above all, an overview of latency and load congestion issues in cloud computing is described from the structural viewpoint. The different viewpoints of edge-related technologies that can solve those issues are analyzed to illustrate a relationship between the edge-related technologies through features and to show those important for the emerging 5G. Consideration for resource-limited edge nodes are described, and the state-of-the-art edge caching techniques are analyzed and described.

2.1. Latency and Load Congestion Issues in Cloud Computing

Cloud computing is almost mature; however, definitional nuances still differ slightly among cloud service vendors, users, researchers, developers, and so on. Standard recommendations (ITU-T Y.3500 and Y.3502) are helpful in allowing cloud people to imagine cloud computing in a compatible manner because the recommendations provide a definition of cloud computing and role-based representations, such as cloud service providers (CSPs), cloud service customers (CSCs) and cloud service partners (CSNs) [22,23]. The conventional cloud has centralized structure, which leads to network congestion [24]. Cloud services are also sensitive to network latency because they are Internet-based. Mobile cloud computing based on a public cloud causes CSCs to experience long

latency because it uses a wide area network [25,26]. The response latency experienced by CSCs is one of the most important performance metrics in cloud computing [14,27,28]. Latency within a cloud is an important issue and latency reduction approaches are adequately studied inside and outside of the cloud [24,29,30]. Therefore, the next-generation cloud environment must reduce network latency, which will require the conventional cloud environment to be geographically distributed onto the edge of the network using both cloud and network techniques.

2.2. Edge-Related Technologies Analysis

To resolve latency and load congestion issues in conventional centralized cloud computing, computing resource decentralization has been widely studied and developed via many approaches, such as Fog Computing, Mobile Edge Computing, Cloudlet, and Edge Computing. Table 1 summarizes current edge-related technologies. The edge-related technologies demand the cooperation of cloud and network technologies. Network nodes are usually under the control of network providers. The computing resources on the edge of the network can be defined as a set of network nodes and computing nodes in close proximity to the user-side. On the other end, CSPs apply cloud technologies to IoT gateways. Combining those approaches can reduce network latency and increase the QoS [2–6,8–13,31–37].

Table 1. A summary of edge-related technologies.

Issued by	Concept	Definition
CISCO 2011	Fog Computing (FC)	• ends the cloud to the edge of networks, in particular wireless networks for the Internet of Things (IoT). • pushes the computing applications, data, and services away from centralized nodes to the network edge, enabling analytics and knowledge generation to occur close to the data sources.
ETSI 2014	Mobile Edge Computing (MEC)	• pushes the cloud computing capabilities close to the Radio Access Networks in 4G, 5G. • renamed to Multi-access Edge Computing in 2016.
Carnegie Mellon Univ. 2013	Cloudlet	• is to “bring the cloud closer to the users”.

However, each of those techniques views the problem slightly differently, and this can interfere with the understanding of edge nodes in this paper. Therefore, an analysis of relationship between the edge-related technology is necessary. Edge-related technologies are still being studied including distributed storage and computational resources or infrastructure [1,38] and edge, regional, and central clouds within a network topology [39]. “Edge Computing” is often used as a generic term [8]. Figure 1 shows the key features available within conventional cloud computing and edge-related technologies. Most of the technologies currently in research and development will deploy infrastructure closer to users based on virtualization technology of cloud computing. Multi-access Edge Computing could be more relevant to radio access networks (RANs) than other technologies. Edge Computing is more concerned with IoT than any other technology. Fog Computing could be aimed at providing reliable services in certain areas, and Cloudlet is considered to be the cornerstone of edge-related technologies [3,5,8,9,31,32,37]. Based on the analysis, we derived a resource-limited edge node, which has minimal cloud and network features, such as virtualization, context awareness, user-proximity, and network compatibility.

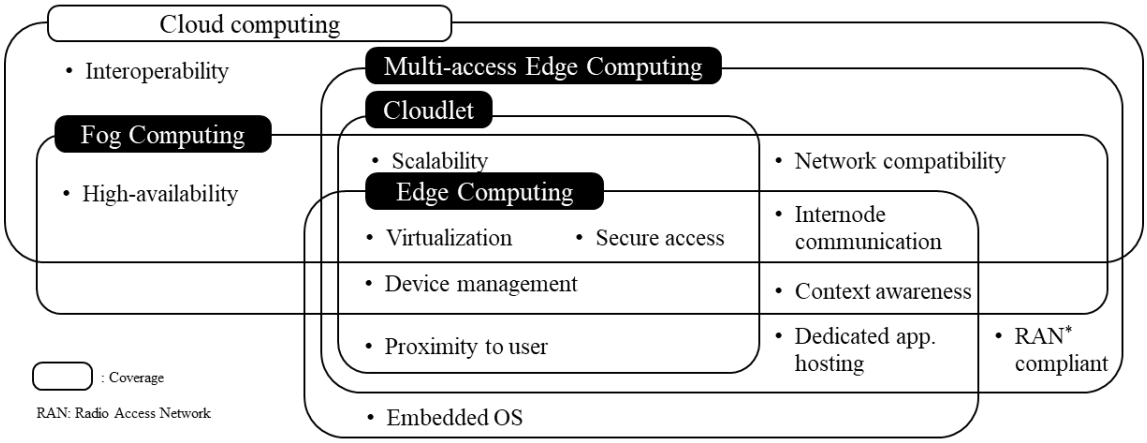


Figure 1. The coverage of key features among conventional cloud computing and edge-related technologies [3,5,8,9,31,32,37].

In addition, the edge-related technologies are obviously important as next-generation computing technologies. By 2024, mobile technology subscriptions will reach 8.9 billion, of which 1.5 billion will be 5G, and video will account for around 74% of mobile data traffic [40]. The amount of mobile data traffic in 2022 (75.7 exabytes per month, 1 EB ≈ 1,000,000 TB) will be 6.6 times higher than in 2017 (11.2 EB) [41]. Enterprises and academia are actively working to settle the traffic of wireless access networks so that it can emerge as 5G by 2020. As an integral part of 5G, the convergence of network and cloud computing technologies is essential to providing a variety of services, including dynamic content delivery, AR and VR, connected vehicles, and video streaming. The integration of cloud computing technology into network nodes is under study to mitigate load congestion and reduce latency. To do this, network softwarization technology is also required. Software-Defined Networking (SDN) enables network nodes to have computing capability, allowing them to run small tasks, such as virtualized network functions and micro-services [3]. As a result, cloud services can float on distributed sites, including both cloud and improved network nodes, which are nodes.

2.3. Cost-Efficiency Considerations for Edge Nodes

Edge-related technologies are being researched and developed in many fields, but cost efficiency in implementation often appears to have been forgotten. Since the introduction of cloud computing, which provides on-demand services, the use of an external cloud and individual cloud deployments have had to be determined by a cost-effectiveness analysis [42,43]. In fact, small and medium enterprises still consider cost-efficiency for cloud adoption and need a lightweight cloud to reduce the cost of cloud adoption. We consider cloud economics in terms of Capital Expenditure (CapEx) and Operating Expenditures (OpEx) [44], and those same considerations should be applied to edge-related technologies. The edge-related technologies are deployed remotely, away from the management of the administrator. Accordingly, edge node hardware must be smaller and more energy-efficient than traditional datacenter hardware. In addition, edge nodes are deployed in multiple locations, so cost-effective deployment is important [7]. Therefore, we consider that edge nodes require lightweight algorithms, services, etc.

2.4. Edge Caching and Edge Data Management Techniques Analysis

Caching data to the edge of the network has been researched extensively to address network traffic management. Edge caching techniques need to consider requirements for lightweightness because an edge node has relatively low computing resources. Low-power devices, such as embedded systems, have begun to have multiple cores, and on-chip caches are under study as on-chip caches can consume half of a system's total energy, which is very expensive. Improving the PID controller, Zhao et al. [45] could create optimal banks for each thread and thus reduce the energy use of the

shared cache by 39.7%. In a low-power device, caching methods using lightweight algorithms are important because the resources or energy consumed by the cache can be a large part of the total consumption. Caching to edge nodes must consider different factors from hardware level caching. For example: What should be cached? How should the limited capacity be used? What is the network status? How should data be delivered to a moving user? Therefore, a new design is sometimes needed for 5G [17]. A relationship analysis and context awareness of Social Network Service (SNS) are used to control the traffic load by applying proactive caching [19]. In addition, cloud data are decomposed into a set of files and services, and replicated to MEC nodes by frequency-based workflow-net [46], or to fog storage sites by predictive fog selection based on trained user's past history [47]. However, the above preactive and proactive methods for data caching would be a relatively computing intensive work to apply to edge nodes, and occupy the storage resource related to the pay-as-you-go pricing policy in resource-limited edge nodes. Thus, it is necessary for a data caching method to perform lightweight computing and to transfer data to edge nodes on demand.

Meanwhile, the normalized delivery time (NDT) is theoretically derived to prove the usefulness of a cache [18], although it is difficult to derive the various metrics into an optimal metric. In addition, the data encoding often uses maximum-distance separable (MDS) codes as an alternative to file fragmentation for energy-efficient edge caching. By using MDS codes, it is possible to obtain the optimal caching location and reduce the backhaul rate [48]. To efficiently manage caches in a 5G Cloud Radio Access Networks (C-RAN) environment, exponential decay and the Analytical Hierarchy Process (AHP) are applied to achieve multi-level mobility and the smallest amount of feedback [15]. Although it is possible to derive a more optimal caching method by considering various factors and a new design, it is more important for an algorithm to be lightweight and agile as edge nodes have limited resources. Accordingly, we propose an efficient caching rate-control algorithm and framework of streaming data to resource-limited edge nodes. We hope the proposed algorithm not only supports high-quality cloud service by reducing network latency and congestion, but also contributes real-time services, such as AR and VR, autonomous vehicles, and ITS.

3. Efficient Caching Rate-Control for Edge Computing

Overall, edge-related technologies are intended to mitigate and resolve the current latency issues between a cloud datacenter and a user, and the load congestion issues on a central cloud datacenter. Edge nodes are deployed in many areas remotely far from the cloud and network administrator so that edge nodes usually have relatively lower computing resources. Furthermore, resource-limited edge nodes embrace cloud computing technologies. Thus, cost-effectiveness analysis is important when deploying edge nodes, and it is generally required that edge nodes and the services be economically optimized. To cope with the mentioned issues in edge-related technologies ad well as to reduce the computational cost on edge nodes, we propose an efficient caching rate-control algorithm, EDCrammer. To remotely control cache in edge nodes, EDCrammer requires the current cache utilization. By the enhanced PID controller, EDCrammer estimates the next amount of live streaming video data for caching. Therefore, it can be lightweight and agile algorithm for data caching, and help improve service performance and utilize minimal resources in edge nodes.

3.1. Concept of the Proposed Edge Caching

EDCrammer is implemented as a framework to efficiently cache data and control the caching rate between a cloud and edge nodes. Extending computing resources to the edge of the network allows cloud services to be deployed to nearby CSCs, who can then receive better QoS than with a conventional, centralized cloud computing structure. As shown in Figure 2, an edge node is newly added between a cloud node and a user device. Cloud services and data can be deployed on the edge node, enhancing service quality and mitigating the load congestion on the central datacenter. However, the edge node has relatively lower computing resources, thus it adopts OS-level virtualization. An efficient caching rate-control algorithm can remotely manage the cache for resource-limited edge

nodes. For example, baseball fans watch a Major League Baseball game. For caching, a cloud node transmits chunks of the game video to the edge node which the baseball fans are connected. The edge node streams the game to the baseball fans. In this process, the EDCrammer can consider the difference in data consumption rate of each user through the cache utilization notification sent from the edge node. By the cache utilization and efficient caching rate-control, EDCrammer can transmit chunks of the game video. In this case, it is very important to dynamically reflect the data consumption rate so that we define a software-defined cache (SDC) that can immediately detect the user's data consumption. Therefore, EDCrammer can efficiently distribute and store data on edge nodes based on errors, defined as a difference between the desired cache utilization and actual cache utilization.

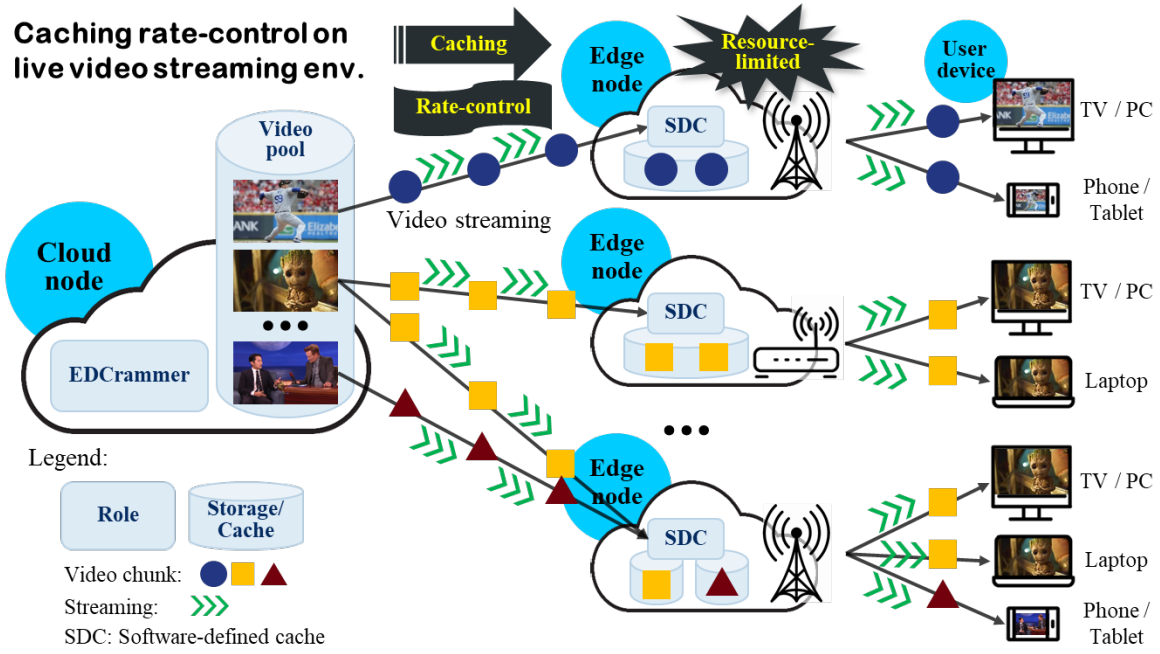


Figure 2. A concept of efficient edge caching between a cloud and edge nodes for live video streaming.

3.2. A Framework for EDCrammer

The framework for EDCrammer controls the data distribution rate using an enhanced PID controller by communication with the SDC dynamically, which has minimum functions. In the framework, a distributed Message Queuing Telemetry Transport (DMQTT) supports communication among EDCrammer, the SDC manager, and a cloud service. Communication by MQTT is performed by the central message broker. If the message broker is located in the cloud, it can lead to long latency between MQTT clients on the edge node and load congestion on the central cloud. Therefore, by a message broker deployed on an edge node, latency and load congestion can be mitigated. In this environment, EDCrammer only requires the current cache utilization from SDC in edge nodes. When receiving the current cache utilization, EDCrammer calculates the error which is a difference between the desired cache utilization and the current cache utilization, and then estimates the next amount of data by computation of the PID controller, and finally transmits data to SDC in edge nodes. Therefore, EDCrammer remotely controls cache utilization in edge nodes

Although rate-control between the cloud and the edge is certainly important, an awareness of how data consumption affects the cache and then coping with user's data consumption as soon as possible is even more important. A conventional cache provides passive read/write storage, making it difficult to dynamically check data production and consumption. The SDC can efficiently monitor all cache operations: creating, reading, updating, and deleting. The SDC can also produce data from EDCrammer and provide it to a service on an edge node or a user device. On the other hand, when a cache is provided as a service in cloud computing, the pay-as-you-go pricing policy must be

considered. If a CSC uses a certain cache capacity, the CSC would get a stable cache hit ratio and pay a little extra cost. For this reason, optimizing cache capacity is important because it allows CSCs to save money and CSPs to save storage resources.

Figure 3 shows the communication among EDCrammer, the SDC manager, the SDC, and a cloud service or application. IP depletion makes it difficult to assign a public IP to all participating entities and establish Peer-to-Peer (P2P) communication. For this reason, MQTT, which performs publish/subscribe-based communications through an MQTT broker, has become the de facto messaging protocol for IoT. We deploy MQTT brokers on both the cloud and edge nodes, in a distributed fashion. Deploying the message broker on the edge node and on the central cloud, our DMQTT can distribute and minimize the traffic load concentrated on the central cloud node as well as reduce the latency among MQTT clients on the edge node. Then, a cloud service or application can use RESTful APIs to get and use data.

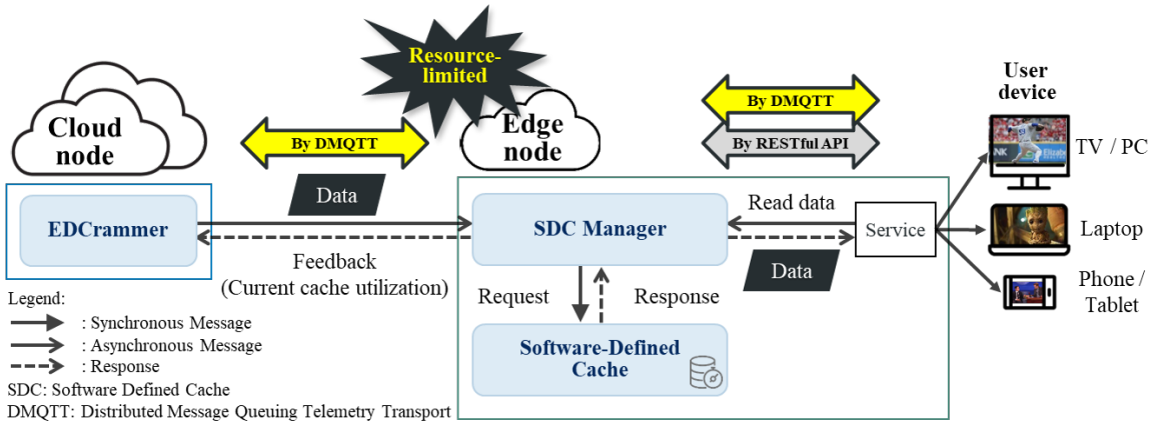


Figure 3. A communication diagram illustrating the interface between entities.

Figure 4 illustrates the overall flows within the framework based on the communication diagram shown in Figure 3. The flows can be divided into the data caching part between the cloud and edge nodes and the data usage part between the edge nodes and user devices. The caching part uses the following steps. (1) First, the SDC manager informs EDCrammer about the cache capacity, the desired cache utilization, and the current cache status. The cache utilization is usually zero at the beginning. (2) An advanced PID controller calculates and predicts the data distribution required to reach the desired cache utilization. The advanced PID controller is described below. (3) The output from the advanced PID controller is checked to prevent over-utilization of the cache capacity and control the rate for caching. (4) EDCrammer splits the data based on the output and (5) distributes it to the SDC manager. (6) The SDC manager identifies the state of the SDC, and (7) stores the data in the SDC. (8) Then, the SDC manager checks the cache utilization and notifies EDCrammer, which initiates a repeat of Steps (1)–(8). The data usage part uses the following steps. Applications running on user devices or others services consume data in the SDC. For our purposes here, we describe both applications and services as apps. (a) An app hits data. (b) The SDC manager reads the data from the SDC, and (c) prepares to process or separate the data. (d) The SDC manager transmits the data to the app. (e) The app then uses the data, and Steps (a)–(e) repeat.

EDCrammer is lightweight and agile for edge nodes, which have relatively lower computing resources than cloud nodes. Both features are well suited to the needs of edge computing: low latency and load distribution. The two features and the programming methodologies used to incorporate them into EDCrammer are described below.

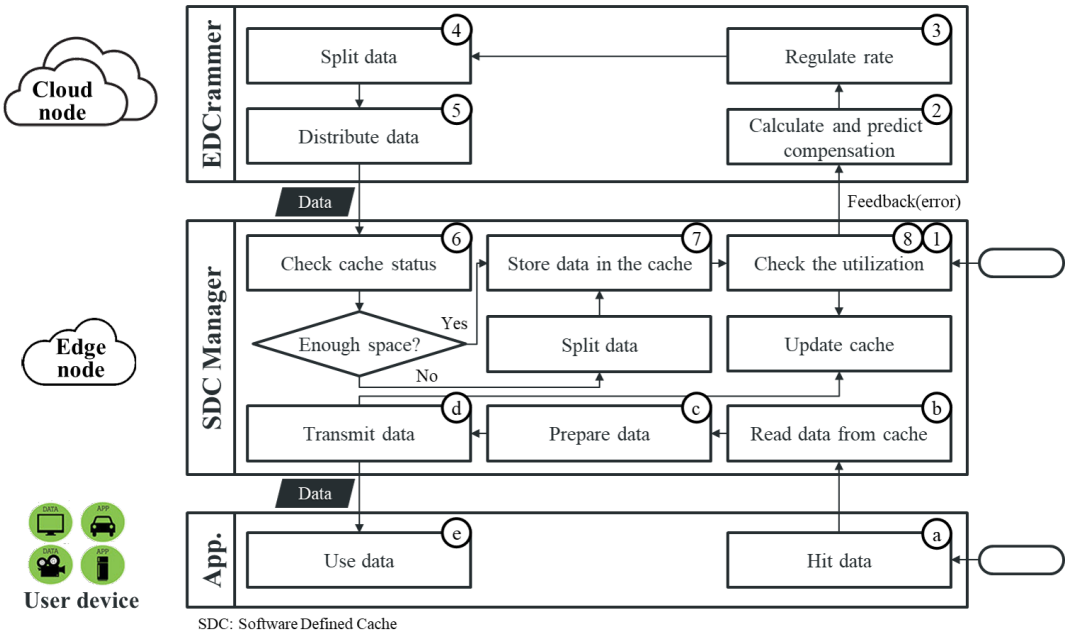


Figure 4. Overall flow for the data caching and hitting process.

Lightweight

- **Rate-of-change-based update operation**: The proposed algorithm considers the rate-of-change of the output of the PID controller. A conventional PID controller updates its output in every cycle. However, if the rate-of-change between the previous output and current output is low, it is unnecessary to update the output every time. Accordingly, we reduce unnecessary updates.
- **SDC with minimum functionality**: The proposed the SDC has minimum functionalities: store_data, read_data, decay_data, and get_cache_status. The store_data function stores data from a cloud node in a cache. The read_data function reads data from the cache upon a hit request from an app. The decay_data function monitors and deletes data in the cache by following a data management policy, such as deletion by late re-hit or re-hit frequency. Data management policies are not covered in this paper.
- **Memory usage minimization with proposed circular list**: To reduce memory usage on the edge nodes, we propose a circular list that uses service-customized list sizes.

In those ways, the EDCrammer algorithm requires lower computing resources than traditional algorithms, making it suitable for edge nodes.

Agile

- **Stream processing-based computation**: We use the stream processing programming methodology, which focuses on the current status because previous data have already been computed. Based on the computed data and current data, the related functions predict the next status.

In that way, EDCrammer performs agile computations, making it useful in reducing the latency of edge nodes.

3.3. Rate-Control Using a Modified PID Controller

Above all, EDCrammer adopts and improves the PID controller. The proportional (P) term provides a fixed percentage close to the setpoint, which is the desired cache utilization. The integral term (I) closes the gap from the proportional term over time. The derivative (D) term responds rapidly

to changes. The well-known advantages of a PID controller are: (1) low resources use; (2) feasibility and ease of implementation without the need for a mathematical model; and (3) ease of tuning gains by simple trial and error. Thus, our PID controller is both lightweight and agile, making it suitable for caching on edge nodes.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

A basic PID controller can be expressed mathematically as given in Equation (1), and it is mainly used in an analog environment. However, caching occurs in a digital environment, thus we modified the mathematical model by applying a stream processing-based computation method, as shown in Equation (2).

$$u(n) = K_p e(n) + K_i \sum_0^n e(k) + K_d \frac{e(n) - e(n-1)}{\Delta iteration} \tag{2}$$

Figure 5 illustrates the advanced digital PID controller proposed in this paper. As mentioned above, the original PID controller was designed to control continuous analog values, mainly an actuator whose value changes gradually until it reaches a desired point. For example, when controlling the speed of an autonomous vehicle, a PID controller effectively controls the actuator. A digital controller requires more precise control because there is no actuator. Existing PID controllers used in grid resource information system support monitoring, managing, deploying resources and services dynamically [49]. The existing PID controllers less consider *termI* overshooting issue and min-max range of PID output. Therefore, we improved the existing PID controllers to add overshooting guard and min-max range, and to consider the PID output variation more sensitively for data caching on resource-limited edge nodes, as shown in Equation (3). The variation control value was set by using *r(n)* (setpoint), K_d gain, and $\alpha(1/100)$. *r(n)* is the desired cache utilization. K_d gain is used to stabilize the variation of the PID. By α, the sensitivity of the proposed algorithm is fine-tuned. If the absolute value of the difference between the current output and the previous output, $|u(n) - u(n-1)|$, of the advanced digital PID controller is smaller than the variation control value, the PID output is not updated because the change is considered insignificant. If it is larger than the variation control value, the change is recognized, and the PID output is updated. After that, the output is regulated from 0 to the maximum cache capacity to reduce the burden of the SDC and control the data transmission rate, as shown in Equation (5). The regulated output or rate-control bit is transmitted to the SDC. The SDC periodically notifies EDCrammer about the current cache utilization. Additionally, *termI* is the sum of the errors. An unregulated error sum could be out of control. Therefore, *termI* is controlled by checking the sign of the current error and the previous output against the saturation of the cache capacity. When the cache is saturated, the proposed algorithm blocks the windup of *termI*. Through the signs of the current error and the previous output, it reacts immediately when the trend changes, as shown in Equation (4). In this study, we used $K_p = 0.7$, $K_i = 0.5$ and $K_d = 0.7$ for each gain of P, I, and D, respectively.

In addition, the output is regulated from 0 to the maximum cache capacity. It is important to consider the amount of data consumed in the SDC when caching because processing the remaining data consumes additional resources. For example, if an app is consuming a small amount of data, and EDCrammer sends more data than it needs, the SDC could become saturated when it should handle the rest of the data. Thus, sending data about the cache capacity could be problematic. Therefore, EDCrammer limits the data using $u(n) \leq CACHE\_MAX$, as shown in Equation (5). The pseudo-code for this is shown in Algorithm 1. In summary, Algorithm 1 controls the caching rate by computing the amount of data sent to the edge node. In addition, data are not sent for rate-control when data consumption is very low. EDCrammer controls the data transmission rate by using the $0 < u(n)$ in Equation (5). The pseudo-code for this is shown in Algorithm 2. In summary, Algorithm 2 controls the caching rate by making a decision between sending data and sending rate-control bit based on the result of Algorithm 1.

$$|u(n) - u(n-1)| > K_d \times r(n) \times \alpha \tag{3}$$

$$sign(e(n)) == sign(u(n-1))\ AND\ isReachedToLimit(u(n-1)) \tag{4}$$

$$0 < u(n) \leq CACHE\_MAX \tag{5}$$

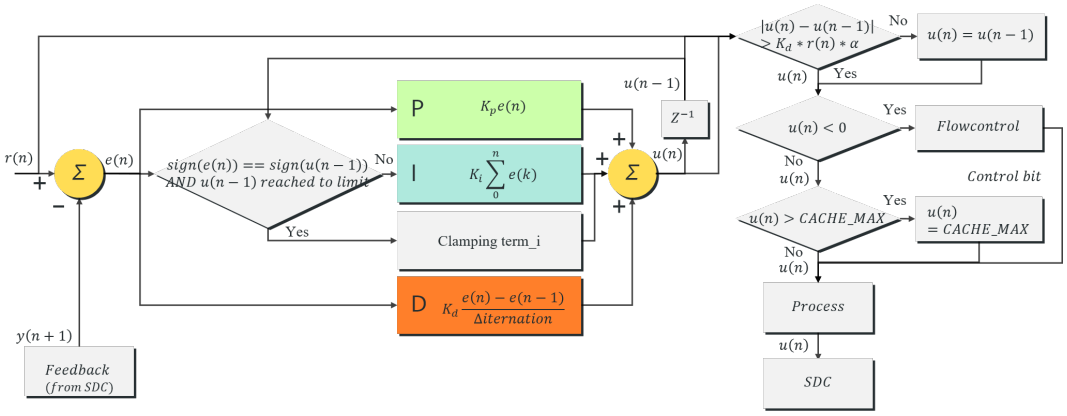


Figure 5. An advanced PID controller.

Algorithm 1: compute() algorithm of EDCrammer to control caching rate for data amount.

Input : Current cache utilization from the edge node
Output: The amount of calculated and regulated data for caching (u(t) or u(t-1))

```
kP = 0.7, kI = 0.5, kD = 0.7

deltaCounter = counter - lastCounter
lastOutput = output

error = setpoint - feedback
termP = error

if signOfLastOutput == signOfError AND isReachedToLimit then
    if termI < -overshootingGuard then
        termI = -overshootingGuard
    if termI > overshootingGuard then
        termI = overshooingGuard
else
    termI += error
end

deltaError = error - lastError
termD = deltaError/deltaCounter

output = kP * termP + kI * termI + kD * termD

if abs(output - lastOuput) > setpoint * kD/100 then
    lastError = error
    lastCounter = counter
else
    output = lastOutput
end
signOfLastOutput = sign(output)

if output < outputMin then
    output = outputMin
if output > outputMax then
    output = outputMax

if output == outputMin OR output == outputMax then
    isReachedToLimit = True
else
    isReachedToLimit = False
end
return output
```

Algorithm 2: transmission_control() algorithm of EDCrammer to control caching rate for data transmission.

```
Input  : The amount of calculated and regulated data from algorithm 1
Output : The amount of calculated and regulated data or rate-control bit
if output != 0 then
    readData()
    sendData()
else
    sendRateControlBit()
end
```

In EDCrammer, which has the advanced PID controller, data caching is performed mainly to enhance performance and meet the required QoS. Data are stored in the SDC, and the cloud service hits the data. In other respects, data production and consumption occur around the SDC, as shown in Figure 6. As data consumption increases, data production could increase, and vice versa. Therefore, to achieve efficient rate-control, an appropriate method is needed to store data in the SDC. Accordingly, the SDC has three data states: (1) stored; (2) hit; and (3) discarded. Figure 6 shows a data life cycle on the SDC. In the data production state, data are stored in the SDC. In the data consumption state, a cloud service hits the data, and then data are discarded. However, because the data could be reused, they are not discarded immediately. It is important to control the rate between data production and consumption. Put another way, it is important that rate-control be performed efficiently between the cloud node and the edge node. EDCrammer is an efficient rate-control algorithm that provides lightweight, agile caching.

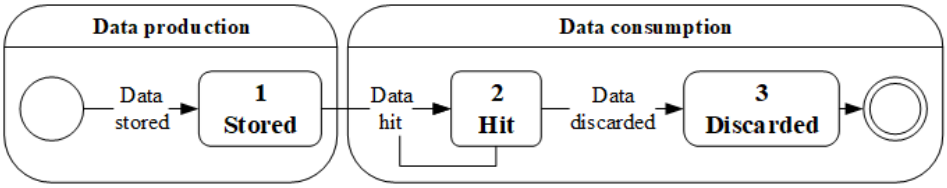


Figure 6. A state diagram of the data life cycle on the framework.

4. Performance Evaluation, and Result Analysis and Discussion

This section details the process used for the performance evaluation and the analysis of the results. The evaluation environment and specifications of the cloud and edge nodes are illustrated. The results of the performance evaluation are analyzed and described. Challenges of this study are discussed.

4.1. Evaluation Environment

Figure 7 shows the environment we used to evaluate the performance of EDCrammer-based caching. To configure a cloud node, we installed Unbuntu 14.04 and Docker engine 17.03-2-ce on a physical server. EDCrammer has an MQTT client and PID controller and was developed in Python 3 and then containerized to run on the Docker engine. To deploy MQTT server, we used an eclipse-mosquitto container. To configure an edge node, Raspbian Stretch Lite 4.14 and Docker engine 18.09.0 were installed on a Raspberry Pi 3 Model B+. the SDC manger has an MQTT client and the SDC. The SDC manager was also developed in Python 3 and containerized to run on the Docker engine. Within that environment, we used four experimental scenarios to evaluate the proposed rate-control algorithm for in- and outflows and caching of streaming data on resource-limited edge nodes.

Table 2 shows the specifications for the cloud and edge nodes. In this performance evaluation, we selected a physical server as the cloud node and a Raspberry Pi 3 Model B+, which has lower computing resources than the cloud node, as the edge node.

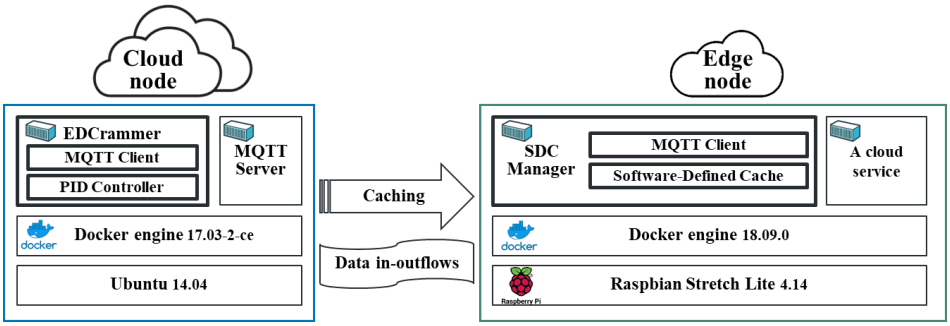


Figure 7. Performance evaluation environment for EDCrammer-based caching.

Table 2. Specifications for the cloud and edge nodes used in the performance evaluation.

	Cloud Node	Edge Node
Hardware	HP ProLiant DL320e Gen8	Raspberry Pi 3 Model B+
CPU	Intel® Xeon® E3-1220v2 (3.1 GHz/4-core/8 MB/69 W)	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4 GHz
RAM	16 GB	1GB LPDDR2 SDRAM
Storage	HDD - HP MB0500GCEHE 500GB	Micro SD Card—Samsung EVO 32 GB
I/O speed	750 MB/s	48 MB/s
Ethernet	HP Ethernet 1Gb 2-port 330i Adapter	Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)

4.2. Experimental Scenarios and Assumptions

For the performance evaluation, we used four experimental scenarios that we performed over the Internet, which has few restrictions, using Cubic congestion control. We considered a live video streaming service (1080 p: resolution, 1920×1080; bitrate range, 3000–6000 Kbps) as a target cloud service. We assumes that: (1) the cloud service and SDC are deployed in an edge node; (2) the cloud service consumes data (video chunks) from SDC; and (3) the data consumption and cycles are different, as shown in the scenarios below. We tested the convergence time by measuring the time required to converge to the desired cache utilization. We tested the stability of the algorithm by measuring the variation in the desired cache utilization during the whole experimental period. Each experiment was tested 30 times. When each experiment was run 10 times, we concluded that the results of each experiment were similar. However, performance tests were conducted up to 30 times to ensure objectivity. The experimental scenarios are as follows.

- **Scenario 1—Periodic and equal**: This scenario tested a case in which a cloud service periodically consumes a single, specific amount of cached data. In this test, the cloud service consumes 512 KB of data from the SDC every 30 ms. This scenario could be ideal because the processing time and throughput can be changed through performance interference in the OS-level virtualization environment. Accordingly, we used this scenario as a control group for comparison with Scenarios 2–4.
- **Scenario 2—Periodic and unequal**: This scenario tested a case in which a cloud service periodically consumes different amounts of cached data. In this test, the cloud service consumes 128 KB–1 MB of data from the SDC every 30 ms. This scenario assumes that the time period is tightly controlled, and the data size changes. This scenario is an experimental group compared with Scenario 1.
- **Scenario 3—Aperiodic and equal**: This scenario tested a case in which a cloud service aperiodically consumes a single, specific amount of cached data. In this test, the cloud service consumes 512 KB of data from the SDC every 8–80 ms (12–120 fps). This scenario assumes that the amount of data is tightly controlled, and time period changes. This scenario is an experimental group compared with Scenario 1.

- **Scenario 4—Aperiodic and unequal**: This scenario tested a case in which a cloud service aperiodically consumes different amounts of cached data. In this test, the cloud service consumes 128 KB–1 MB of data from the SDC every 8–80 ms (12–120 fps). AR/VR requires 120 fps, and video streaming generally requires more than 30 fps. This scenario simulates the way that typical cloud services consume data. This scenario is an experimental group compared with the others.

4.3. Experimental Results and Analysis of the Results

This section shows and analyzes the experimental results. The three main metrics in our experiments were convergence time, convergence deviation, and cache hit ratio. Through the convergence time and deviation, we show the quickness and stability of EDCrammer. Through the cache hit ratio, we show the effectiveness of this framework. Tables 3–5 show representative performance evaluation results from 30 experiments.

As a lightweight rate-control algorithm, EDCrammer adopts and enhances the existing PID controller. EDCrammer can provide agile rate-control for in- and outflows and caching of streaming data on resource-limited edge nodes. As shown in the graphs of the results, the three algorithms for all four scenarios reached the desired cache utilization (*setpoint*) within a few iterations. Furthermore, the result graphs show that EDCrammer immediately computed and compensated for the amount of caching data by using the current cache utilization (*feedback*). It is possible to maintain the stability of cache utilization according to the *setpoint*. Therefore, we conclude that EDCrammer quickly reaches and maintains the desired cache utilization in any situation.

Table 3. Representative performance evaluation results of basic PID algorithm.

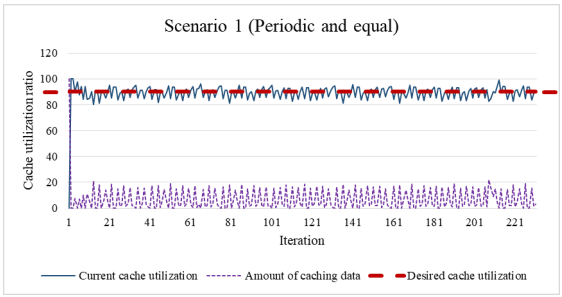
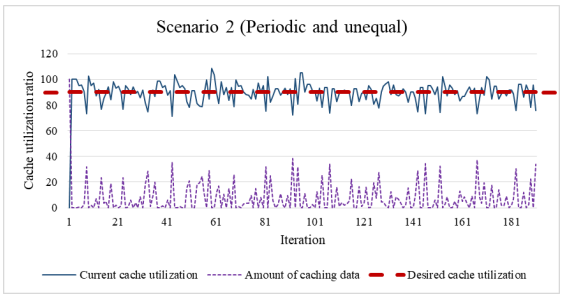
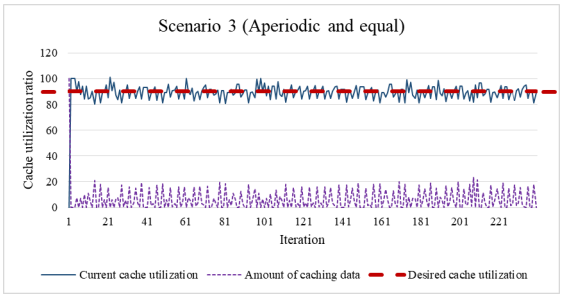
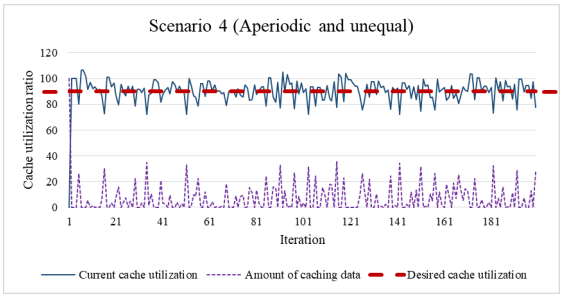
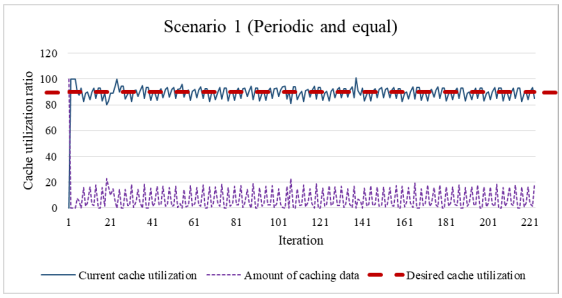
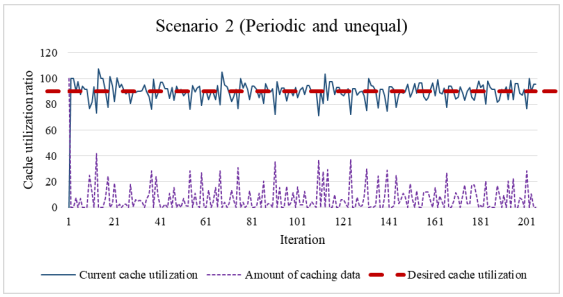
Basic PID Algorithm	
	
(a)	**(b)**
	
(c)	**(d)**

Table 4. Representative performance evaluation results of existing algorithm.

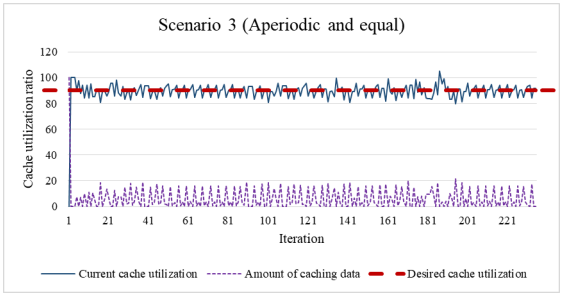
Existing Algorithm



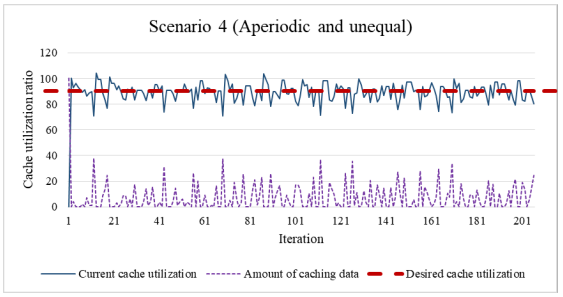
(a)



(b)



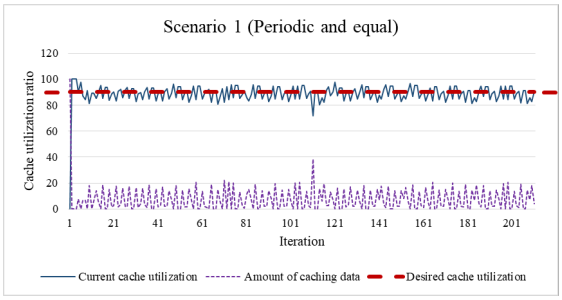
(c)



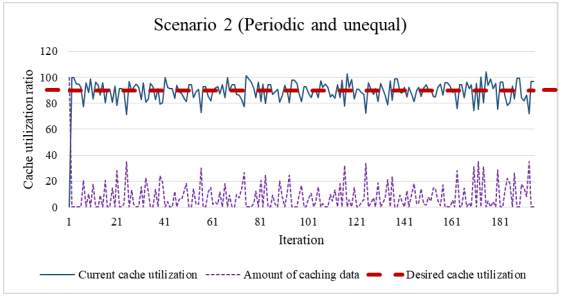
(d)

Table 5. Representative performance evaluation results of proposed algorithm.

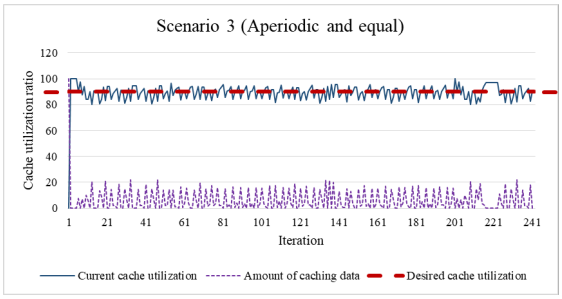
Proposed Algorithm



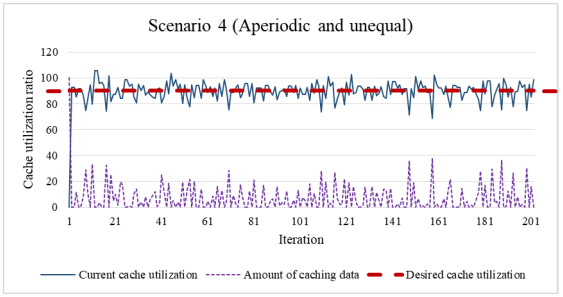
(a)



(b)



(c)



(d)

The PID controller performs agile computation using only the difference between the feedback and the setpoint to reach the desired cache utilization. The average convergence time is when the error between the feedback and setpoint is within 5%. Due to the nature of the algorithm, the distribution

based on the setpoint is suitable to represent the performance of EDCrammer, thus we used that to check its stability of EDCrammer.

$$error_i = setpoint - feedback_i \tag{6}$$

$$STDEV = \sqrt{\frac{\sum_{i=1}^{n}(feedback_i - setpoint)^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(error_i)^2}{n}} \tag{7}$$

The performance evaluation was conducted 30 times, and the average of the standard deviation represents the average convergence deviation for the existing and proposed algorithms as follows.

$$\overline{STDEV} = \frac{\sum_{i=1}^{n} STDEV_i}{n}, (n = 30)$$

Table 6 shows the average performance of the existing and proposed algorithms over 30 experiments. Each experiment performed a different number of caching operation for about 30 s. A caching operation (*an iteration*) takes 0.15 s on average. Experiments using the basic PID algorithm were performed as a basis for comparison. The existing algorithm that reflects the rate of change in PID output has an advantage in environment with low rate of change. The proposed algorithm also reflects the rate of change in the PID output and increases sensitivity. This algorithm has an advantage in dynamically changing environment. In Scenarios 1 and 2, the existing algorithm showed better performance than the proposed one and the basic PID one because those are relatively stable scenarios compared to Scenarios 3 and 4. In detail, in Scenario 1, the average convergence time of the existing algorithm was the fastest of the three. It was 27% faster than that of the proposed one and 24% faster than that of the basic PID one. The average convergence deviation of the existing algorithm was the most stable of the three. It was 9% more stable than that of the proposed one and 5% more stable than that of the basic PID one. In Scenario 2, the average convergence time of the existing algorithm was the fastest of the three. It was 8% faster than that of the proposed one and 85% faster than that of the basic PID one. The average convergence deviation of the existing algorithm was the most stable of the three. It was 1% more stable than that of the proposed one and 2% more stable than that of the basic PID one. On the other hand, the proposed algorithm showed better performance than the existing one and the basic PID one in Scenarios 3 and 4. In Scenario 3, the average convergence time of the proposed algorithm was the fastest of the three. It was 14% faster than that of the existing one and 77% faster than that of the basic PID one. The average convergence deviation of the proposed algorithm was the most stable of the three. It was 3% more stable than that of the existing one and 4% more stable than that of the basic PID one. In Scenario 4, the average convergence time of the proposed algorithm was the fastest of the three. It was 38% faster than that of the existing one and 141% faster than that of the basic PID one. The average convergence deviation of the proposed algorithm was the most stable of the three. It was 4% more stable than that of the existing one and the basic one respectively. As a result, the existing algorithm has better performance in periodic environments, whereas the proposed algorithm has better performance in aperiodic environments.

Table 6. Performance evaluation results of basic PID, existing, and proposed algorithms.

Scenario	**1**			**2**			**3**			**4**		
Algorithm	**B**	**E**	**P**	**B**	**E**	**P**	**B**	**E**	**P**	**B**	**E**	**P**
Average running time (sec)	30.19	30.17	30.20	30.18	30.21	30.20	30.20	30.20	30.17	30.17	30.23	30.20
Average interation (n)	225	216	199	182	179	165	231	229	224	192	198	195
Average *sec/iteration*	0.13	0.14	0.15	0.17	0.17	0.18	0.13	0.13	0.13	0.17	0.15	0.15
Average convergence time	1.04	0.84	1.06	1.87	1.01	1.10	1.43	0.92	0.81	1.87	1.07	0.77
Average convergence deviation	4.58	4.36	4.77	7.29	7.18	7.27	4.73	4.65	4.53	7.14	7.12	6.87

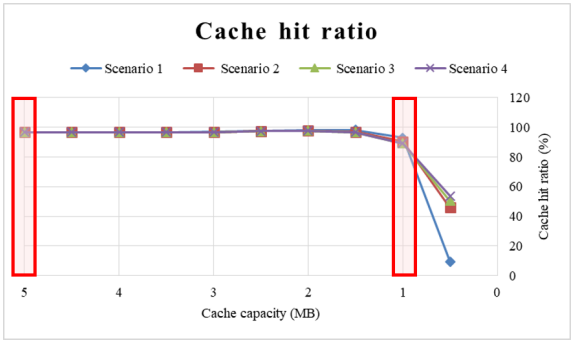
* B: Basic PID algorithm, E: Existing algorithm, P: Proposed algorithm.

Cloud computing provides cloud services anywhere, anytime, on any device. Cloud nodes and edge nodes dynamically receive various user requests and handle them as soon as possible. Caching in a cloud computing environment is also a cloud service that should be done dynamically as soon as possible. Scenario 4 represents that dynamic case. Thus, the proposed algorithm is more suitable for actual cloud services than the existing algorithm.

Table 7 shows the cache hit ratio by cache capacity. The objective of this experiment was to find the optimal cache capacity, which is related to billing and insufficient resources in edge nodes. An edge node's limited computing resources mean that reducing the capacity of the cache is beneficial in terms of both the resource efficiency of the edge node and cost reduction for the user. We performed this experiment using the proposed algorithm for both CSPs and CSCs, and each cache hit ratio represents the average of 30 experiments. We repeated this experiment starting with 5 MB and decreasing by 0.5 MB. From 5 MB to 1.5 MB, it showed excellent performance, maintaining a cache hit ratio of 96% or more. However, below 1 MB, the cache hit ratio decreased gradually. This result could be related to the amount of data consumed by the cloud service in the experiment. In this experiment, from 128 KB to 1 MB of data were consumed. Therefore, to guarantee an adequate cache hit ratio for high quality service, 1.5 times the cache capacity above the maximum consumption per unit time could be needed. As a result, EDCrammer, a lightweight and agile algorithm, can reduce cache capacity. It could help CSPs to efficiently manage the storage resources of edge nodes, and it could help CSCs reduce their service fees.

Table 7. Cache hit ratio by cache capacity.

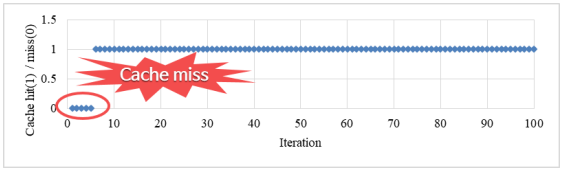
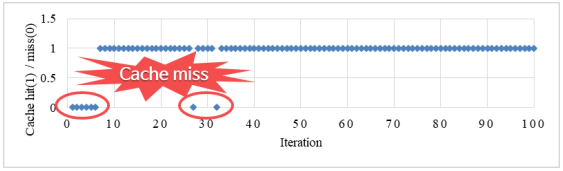
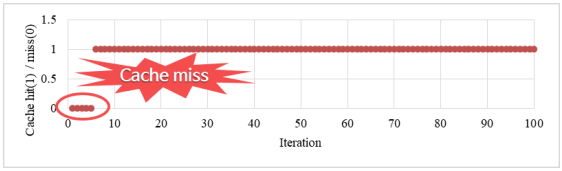
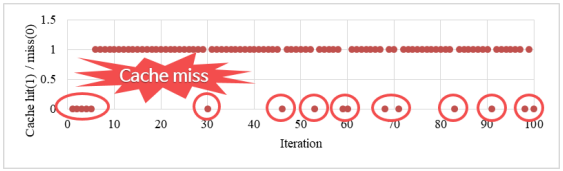
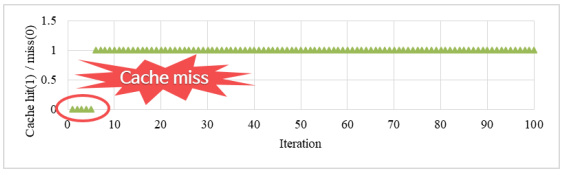
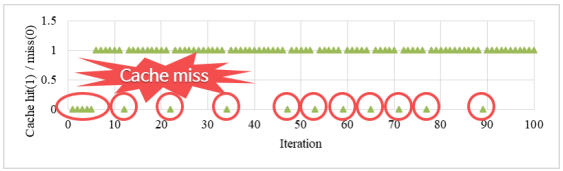
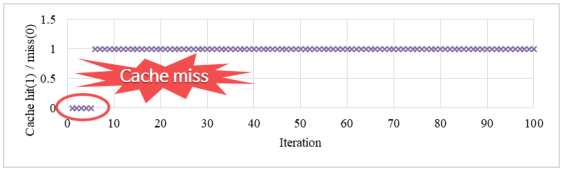
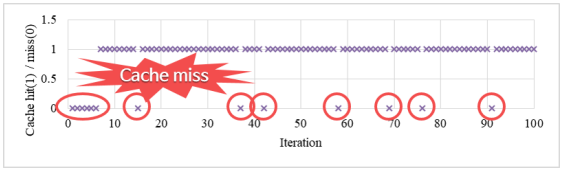
The Average Values of the Cache Hit Ratio					A Graph of the Cache Hit Ratio
MB	**Scenario**				
	1	**2**	**3**	**4**	
5	96.73	96.76	96.77	96.74	
4.5	96.74	96.74	96.72	96.71	
4	96.74	96.76	96.74	96.76	
3.5	96.71	96.76	96.74	96.70	
3	96.88	96.74	96.65	96.64	
2.5	97.42	97.40	97.40	97.38	
2	97.89	97.70	97.78	97.69	
1.5	98.31	97.00	96.66	96.65	
1	92.93	90.35	89.24	89.08	
0.5	9.51	46.01	50.58	53.75	



(a) Average cache hit ratios.

Table 8 shows the cache hit patterns marked at 5 MB and 1 MB. This experiment was performed to analyze the pattern of cache hits and determine the reason for the cache miss ratio of around 4%. The analysis in Table 7 indicates that 5 MB of cache capacity is sufficient, and 1 MB of cache capacity is insufficient. Both patterns confirmed that a certain period of waiting or preparation time is necessary. At the beginning of all graphs, a cache miss occurred within 10 iterations or within 1.5 s. This issue can thus be solved using methods such as buffering or delayed start, which are already applied to typical video streaming.

Table 8. Cache hit patterns.

Cache Hit Pattern with Sufficient Cache Capacity	Cache Hit Pattern with Insufficient Cache Capacity
	
(a) Scenario 1	**(b)** Scenario 1
	
(c) Scenario 2	**(d)** Scenario 2
	
(e) Scenario 3	**(f)** Scenario 3
	
(g) Scenario 4	**(h)** Scenario 4

4.4. Discussion

In performance experiments, we had a problem when assuming a congested network between the cloud and the edge. Due to the congested network, the edge node was unable to smoothly convey the current cache utilization to EDCrammer, and the EDCrammer also had difficulty in data caching in the edge node. To this end, we conducted further studies, confirmed that the probability of cloud-edge congestion is low in the existing network, and excluded this scenario from the experiment. However, this problem could occur in densely crowded environment as the traffic is concentrated to an access point (AP) of Fog, MEC, etc. The authors of [50,51] solved this issue by network slicing. Therefore, it is necessary to consider network slicing technique for data caching in a densely crowded environment.

In this paper, we consider an environment that the cloud node transmits data (data caching) to the edge node with consideration of user's data consumption. As a characteristic of the edge node that the computing resources are limited, an efficient caching rate-control algorithm, EDCrammer, that can store the data quickly in the edge node without requiring a lot of computing power has been proposed, and showed its necessity. However, since the proposed algorithm mainly considers mitigation of latency and load congestion, further studies on trust, security, privacy, forensic, etc. [52–55] are needed, such as intrusion awareness [56–59] and secure data aggregation [60]. We expect that applying those would provide more secure data caching. Meanwhile, since AI-based data caching [61–63] and AI-based edge management [59,64] are computationally intensive, this paper assumes that it is not appropriate for edge nodes. However, it can be effective for data caching when multiple factors are considered. Therefore, a lightweight AI or ML method for edge nodes is necessary. We expect that applying those would provide more precise and efficient data caching.

5. Conclusions and Future Works

Streaming data applications are increasing rapidly every year due to the advent of one-person media (e.g., personal broadcasting) and the emergence of 5G. In addition, user devices are changing from fixed types such as TV or PC to moving types (mobile) such as smartphones and tablet PCs, and most of the users would prefer smartphones. Many unspecified persons can freely move and watch high-quality media through various devices. Therefore, service providers have begun to experience difficulties in providing services to users. Most services are provided from a cloud datacenter. However, existing cloud computing has inevitable latency caused by its geographical location and the huge traffic concentrated in the central cloud. To solve those issues, edge computing is being actively researched. Of course, it would be nice to place many edge nodes near every user, but cost-effectively placing edge nodes is difficult. Therefore, in this paper, we assume that edge nodes have resource constraints. Lightweight, agile caching with a PID controller is proposed as an efficient rate-control algorithm for streaming high-quality data. Through the algorithm, the constrained computing resources of the edge nodes can be used efficiently. Existing caching techniques predominantly predict user patterns or data flows and deploy data in advance to provide data when needed by users. In addition, the pay-as-you-go pricing policy has been less considered, but it is a very important policy that must be considered when adopting cloud technology to edge nodes. For example, if a user uses a sufficient cache capacity, the cache hit ratio will increase, but the transaction will be costly. Therefore, the proposed EDCrammer is needed. The lightweight, agile rate-control algorithm EDCrammer performs minimal operations to prevent overloading the edge nodes and it satisfies the desired cache utilization quickly and reliably, even with those minimal operations. Improving existing algorithms to perform more sensitive caching is also an advantage of EDCrammer. In conclusion, EDCrammer is suitable for dynamic data streaming environments, which we demonstrated through the results of Scenario 4, in which the average convergence time of the proposed algorithm was 39% faster than that of the existing one, and the average convergence deviation of the proposed algorithm was also 4% more stable than the existing one. In addition, EDCrammer supports high cache hit ratios of more 96%, and the 4% miss can be improved through waiting time or preparation time. Finally, EDCrammer can help distribute the streaming data traffic to the edge nodes of 5G environment, mitigate the uplink load on the central cloud, and ultimately provide users with high-quality video services. We hope this algorithm can contribute to 5G environment, AR, VR, ITS, IoT, etc.

This study has a few limitations. In this study, we calculated K_p, K_i, and K_d using an engineering approach. When providing services in the cloud, there will be more variables than in our experiments, thus it would be better to use automatic gain tuning such as Ziegler–Nichols or Tyreus–Luyben. In addition, the possibility of this framework and the proposed algorithm was confirmed by comparing with the PID basic algorithm and the existing algorithm. However, it was still difficult to make a decision about optimal design. Therefore, we will compare more various algorithms through future studies. In addition, EDCrammer needs some different functionalities to provide multi-tenancy. For example, two CSCs might watch the same video. In that case, a CSP does not need to provide two different caches. Therefore, a method to save total cache capacity and guarantee QoS is necessary. Those limitations will be considered in our future work.

Author Contributions: Conceptualization, E.-N.H.; Software, Y.K.; Supervision, E.-N.H.; Validation, Y.K.; Visualization, Y.K.; Writing—original draft, Y.K.; Writing—review and editing, E.-N.H.

Funding: This research received no external funding.

Acknowledgments: This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00294, Service mobility support distributed cloud technology).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AR	Augmented Reality
C-RAN	Cloud Radio Access Networks
CSC	Cloud Service Customer
CSP	Cloud Service Provider
IoT	Internet of Things
ITS	Intelligent Transportation System
MDS	Maximum-Distance Separable
MQTT	Message Queuing Telemetry Transport
OS	Operating System
PID	Proportional-Integral-Differential
QoS	Quality of Service
SDC	Software-Defined Cache
VR	Virtual Reality

References

1. Kovacs, B. Distributed Cloud—A key Enabler of Automotive and Industry 4.0 Use Cases Ericsson Review (English Edition). 2018. Available online: https://www.ericsson.com/en/ericsson-technology-review/archive/2018/distributed-cloud (accessed on 23 June 2019).
2. Varghese, B.; Wang, N.; Barbhuiya, S.; Kilpatrick, P.; Nikolopoulos, D.S. Challenges and opportunities in edge computing. In Proceedings of the 2016 IEEE International Conference on Smart Cloud (SmartCloud), New York, NY, USA, 18–20 November 2016; pp. 20–26.
3. Wang, S.; Zhang, X.; Zhang, Y.; Wang, L.; Yang, J.; Wang, W. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access* **2017**, *5*, 6757–6779. [CrossRef]
4. Panwar, N.; Sharma, S.; Singh, A.K. A survey on 5G: The next generation of mobile communication. *Phys. Commun.* **2016**, *18*, 64–84. [CrossRef]
5. Klas, G.I. Fog computing and mobile edge cloud gain momentum open fog consortium, etsi mec and cloudlets. *Y.I Readings*, 2015. Available online: https://yucianga.info/?p=938 (accessed on 23 June 2019).
6. Tang, B.; Chen, Z.; Hefferman, G.; Wei, T.; He, H.; Yang, Q. A hierarchical distributed fog computing architecture for big data analysis in smart cities. In Proceedings of the ASE BigData & SocialInformatics 2015, Kaohsiung, Taiwan, 7–9 October 2015; p. 28.
7. NOKIA. The Edge Cloud: An Agile Foundation to Support Advanced New Services; Whitepaper; LightReading. 2018. Available online: https://www.lightreading.com/lg_redirect.asp?piddl_lgid_docid=750800&_mc=RSS_LR_EDT (accessed on 23 June 2019).
8. Dolui, K.; Datta, S.K. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–6.
9. Shahzadi, S.; Iqbal, M.; Dagiuklas, T.; Qayyum, Z.U. Multi-access edge computing: Open issues, challenges and future perspectives. *J. Cloud Comput.* **2017**, *6*, 30. [CrossRef]
10. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
11. Yi, S.; Hao, Z.; Qin, Z.; Li, Q. Fog computing: Platform and applications. In Proceedings of the 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Washington, DC, USA, 12–13 November 2015; pp. 73–78.
12. Luan, T.H.; Gao, L.; Li, Z.; Xiang, Y.; Wei, G.; Sun, L. Fog computing: Focusing on mobile users at the edge. *arXiv* **2015**, arXiv:1502.01815.
13. Hu, Y.C.; Patel, M.; Sabella, D.; Sprecher, N.; Young, V. Mobile edge computing—A key technology towards 5G. *ETSI White Pap.* **2015**, *11*, 1–16.
14. Satyanarayanan, M.; Bahl, P.; Caceres, R.; Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Comput.* **2009**, *8*, 14–23. [CrossRef]

15. Tsai, C.; Moh, M. Cache Management for 5G Cloud Radio Access Networks. In Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication, Langkawi, Malaysia, 5–7 January 2018; p. 83.
16. Zhu, K.; Zhi, W.; Chen, X.; Zhang, L. Socially Motivated Data Caching in Ultra-Dense Small Cell Networks. *IEEE Netw.* **2017**, *31*, 42–48. [CrossRef]
17. Liu, D.; Chen, B.; Yang, C.; Molisch, A.F. Caching at the wireless edge: Design aspects, challenges, and future directions. *IEEE Commun. Mag.* **2016**, *54*, 22–28. [CrossRef]
18. Tandon, R.; Simeone, O. Cloud-aided wireless networks with edge caching: Fundamental latency trade-offs in fog radio access networks. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, 10–15 July 2016, pp. 2029–2033.
19. Bastug, E.; Bennis, M.; Debbah, M. Living on the edge: The role of proactive caching in 5G wireless networks. *IEEE Commun. Mag.* **2014**, *52*, 82–89. [CrossRef]
20. Wu, J.; Ping, L.; Ge, X.; Wang, Y.; Fu, J. Cloud storage as the infrastructure of cloud computing. In Proceedings of the 2010 International Conference on Intelligent Computing and Cognitive Informatics, Kuala Lumpur, Malaysia, 22–23 June 2010; pp. 380–383.
21. Qureshi, M.K. Pay-As-You-Go: Low-overhead hard-error correction for phase change memories. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, Porto Alegre, Brazil, 3–7 December 2011; pp. 318–328.
22. *Y.3500: Information Technology—Cloud Computing—Overview and Vocabulary*; Standard; International Telecommunication Union Telecommunication Standardization Sector: Geneva, Switzerland, 2014.
23. *Y.3502: Information Technology—Cloud Computing—Overview and Vocabulary*; Standard; International Telecommunication Union Telecommunication Standardization Sector: Geneva, Switzerland, 2014.
24. Srivastava, S.; Singh, S.P. A survey on latency reduction approaches for performance optimization in cloud computing. In Proceedings of the 2016 Second International Conference on Computational Intelligence & Communication Technology (CICT), Ghaziabad, India, 12–13 February 2016; pp. 111–115.
25. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808. [CrossRef]
26. Corcoran, P.; Datta, S.K. Mobile-edge computing and the Internet of Things for consumers: Extending cloud computing and services to the edge of the network. *IEEE Consum. Electron. Mag.* **2016**, *5*, 73–74. [CrossRef]
27. Pathak, A.; Wang, Y.A.; Huang, C.; Greenberg, A.; Hu, Y.C.; Kern, R.; Li, J.; Ross, K.W. Measuring and evaluating TCP splitting for cloud services. In *International Conference on Passive and Active Network Measurement*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 41–50.
28. Cai, Y.; Yu, F.R.; Bu, S. Dynamic operations of cloud radio access networks (C-RAN) for mobile cloud computing systems. *IEEE Trans. Veh. Technol.* **2016**, *65*, 1536–1548. [CrossRef]
29. Díaz, M.; Martín, C.; Rubio, B. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *J. Netw. Comput. Appl.* **2016**, *67*, 99–117. [CrossRef]
30. Wu, D.; Liu, X.; Hebert, S.; Gentzsch, W.; Terpenny, J. Democratizing digital design and manufacturing using high performance cloud computing: Performance evaluation and benchmarking. *J. Manuf. Syst.* **2017**, *43*, 316–326. [CrossRef]
31. Borcoci, E.; Obreja, S. Edge Computing Architectures—A Survey on Convergence of Solutions. In Proceedings of the FUTURE COMPUTING 2018: The Tenth International Conference on Future Computational Technologies and Applications (IARIA), Barcelona, Spain, 18–22 February 2018; p. 8.
32. Borcoci, E. Fog Computing, Mobile Edge Computing, Cloudlets-which one. In Proceedings of the SoftNet Conference, Rome, Italy, 21–25 August 2016.
33. Dastjerdi, A.V.; Gupta, H.; Calheiros, R.N.; Ghosh, S.K.; Buyya, R. *Fog Computing: Principles, Architectures, and Applications*; Internet of Things, Elsevier: Amsterdam, The Netherlands, 2016; pp. 61–75.
34. Garcia Lopez, P.; Montresor, A.; Epema, D.; Datta, A.; Higashino, T.; Iamnitchi, A.; Barcellos, M.; Felber, P.; Riviere, E. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 37–42. [CrossRef]
35. Bonomi, F.; Milito, R.; Natarajan, P.; Zhu, J. Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*; Springer: Cham, Switzerland, 2014; pp. 169–186.

36. Beck, M.T.; Maier, M. Mobile edge computing: Challenges for future virtual network embedding algorithms. In *The Eighth International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2014)*; IARIA XPS Press: France, 2014; Volume 1, p. 3.
37. nebbiolotechnologies. Fog vs Edge Computing. Available online: https://www.nebbiolo.tech/wp-content/uploads/whitepaper-fog-vs-edge.pdf (accessed on 23 June 2019).
38. Khethavath, P.; Thomas, J.P.; Chan-Tin, E. Towards an efficient distributed cloud computing architecture. *Peer-to-Peer Netw. Appl.* **2017**, *10*, 1152–1168. [CrossRef]
39. 5G PPP. View on 5G Architecture (Version 3.0). 5G Architecture WG. 2019. Available online: https://5g-ppp.eu/wp-content/uploads/2019/06/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf (accessed on 23 June 2019).
40. ERICSSON. Mobility Report 2018. Available online: https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf (accessed on 23 June 2019).
41. Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022. 2019. Available online: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.pdf(accessed on 23 June 2019).
42. Ercan, T. Effective use of cloud computing in educational institutions. *Procedia-Soc. Behav. Sci.* **2010**, *2*, 938–942. [CrossRef]
43. Kondo, D.; Javadi, B.; Malecot, P.; Cappello, F.; Anderson, D.P. Cost-Benefit Analysis of Cloud Computing versus Desktop Grids. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2009, Rome, Italy, 23–29 May 2009.
44. Kostic, N. CapEx Vs OpEx: Cloud Computing. Available online: https://meritsolutions.com/capex-vs-opex-cloud-computing-blog/ (accessed on 23 June 2019).
45. Zhao, H.; Ye, J.; Watanabe, T. A Low-power Shared Cache Design with Modified PID Controller for Efficient Multicore Embedded Systems. *J. Inform. Process.* **2019**, *27*, 149–158. [CrossRef]
46. Al Ridhawi, I.; Aloqaily, M.; Kotb, Y.; Al Ridhawi, Y.; Jararweh, Y. A collaborative mobile edge computing and user solution for service composition in 5G systems. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3446. [CrossRef]
47. Al Ridhawi, I.; Mostafa, N.; Kotb, Y.; Aloqaily, M.; Abualhaol, I. Data caching and selection in 5G networks using F2F communication. In Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Canada, 8–13 October 2017; pp. 1–6.
48. Gabry, F.; Bioglio, V.; Land, I. On energy-efficient edge caching in heterogeneous networks. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3288–3298. [CrossRef]
49. Noh, A.S.I.; Huh, E.N.; Sung, J.Y.; Lee, P.W. An optimal and dynamic monitoring interval for grid resource information system. In Proceedings of the International Conference on Computational Science and Its Applications, Singapore, 9–12 May 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1144–1153.
50. Aloqaily, M.; Balasubramanian, V.; Zaman, F.; Al Ridhawi, I.; Jararweh, Y. Congestion Mitigation in Densely Crowded Environments for Augmenting QoS in Vehicular Clouds. In Proceedings of the 8th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications, Montreal, QC, Canada, 28 October–2 November 2018; pp. 49–56.
51. Aloqaily, M.; Al Ridhawi, I.; Salameh, H.B.; Jararweh, Y. Data and service management in densely crowded environments: Challenges, opportunities, and recent developments. *IEEE Commun. Mag.* **2019**, *57*, 81–87. [CrossRef]
52. Mukherjee, M.; Matam, R.; Shu, L.; Maglaras, L.; Ferrag, M.A.; Choudhury, N.; Kumar, V. Security and privacy in fog computing: Challenges. *IEEE Access* **2017**, *5*, 19293–19304. [CrossRef]
53. Huang, C.; Lu, R.; Choo, K.K.R. Vehicular fog computing: Architecture, use case, and security and forensic challenges. *IEEE Commun. Mag.* **2017**, *55*, 105–111. [CrossRef]
54. Esposito, C.; Castiglione, A.; Pop, F.; Choo, K.K.R. Challenges of connecting edge and cloud computing: A security and forensic perspective. *IEEE Cloud Comput.* **2017**, *4*, 13–17. [CrossRef]
55. Roman, R.; Lopez, J.; Mambo, M. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Gener. Comput. Syst.* **2018**, *78*, 680–698. [CrossRef]

56. Otoum, S.; Kantarci, B.; Mouftah, H. Adaptively supervised and intrusion-aware data aggregation for wireless sensor clusters in critical infrastructures. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
57. Otoum, S.; Kantarci, B.; Mouftah, H.T. Detection of known and unknown intrusive sensor behavior in critical applications. *IEEE Sens. Lett.* **2017**, *1*, 1–4. [CrossRef]
58. Otoum, S.; Kantarci, B.; Mouftah, H.T. Mitigating False Negative intruder decisions in WSN-based Smart Grid monitoring. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 153–158.
59. Otoum, S.; Kantarci, B.; Mouftah, H.T. On the Feasibility of Deep Learning in Sensor Network Intrusion Detection. *IEEE Netw. Lett.* **2019**, *1*, 68–71. [CrossRef]
60. Wang, H.; Wang, Z.; Domingo-Ferrer, J. Anonymous and secure aggregation scheme in fog-based public cloud computing. *Future Gener. Comput. Syst.* **2018**, *78*, 712–719. [CrossRef]
61. Nassar, A.T.; Yilmaz, Y. Reinforcement-Learning-Based Resource Allocation in Fog Radio Access Networks for Various IoT Environments. *arXiv* **2018**, arXiv:1806.04582.
62. Li, S.; Xu, J.; Van Der Schaar, M.; Li, W. Popularity-driven content caching. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–15 April 2016; pp. 1–9.
63. Manning, C.; Raghavan, P.; Schütze, H. Introduction to information retrieval. *Nat. Lang. Eng.* **2010**, *16*, 100–103.
64. Huang, L.; Bi, S.; Zhang, Y.J.A. Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks. *arXiv* **2018**, arXiv:1808.01977.