Article

A Multilayer Data Processing and Aggregating Fog-Based Framework for Latency-Sensitive IoT Services

Eman-Yaser Daraghmi ¹ , Meng-Chian Wu ² and Shyan-Ming Yuan ^{2,*} ¹ Applied Computing Department, College of Applied Sciences, Palestine Technical University Kadoori (PTUK), Tulkarm P.O. Box 7, Palestine; e.daraghmi@ptuk.edu.ps² Computer Science Department, National Chiao Tung University (NCTU), Hsinchu 300, Taiwan; purpletom07@gmail.com

* Correspondence: smyuan@cs.nctu.edu.tw; Tel.: +886-3-5712121 (ext. 56631)

Abstract: This study proposes a Client-Fog-Cloud (CFC) multilayer data processing and aggregation framework that is designed to promote latency-sensitive applications in an IoT context. The framework is designed to address the current IoT-based challenges: wide distribution, massive uploading, low latency, and real-time interaction. The proposed framework consists of the device gateway, the fog server and the cloud. The device gateway collects data from clients and uploads it to the nearest fog node. Received data will be pre-processed and filtered by the fog server before being transferred to the cloud for further processing or storage. An abduction alert fog-based service was implemented to evaluate the proposed framework. Performance was evaluated by comparing the response time and the delay time of the proposed architecture with the traditional cloud computing architecture. Additionally, the aggregation rate was evaluated by simulating the speed of bike riding as well as the walking speed of young adults and elderly. Results show that comparing with the traditional cloud, our proposal noticeably reduces the average response time and the delay time (i.e., whether the newest data or the historical data are being queried). Results indicate the capability of the proposed framework to reduce the response time by 32% and the data transferred to the cloud by 30%.

Citation: Daraghmi, E.-Y.; Wu, M.-C.; Yuan, S.-M. A Multilayer Data Processing and Aggregating Fog-Based Framework for Latency-Sensitive IoT Services. *Appl. Sci.* **2021**, *11*, 1374. <https://doi.org/10.3390/app11041374>

Academic Editor: Eui-Nam Huh

Received: 4 December 2020

Accepted: 1 February 2021

Published: 3 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: fog computing; Internet of Things; latency sensitive; aggregation

1. Introduction

Fog computing is as a promising technology that has been emerged to handle the growth of smart devices as well as the popularity of latency-sensitive and location-awareness Internet of Things (IoT) services [1–7]. After the emergence of IoT-based services, the industry of internet-based devices has grown. The number of these devices has raised from millions to billions, and it is expected to increase further in the near future [8]. Thus, additional challenges will be added to the traditional centralized cloud-based architecture as it will not be able to handle that growth and to support all connected devices in real-time without affecting the user experience.

IoT data that needs analysis, storage and processing are transferred to the data centers of the cloud. Because of the cloud centralized architecture, sending such data may not be efficient, or even not feasible in some cases due to the bandwidth constraints [9]. Moreover, the emergence of latency-sensitive and location-aware IoT services issues additional challenges as the distant cloud is not suitable to meet the ultra-low latency requirements of these services, support location-aware applications, or scale to the magnitude of the data that these applications produce [9]. To address these challenges, fog computing extends the paradigm of the cloud by utilizing the capabilities of edge devices and users' clients to compute, store, and exchange data among each other and with the cloud [10]. It shifts parts of the computations, communications, and resources' caching from the remote cloud to the network edge. Fog computing enables an innovative mixture of IoT-based services and applications for end-users as it effectively reduces both the service delay and the traffic load

by empowering end User Equipment (UE) with multitier computing or service. Therefore, data can be processed, and services can be provided flexibly at different tiers that are closer to UEs.

The fog server possesses computational power and storage to provide real-time delivery of data and to support latency-sensitive applications while maintaining satisfactory quality services. In contrast to the cloud architecture, fog computing deploys small routers, servers, switches, access points or gateways that require less space to occupy. Thus, fog nodes can be located and distributed close to IoT clients allowing latency to be reduced and location awareness services to be supported. Fog nodes are deployed in a decentralized manner, widely spread and distributed, as well as geographically available in large numbers. Additionally, data filtering and connection aggregation are supported by fog computing. Compared to the cloud, the cost of the fog server is usually lower than that of the cloud as high availability of computing resources at relatively high-power consumption will be provided by the cloud, whereas moderate availability of computing resources at lower power consumption will be provided by the fog computing.

Research in the field of fog computing is relatively new but is increasing very quickly [11–22]. Researchers have to consider the challenges arising from the huge number of IoT devices and their interactions, different technologies and distinct applications. Therefore, frameworks and models that simulate the actual fog system have to be developed and designed to overcome these challenges. This work proposes a Client-Fog-Cloud (CFC) multilayer data processing and aggregation framework to promote real-time interaction, wide distribution, massive uploading, and latency-sensitive services in an IoT context. The proposed framework consists of the device gateway, the fog server and the cloud. The device gateway collects data from clients and uploads it to the nearest fog node. Received data will be pre-processed and filtered by the fog server before being transferred to the cloud. Accordingly, the fog server can handle urgent requests (i.e., requests that need an emergent response, such as delay sensitive and emergency services) and meet the requirements of latency-sensitive services. Urgent requests will be pre-processed by the fog server and returned to the clients directly, while other requests will be transferred to the cloud for further processing and storage. A child abduction alert fog-based service is implemented to evaluate the proposed framework. Users of the service have to configure and set the threshold of the fog server first. The locations of parents and children will be uploaded to the fog server continuously. The fog server computes the distance between children and their parents to check whether it becomes over the configured threshold or not. An alert with the current location of the child will be sent to the parent by the fog server whenever the computed distance is over the user-defined threshold. Data from multiple clients will be aggregated and processed at the nearest fog node before being transferred to the cloud. Data aggregation will be performed on IoT clients that are located in the same geographic region before being transferred to the cloud, which is potentially located on a distant geographic region.

We evaluate the performance and the feasibility of the proposed framework. The response time and the delay time of the proposed CFC architecture and the traditional cloud were compared. Additionally, the aggregation rate was evaluated by simulating the speed of bike riding as well as the walking speed of young adults and elderly. Results show that comparing with the traditional cloud, our proposal noticeably reduces the average response time and the delay time (i.e., whether the newest data or the historical data being queried). Results indicate the capability of the proposed framework to reduce the response time by 32% and the data transferred to the cloud by 30%.

This paper is organized in five main sections. Section 2 discusses the related work. Section 3 illustrates the design and the implementation of the proposed framework. The detailed functions provided by the proposed fog server is presented in Section 4. Section 5 demonstrates the conducted experiments and their results. Finally, Section 6 concludes this paper and identifies the future work.

2. Related Work

Research in this field of fog computing is relatively new but is increasing very quickly [22]. Subhadeep Sarkar et al. [18] evaluate the capability of the traditional cloud computing integrated with the fog computing to support the increasing demands of the latency-sensitive IoT applications. In their work, a comparative analysis of both paradigms—fog computing and conventional cloud—in the context of IoT was performed. They illustrate the various network devices and networking links within the fog computing paradigm and demonstrate the traffic exchange pattern. Additionally, the fog computing performance metrics in terms of the power consumption, service latency, the corresponding costs incurred, and CO₂ emission for different renewable and non-renewable energy resources was mathematically characterized. Results of performance evaluation indicate the enhanced performance of fog computing both in terms of the provisioned quality of service (QoS) and eco-friendliness. They justify the fog paradigm as an improved, eco-friendly computing platform that can support IoT better compared to the existing cloud computing paradigm.

In 2016, Ping Guo et al. [11] presented an optimal deployment and dimensioning (ODD) study of fog computing to support a vehicular network. Authors in the paper formulate the problem as an integer linear programming (ILP) in order to minimize the cost of deployment. Two different models based on coupling and decoupling fog devices with Roadside Units (RSUs) are developed.

M. Aazam et al. [16] present a resource allocation prediction and pricing model. They introduced a service-oriented resource management model for IoT devices, through fog computing in order to provide efficient, effective, and fair management of resources. Their model demonstrates the issues of resource prediction, customer type-based resource estimation and reservation, advance reservation, and pricing for new and existing IoT customers, on the basis of their characteristics. Java and CloudSim toolkit were used for implementation. Results indicate the validity and performance of their system. Yung-Chiao Chen et al. [17] propose a fog–cloud architecture for information-centric IoT applications with job classification and resource scheduling functions. Their proposal supports the quality of service (QoS) by classifying IoT applications and scheduling computing resources to optimize the dispatch of cloud and fog resources at minimum cost. In their work, one cloud, three fogs, four types of IoT application and 2500 IoT tasks are designed in the cloud–fog simulation environment. Results indicate that the proposed architecture outperforms the computing without scheduling by decreasing the load of the cloud computing by approximately 20.47%.

Bo Tang et al. [12] present a distributed hierarchical fog computing paradigm for analyzing big data in smart cities. In this work, intelligence at the edge of a layered fog computing network was distributed as the geo-distribution characteristic in big data generated by massive sensors. Latency-sensitive applications will be performed by the computing nodes of each layer. The computing nodes also support quick control loop to guarantee the safety of infrastructure components. As a use case, the authors implemented a smart pipeline monitoring. They implemented a prototypical four-layer fog-based computing paradigm to illustrate the system's effectiveness and the feasibility. Jianhua He et al. [2] propose a multi-tier fog computing framework for smart city applications. Large scale analytics services can be run over their system with the support of opportunistic and dedicated fog and they call them ad-hoc fog and dedicated fog. They also develop a discrete-event driven system-level simulator to evaluate the performance. Findings demonstrate the efficiency of analytics services over multi-tier fogs and the effectiveness of the proposed schemes. Fogs can largely improve the performance of smart city analytics services compared to the cloud model in terms of job blocking probability and service utility. Xavi Masip-Bruin et al. [14] present a Fog To Cloud (F2C) computing system. In this work, the authors introduce a layered F2C architecture, its benefits and strengths, as well as the arising research challenges, making the case for the real need for their coordinated management. Mehdi Sookhak et al. [15] present Fog Vehicular Computing (FVC) to augment the computation and storage power of

fog computing. Opeyemi Osanaiye et al. [13] present a detailed review of fog computing, its architecture and applications. They present the security, privacy and resource availability challenges and propose a novel smart pre-copy Virtual Machine (VM) live migration conceptual framework to cater for malicious attacks or failure of physical servers which result in unavailability of services and resources.

3. Design and Implementation

3.1. Overview

The architecture of the proposed “Client-Fog-Cloud” (CFC) framework is illustrated in this section. As shown in Figure 1, CFC consists of three main parts: (1) The device gateway; (2) the fog server; (3) the Cloud. The device gateway receives IoT data from clients that are not able to connect with the fog directly and uploads it to the nearest fog node. The fog server handles IoT data that will either be received from clients directly or device gateways. The fog server can be viewed as a middle layer or intermediate layer that links clients with the cloud. The fog nodes pre-process and filter the uploaded data before being transferred to the cloud. Accordingly, the fog server can handle urgent requests and meet the requirements of latency-sensitive services. The cloud receives all data for further processing or storage.

Figure 1. Conceptual architecture of the proposed system.

The general architecture of the fog server is shown in Figure 2. The fog server consists of six main modules: accessing module, device module, storage module, security module, processing module, and logic controller (i.e., message broker).

Figure 2. General architecture of the fog server.

The main functions of the proposed fog server are detailed in the following points.

1. **Uploading data:** data can be uploaded to the fog server and stored in the database. Data synchronization is enabled between the fog server and the cloud. Up-to-date data (i.e., newest data) will be stored in the fog server, while historical data will be stored in the cloud.
2. **Data query:** clients can query the newest data from the fog server with a promised response time. Clients also can query the historical data from the cloud as data in the fog server will be synchronized with the cloud.
3. **Data Processing:** the fog server supports data processing.
4. **Filtering data:** with some configurations, data in the fog server will be synchronized with the cloud. For example, assume that the proposed CFC framework is utilized to provide a child abduction alert service (i.e., latency-sensitive and location awareness service). Users of the service have to configure and set the threshold of the fog server. The locations of parents and children will be uploaded to the fog server continuously. The fog server computes the distance between children and their parents to check whether there is a distance change or not (i.e., if distance increases over the configured threshold). An alert with the current location of the child will be sent to the parent by the fog server whenever the computed distance is over a user-defined threshold. Data will be synchronized to the cloud periodically; they will be filtered first and synchronized when the change is out of threshold to reduce the amount of data being transferred to the cloud and thus reducing the cloud overhead.
5. **Connection Aggregation:** the fog server plays a key role in minimizing the number of concurrent cloud connections. Data from multiple clients will be aggregated and processed at the nearest fog node before being transferred to the cloud. The fog server aggregates data received from multiple clients' nodes, usually placed on the same geographic region, in order to aggregate all data for a given geographic region before sending data to the cloud layer, which are potentially placed in a distant geographic region. Assume that there are 2000 devices connecting to the cloud directly. This situation requires 2000 concurrent cloud connections to be established. In contrast, with the proposed framework, connection aggregation is supported. The data of these 2000 devices will be aggregated and transferred to the cloud in only one connection.
6. **Authentication:** the challenge–response protocol is employed to authenticate users by the fog server.

In practice, logical and physical redundancy is required when implementing fog computing-based systems. Thus, eliminating the single point of failure risk which affects the availability and the reliability of the entire system. Moreover, when data are transmitted through a single point, a bottleneck problem could occur. High-availability and redundancy clusters are the key factors to avoid the single point of failure risk. The set of fog servers is modeled as an undirected graph $G = (N, E)$ where N represents the set of fog servers and E describes the connection-links among them. The fog servers are categorized into two types:

(1) an in-domain fog server, and (2) a master fog server. The in-domain fog server represents a server which is located in one cluster and only has connections via short-links with all in-domain servers placed in the same cluster and the master server of that cluster. The master server represents a server located in one cluster and has a connection via short-links with all in-domain servers placed in the same cluster and at the same time has connection via long-links with some master nodes located in other clusters (Figure 3). The long-links (i.e., red lines in the figure) create connections among master nodes and are responsible for achieving the high clustering coefficient in the network. Short-links (i.e., blue lines) create connections among in-domain nodes, and among master servers and in-domain servers. Clustering minimizes the outages of the system components. When a fog server fails, then immediately another server has to process the role of the failed one. The details of clusters' constructions are beyond the scope of this research (more details [23]).

Figure 3. Eliminating the single point of failure risk.

3.2. System Implementation

This section illustrates in detail the three main components of the proposed CFC framework.

3.2.1. Device Gateway

This component is responsible for collecting and gathering data from clients' devices, such as sensors or smart wearable devices, and then transferring them to the fog server in a static frequency. The device gateway receives IoT data from clients who are not able to make connections with the fog server directly. It is important to note that IP protocol

may not be supported by all clients' devices. IP-enabled devices can communicate with the fog directly allowing them to transfer data in an IoT context, whereas other devices that support other protocols, such as ZigBee, have to communicate with the fog through the device gateway. The device gateway plays a key role in exchanging data between different protocols. MPIGate [24] (i.e., it stands for a multi-protocol-interface gateway) that integrates actuators and sensors that are supporting different network protocols (e.g., ZigBee, EIB/KNX, Bluetooth, WiFi) is used as a device gateway.

In fog computing, nodes are placed close to clients' devices (IoT source nodes) allowing them to upload their data to the nearest fog node (i.e., latency will be noticeably reduced). Clients who upload their data via the device gateway have to share their location coordinate allowing it to upload the received IoT data to the nearest fog node.

3.2.2. Fog Server

The detailed architecture of the fog server is shown in Figure 4. The fog server consists of several modules that can be categorized as frontend and backend modules. Several programming languages can be used for implementation. In this research, we use java and node.js to for implementing the proposed framework. The frontend module, written in node.js, is the accessing module that handles the network connections. The backend modules are written in java to handle, process and store data. The logic controller or the message broker, namely Active MQ, connects all modules and allows them to exchange messages. The modules of the proposed fog server are illustrated in the following points. Figure 5 shows the internal communications between the server modules.

Figure 4. Detailed architecture of the fog server.

1. Accessing Module

Node.js is used to implement the accessing module in the proposed prototype. Because of the characteristic of event-driven and non-blocking Input/Output (I/O) model, node.js is considered as an effective way to handle the network connections. Four main sub-modules are included in the accessing module.

- **Network Request Handler**

It handles the network connections by receiving the requests and delivering them to the corresponding backend module according to the type of the received request. After processing data via the backend modules, the network request handler sends the response back to the clients.

- AuthTracker

Authentication is one of the main functions that is applied by the device module in the fog server. Authentication of the device module is considered as a strong authentication since the device module employs the complete authentication to generate a ticket that will be stored in the AuthTracker upon finishing the complete flow. To reduce the response time, there is no need for strong authentication whenever an authenticated device would like to connect to the fog server, instead the ticket stored in the AuthTracker can perform quick authentication. Data processing by the fog server will be performed after passing the authentication process. Note that a ticket will be invalid when there is no connection between the device and the fog server for a defined period of time.

- Data Sync Handler

It periodically synchronizes data. The synchronization time period is a user-defined parameter (i.e., default is 5 min).

- Query Tracker

It caches the newest data by sending requests to the database to retrieve all the newest data every second. The query tracker plays a key role in achieving a promising response time. Retrieved data will be maintained in the accessing module. Consequently, when the accessing module receives queries from clients, it can return the response immediately without passing the request to the database through the message broker. Therefore, the overhead will be reduced and the response time will be more stable. Assume that there are 1000 clients who send queries simultaneously for every second. In that case, without the query tracker, the fog server has to send 1000 requests to the database which in turn has to retrieve each response individually. On the other hand, by implementing the fog server with the query tracker, the fog server only sends a request to the database that in turn retrieves 1000 data collectively.

2. Device Module

This module is responsible for authentication and registration. The challenge–response protocol is applied for authentication. Clients generate a challenge that will be uploaded to the device module which in turn calculates the response accordingly with the help of the security module. This module includes two sub-modules as detailed in the following points.

- Device Handler

It handles incoming requests. It communicates with the storage module for registration requests to verify whether the database stores the registration information of that device. For authentication requests, the device handler, first, checks the registration information of the device and then communicates with the security module in order to calculate the corresponding response.

- Device Tracker

It caches the device information after registration in order to minimize the authentication overhead by reducing the database operations.

3. Processing Module

It performs data processing and filtering before relaying data to the storage module. The processing module consists of:

- Processing Handler: It receives the transferred data from the accessing module.
- Processing Unit: It processes data and returns response when needed.
- Data Tracker: It filters data. Data Tracker has an in-memory object to store the newest data of every device that is connected with the fog server. Data will be relayed to the storage module periodically by a user defined period.

4. Storage Module

It handles the database operations. Database manipulation can be performed only by the storage module. This module consists of:

- DBHandler: It receives requests from the other modules and performs the corresponding operation according to the request types.
- Database: PostgreSQL is used as the database of the proposed fog server. PostgreSQL is a powerful object-relational database.

5. Security Module

Generally, this module is responsible for the encryption and the verification algorithms. This module is not covered in this paper as it is beyond its scope.

6. Message broker

A module can connect with another module through the corresponding queue. Modules can fire the message and continue performing their functions without the need for waiting for the response. A module will be notified to receive messages (if any) in an event-driven manner which in turn increases the performance of the proposed fog server. Several JMS APIs can be employed for implementation, such as open JMS, rabbitMQ, and ActiveMQ. In this work, ActiveMQ is utilized for implementing the message broker.

Figure 5. Internal communications between the server modules.

3.2.3. Cloud

The architecture of the cloud is similar to the architecture of the fog server. However, there is one main difference between them regarding data handling. The fog server receives data from device gateways which upload data every second or even less to ensure that the uploaded data are current and up to date. Thus, the fog server can receive a large number of device gateways connections with small number of data per connection. On the other hand, the cloud receives data from the periodic synchronization of fog server. Fog server will synchronize the data every 5 min. Thus, the data received by the cloud will be tens of millions per connection.

4. Functions of the Fog Server in Details

This section illustrates the functions of the proposed fog server in details.

4.1. Authentication

This section illustrates the authentication process of the proposed fog server (see Figure 6). As mentioned before, strong authentication has to be performed when the client connects to the fog server for the first time. The authentication information of this client will be maintained in the AuthTracker to allow performing weak authentication later.

Figure 6. Authentication data flow.

1. A client sends a request to the fog server that will be authenticated first by the AuthTracker (quick authentication).
2. As there is no authentication information stored for that client yet, the fog server sends the “need_auth” back to client.
3. The authentication flow will be started in this step. The client sends the challenge that will be generated by the client to the fog server.
4. After receiving the challenge, the device module of fog server will check the registration of the device first.
5. If the registration information exists, the corresponding response will be calculated and sent back to the accessing module.
6. While the accessing module receives the response, the device module will generate a ticket to be stored in the AuthTracker simultaneously.
7. The response will be sent back to client.
8. After the client receives the response, it will send the same request to fog server again and this request should be authenticated by AuthTracker as well. The quick authentication will be passed as AuthTracker already has stored the authentication information of that client.

4.2. Data Uploading

Figure 7 shows how a client uploads data to the fog server.

1. A client sends “data upload” request to the fog server. To process the received request, the client has to pass the authentication process first (authentication steps are detailed in Section 4.1).
2. Data will be sent from the accessing module to the processing module.
3. The processing module will send the received data to the storage module to store.

4.3. Data Query

Figure 8 shows the sequence of data query.

1. A client sends a “data query” request to the fog server. To process the received request, the client has to pass the authentication process first (authentication steps are detailed in Section 4.1).
2. The query tracker in the accessing module caches the newest data and sends a request to the database to retrieve all the newest data every second.
3. Data are maintained in the accessing module. Consequently, when receiving a “data query” request, the response will be returned immediately without passing this through the message broker to the database.

4.4. Synchronization with Cloud

As shown in Figure 9, the fog server selects all the data that need to be synchronized and transfers them to the cloud periodically. The time period of synchronization is a user-defined parameter, and the default is 5 min.

Figure 7. Data uploading workflow.

Figure 8. Data query workflow.

Figure 9. Synchronization workflow.

5. Experiment and Evaluation

Three experiments were performed to evaluate the performance of the proposed CFC framework and to compare it with the traditional cloud architecture.

5.1. Environment Setting

Table 1 shows the specification of the host and the guest machines utilized to perform the experiments. It also shows the specifications of the cloud instance. The fog server was deployed as a guest VM on a host machine (see Table 1). In order to compare the performance of the fog server with the cloud, the services were applied on the AWS web service.

Table 1. The specifications of the cloud server, the host and the guest machines.

	Host Machine	Guest Machine	Amazon Web Service—EC2
OS	VMware	CentOS 7	CentOS 7
CPU	I7-6700 core 4	Core 2	Core 2
Memory	40 GB	4 GB	4 GB
HDD	2 TB	50 GB	30 GB
Location	-	-	Mumbai

5.2. Experiment Setup

To set up our experiment, the network limitation regarding the number of connections that a fog server can receive is evaluated. The findings of this experiment will be considered as the guideline for the next experiments. Clients are simulated by iFogSim [25] which is considered as one of the most popular fog computing simulation tools to simulate fog environments and IoT data. It was employed to create connections and send requests to the fog server. A response will be sent back immediately from the fog server to the clients after receiving the requests. The time between sending/receiving requests/responses is called the response time. Different numbers of clients are used to measure the response time. Figure 10 shows the average response time in each circumstance. The vertical axis represents time in milliseconds and the horizontal axis represents the round which indicates the average response time of 100 times. Results indicate that the response time remains stable with 400 to 2000 clients' connections and it starts to oscillate with 2400 clients' connections; as the iFogSim has some limitations regarding the number of network links and limited network usage, it supports only tree-like topologies. Therefore, 2000 clients were used to perform the next experiments.

Figure 10. Network limitation.

5.3. Experiment Design

5.3.1. First Experiment: Query Newest Data

The first experiment is designed to measure the response time of the proposed CFC framework and the traditional cloud architecture when query the newest data. iFogSim [25] is utilized for simulation. According to the results of the experimental setup, 2000 clients were employed to make connections with the fog server. To simulate reality, the clients were divided into two categories: we assumed that 1600 clients uploaded data in the speed of 760 Mbytes/s and 400 clients sent query requests (as in reality the number of clients uploading data is greater than the number of clients requesting data). Overall, 1600 clients employ 1600 gateways for collecting the IoT devices' data. In this experiment, we assume that there were 10,000 devices and every device sends its location x , y , and z , its identifier and a timestamp which form 76 bytes in total. Consequently, a gateway will collect device data from 10,000 devices making the speed of data upload data 760 Mbytes/s.

For evaluation, a child abduction alert service (i.e., latency-sensitive and location awareness service) was simulated. The threshold of the fog server is configured first. The locations of parents and children will be uploaded to the fog server continuously. The fog server computes the distance between children and their parents to check whether the distance increases over the configured threshold or not. An alert with the current location of the child will be sent to the parent by the fog server whenever the computed distance is over the defined threshold. Data will be synchronized to the cloud periodically; they will be filtered first and synchronized when the change is out of threshold to reduce the amount of data being transferred to the cloud and thus reducing cloud overhead.

We measured the response time of these 400 clients who sent query requests regarding getting the newest data. The response time means the time difference between sending the request and receiving the response. Note that when requesting the newest data, the fog server will return the response. In contrast, requests that query historical data need a connection with the cloud. In this experiment, the response time and the delay time of the proposed framework were compared with three fog computing-based frameworks [26–28] that were published in the GitHub website [29].

5.3.2. Second Experiment: Query Historical Data

The second experiment measures the response time of the proposed CFC framework and the traditional cloud architecture when querying historical data. Similar to the previous experiment, 1600 clients uploaded data in the speed of 760 Mbytes/s and 400 clients sent query requests.

5.3.3. Third Experiment: Aggregation Rate

The third experiment measures the aggregation rate of the fog server and evaluates its capability in filtering data. iFogSim [25] is utilized to simulate the wide and dense geographical distribution of clients. First, we simulate the walking speed of young adults and elderly. Then, we simulate the speed of bike riding to test the aggregation rate of the fog server. In total, 1000 clients uploaded data in the speed of 760 bytes/s for 10 min. The fog server synchronizes the database with the cloud. Data being uploaded to the cloud are calculated to find the aggregation rate.

5.4. Results

5.4.1. First Experiment: Query Newest Data

Figure 11 shows the average response time of query the newest data (i.e., the average response time of 400 query/round) for the proposed framework and the traditional cloud architecture. The vertical axis represents the time in milliseconds and the horizontal axis represents the round. Results show that the average response time for querying the newest data from the proposed framework is very low. It ranges from 20 ms to 80 ms. In contrast, the average response time ranges from 120 to 180 ms when using the traditional cloud

architecture. Results indicate higher response time for the traditional cloud architecture compared with the CFC framework.

Figure 11. The response time of query the newest data (the proposed architecture vs. traditional cloud).

Moreover, our proposed framework was compared with three fog computing-based frameworks [26–28] that were published in the GitHub website [29] Figure 12 shows the results of the comparison. Results show that the average response time when querying the newest data from the proposed framework is lower than the others. The proposed framework outperforms the other fog computing-based frameworks regarding querying the newest data.

Figure 12. The response time of query for the newest data.

Figures 13 and 14 show the average time delay of a query relating to the newest data (i.e., the average time delay of 400 query/round). The vertical axis represents time in seconds and the horizontal axis represents the round. The time delay indicates the time between sending/receiving the request/response. Thus, the time delay specifies whether the received data are the newest data or not. For example, 2 s time delay indicates that the data which the client had received were the data uploaded before 2 s. Results show that the time delay ranges from 1 to 4 s for the proposed framework.

Figure 13. The time delay of newest data query (the proposed architecture vs. traditional cloud).

Figure 14. The time delay of newest data query.

In cloud architecture, the time delay increases in an exponential manner and it ranges from 5 to 400 s. The reason behind this is the large size of data stored in the cloud; thus, the selection operation of the query tracker will be subjected to a heavy load. It is important to note that while performing data query, there are many other clients who are uploading their data simultaneously. This situation will affect the insertion speed as well. Results of the second experiment indicate that the query response will not include the newest data as time delay increases in an exponential manner when using the traditional cloud architecture.

5.4.2. Second Experiment: Query Historical Data

Figure 15 shows the average response time of a query relating to the historical data for the proposed framework and the traditional cloud architecture. As shown in the Figure, the average response time ranges between 2 and 4 s for the proposed framework, while it ranges from 8 to 12 s for the traditional cloud. Results show that the response time of a query using the historical data is greater than the average response time of a query using the newest data.

Figure 15. The response time of a query using the historical data (the proposed architecture vs. traditional cloud).

The main reason is the fog server stores the newest data in its storage module; thus, it is responsible for responding to the newest data queries, whereas the cloud stores historical data; thus, responding to historical data queries requires interaction with the cloud. Fog nodes are placed close to clients allowing response time to be noticeably minimized, while the cloud is placed in a distant geographic region. The query tracker of the fog server plays a key role in achieving a promising response time as it caches the newest data by sending requests to the database to retrieve all the newest data every second. Retrieved data will be maintained in the accessing module. Consequently, when the accessing module receives queries from clients, it can return the response immediately without passing the request to the database through the message broker. Therefore, the overhead will be reduced and the response time will be minimized. The device tracker caches the registration

information of devices in order to minimize the authentication overhead. Additionally, because of the message broker, all implemented modules can fire the message and continue performing their functions without the need for waiting for the response. A module will be notified to receive messages (if any) in an event-driven manner which in turn increases the performance.

As shown in the figure, the response time is very high when the traditional cloud architecture is employed. It is the highest in comparison with all previous results. In our proposed architecture, the fog layer plays a key role in reducing the cloud overhead. In the traditional cloud computing architecture, there is no fog server and the cloud is responsible for all functions provided by that server (i.e., in CFC framework, fog server sends only selected data to the cloud for analysis and long-term storage).

5.4.3. Third Experiment: Aggregation Rate

Figure 16 shows the aggregation rate of the proposed framework. It indicates the aggregation rate for the walking speed of young adults and for elderly people. It also shows the aggregation rate for bike riding.

According to previous research, the walking speed of young adults ranges from 1.0416 to 1.508 m/s [22], the walking speed of elderly is about 0.88 to 1.083 m/s [30], and the speed of riding a bike is about 4.166 to 6.944 m/s [31]. The vertical axis represents the aggregation rate in % and the horizontal axis represents the round. The total number of aggregated data is 6 million bytes for every round. The threshold of fog server was set to 0.4 m. This number indicates that the fog server continuously checks whether there is a distance change or not (i.e., if distance increases over the defined threshold “0.4 m”). Data will be synchronized only when the change is out of threshold to reduce the amount of data being transferred to the cloud. Results indicate that the proposed prototype is suitable for uploading and aggregating large amounts of data.

Figure 16. Aggregation rate.

5.4.4. Additional Results

Figure 17 shows the average response time of a query using both the newest and the historical data when simulating 2400 clients. The vertical axis represents the time in milliseconds and the horizontal axis represents the round. The average response time of newest data query ranges from 29 to 187 ms, while the average response time of historical data query ranges from 134 to 345 ms.

Figure 18 shows the average time delay of a query for the newest and the historical data (2400 clients were simulated). The vertical axis represents time in second and the horizontal axis represents the round. The time delay indicates the time between sending/receiving the request/response. The average delay time of newest data query ranges from 2 to 8 s, while the average delay time of historical data query ranges from 4 to 18 s.

Figure 17. The response time of newest and historical data query (2400 clients' connections).

Figure 18. The delay time of newest and historical data query (2400 clients' connections).

Results show that the average response time and delay time of the experiment when using 2400 clients oscillate and are not stable. Findings, as detailed in the setup experiment in Section 5.2, indicate that the results remain stable with 400 to 2000 clients' connections and start to oscillate with 2400 clients' connections. To handle the increment in the clients' connections, additional fog nodes and gateways should be set and configured to simulate greater numbers of clients.

6. Conclusions

This work proposes a multilayer data processing and aggregation fog-based framework. The proposed framework meets the requirements of IoT context: wide distribution, massive uploading, latency-sensitive, location awareness, and real-time interaction. For evaluation, extensive experiments were performed to evaluate the response time and the delay time of the proposed framework and its aggregation rate. Results show that comparing with the traditional cloud architecture, CFC framework reduces the average response time (i.e., whether the newest data or the historical data are being queried). Results show that the CFC framework has a better behavior than the conventional cloud architecture since fog nodes are placed closed to the clients allowing the response time to be noticeably minimized, while the cloud is placed in a distant region.

The proposed framework also minimizes the amount of data transferred to the cloud. Data from multiple clients will be aggregated and processed at the nearest fog node before being transferred to the cloud. Data aggregation will be performed on IoT clients that are located on the same geographic region before being transferred to the cloud, which is potentially located in a distant geographic region. Many tests, adaptations, and experiments have been left for future work (i.e., performing additional experiments with real data, implementing additional services for smart city applications). Future work concerns deeper analysis of the proposed framework, such as adding a load-balancing algorithm between fog nodes.

Author Contributions: Conceptualization, E.-Y.D.; methodology, M.-C.W.; formal analysis, M.-C.W.; writing—original draft preparation, M.-C.W.; writing—review and editing, E.-Y.D.; supervision, S.-M.Y.; and project administration, S.-M.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data supporting this study are available within the article.

Acknowledgments: We thank National Chaio Tung University (NCTU) and Palestine Technical University—Kadoori (PTUK) for their support.

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

References

1. Hao, W.; Sun, G.; Muta, O.; Zhang, J.; Yang, S. Coordinated Hybrid Precoding Design in Millimeter Wave Fog-RAN. *IEEE Syst. J.* **2020**, *14*, 673–676. [\[CrossRef\]](#)
2. He, J.; Wei, J.; Chen, K.; Tang, Z.; Zhou, Y.; Zhang, Y. Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities. *IEEE Internet Things J.* **2018**, *5*, 677–686. [\[CrossRef\]](#)
3. Kar, P.; Misra, S.; Obaidat, M.S. RILoD: Reduction of Information Loss in a WSN System in the Presence of Dumb Nodes. *IEEE Syst. J.* **2019**, *13*, 336–344. [\[CrossRef\]](#)
4. Kim, M.; Han, A.; Kim, T.; Lim, J. An Intelligent and Cost-Efficient Resource Consolidation Algorithm in Nanoscale Computing Environments. *Appl. Sci.* **2020**, *10*, 6494. [\[CrossRef\]](#)
5. Montoya-Munoz, A.I.; Rendon, O.M.C. An Approach Based on Fog Computing for Providing Reliability in IoT Data Collection: A Case Study in a Colombian Coffee Smart Farm. *Appl. Sci.* **2020**, *10*, 8904. [\[CrossRef\]](#)
6. Scarpiniti, M.; Baccarelli, E.; Momenzadeh, A. VirtFogSim: A Parallel Toolbox for Dynamic Energy-Delay Performance Testing and Optimization of 5G Mobile-Fog-Cloud Virtualized Platforms. *Appl. Sci.* **2019**, *9*, 1160. [\[CrossRef\]](#)

7. Scarpiniti, M.; Baccarelli, E.; Momenzadeh, A.; Uncini, A. SmartFog: Training the Fog for the Energy-Saving Analytics of Smart-Meter Data. *Appl. Sci.* **2019**, *9*, 4193. [\[CrossRef\]](#)
8. Nasir, M.; Muhammad, K.; Lloret, J.; Sangaiah, A.K.; Sajjad, M. Fog computing enabled cost-effective distributed summarization of surveillance videos for smart cities. *J. Parallel Distrib. Comput.* **2019**, *126*, 161–170. [\[CrossRef\]](#)
9. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [\[CrossRef\]](#)
10. Shirazi, S.N.; Gouglidis, A.; Farshad, A.; Hutchison, D. The Extended Cloud: Review and Analysis of Mobile Edge Computing and Fog From a Security and Resilience Perspective. *IEEE J. Select. Areas Commun.* **2017**, *35*, 2586–2595. [\[CrossRef\]](#)
11. Guo, P.; Lin, B.; Li, X.; He, R.; Li, S. Optimal Deployment and Dimensioning of Fog Computing Supported Vehicular Network. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016; pp. 2058–2062. [\[CrossRef\]](#)
12. Tang, B.; Chen, Z.; Hefferman, G.; Pei, S.; Wei, T.; He, H.; Yang, Q. Incorporating Intelligence in Fog Computing for Big Data Analysis in Smart Cities. *IEEE Trans. Ind. Inf.* **2017**, *13*, 2140–2150. [\[CrossRef\]](#)
13. Osanaiye, O.; Chen, S.; Yan, Z.; Lu, R.; Choo, K.-K.R.; Dlodlo, M. From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework. *IEEE Access* **2016**, *5*, 8284–8300. [\[CrossRef\]](#)
14. Masip-Bruin, X.; Marin-Tordera, E.; Tashakor, G.; Jukan, A.; Ren, G.-J. Foggy clouds and cloudy fogs: A real need for coordinated management of fog-to-cloud computing systems. *IEEE Wirel. Commun.* **2016**, *23*, 120–128. [\[CrossRef\]](#)
15. Sookhak, M.; Yu, F.R.; He, Y.; Talebian, H.; Safa, N.S.; Zhao, N.; Khan, M.K.; Kumar, N. Fog Vehicular Computing: Augmentation of Fog Computing Using Vehicular Cloud Computing. *IEEE Veh. Technol. Mag.* **2017**, *12*, 55–64. [\[CrossRef\]](#)
16. Aazam, M.; Huh, E.-N. Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT. In Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, Gwangju, Korea, 24–27 March 2015; pp. 687–694. [\[CrossRef\]](#)
17. Chen, Y.-C.; Chang, Y.-C.; Chen, C.-H.; Lin, Y.-S.; Chen, J.-L.; Chang, Y.-Y. Cloud-fog computing for information-centric Internet-of-Things applications. In Proceedings of the 2017 International Conference on Applied System Innovation (ICASI), Sapporo, Japan, 13–17 May 2017; pp. 637–640. [\[CrossRef\]](#)
18. Sarkar, S.; Chatterjee, S.; Misra, S. Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Trans. Cloud Comput.* **2018**, *6*, 46–59. [\[CrossRef\]](#)
19. Yuan, X.; He, Y.; Fang, Q.; Tong, X.; Du, C.; Ding, Y. An Improved Fast Search and Find of Density Peaks-Based Fog Node Location of Fog Computing System. In Proceedings of the 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, UK, 21–23 June 2017; pp. 635–642. [\[CrossRef\]](#)
20. Dong, Y.; Guo, S.; Liu, J.; Yang, Y. Energy-Efficient Fair Cooperation Fog Computing in Mobile Edge Networks for Smart City. *IEEE Internet Things J.* **2019**, *6*, 7543–7554. [\[CrossRef\]](#)
21. Shah-Mansouri, H.; Wong, V.W.S. Hierarchical Fog-Cloud Computing for IoT Systems: A Computation Offloading Game. *IEEE Internet Things J.* **2018**, *5*, 3246–3257. [\[CrossRef\]](#)
22. Mouradian, C.; Kianpisheh, S.; Abu-Lebdeh, M.; Ebrahimnezhad, F.; Jahromi, N.T.; Glitho, R.H. Application Component Placement in NFV-Based Hybrid Cloud/Fog Systems With Mobile Fog Nodes. *IEEE J. Select. Areas Commun.* **2015**, *37*, 1130–1143. [\[CrossRef\]](#)
23. Daraghmi, E.Y.; Yuan, S.-M. A small world based overlay network for improving dynamic load-balancing. *J. Syst. Softw.* **2015**, *107*, 187–203. [\[CrossRef\]](#)
24. MPIGate: Multi Protocol Interface Gateway. Available online: <http://mpigate.loria.fr/about.html> (accessed on 5 October 2020).
25. Gaurav, K. iFogSim: An Open Source Simulator for Edge Computing, Fog Computing and IoT. Available online: <https://www.opensourceforu.com/2018/12/ifogsim-an-open-source-simulator-for-edge-computing-fog-computing-and-iot/> (accessed on 20 December 2018).
26. Fog-and-Cloud-Computing-Optimization-in-Mobile-IoT-Environments. Available online: <https://github.com/JoseCVieira/Thesis---Fog-and-Cloud-Computing-Optimization-in-Mobile-IoT-Environments> (accessed on 20 January 2021).
27. Fog Computing. Available online: <https://github.com/imrahulr/FogComputing> (accessed on 20 January 2021).
28. Enhancing QoS in Fog Architecture Using P2P Load Distribution. Available online: <https://github.com/shashankshampi/SCloudSim> (accessed on 20 January 2021).
29. GitHub. Available online: <https://github.com/> (accessed on 20 January 2021).
30. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58. [\[CrossRef\]](#)
31. Yi, S.; Li, C.; Li, Q. A Survey of Fog Computing: Concepts, Applications and Issues. In Proceedings of the 2015 Workshop on Mobile Big Data—Mobidata '15, Hangzhou, China, 21 June 2015; pp. 37–42. [\[CrossRef\]](#)