A Framework for Seamless Offloading in IoT Applications using Edge and Cloud Computing

Himesh Welgama, Kevin Lee and Jonathan Kua
School of Information Technology, Deakin University, VIC 3220, Geelong, Australia

Keywords: Edge, Cloud, Docker, IoT, Offloading.

Abstract: Typical Internet of Things (IoT) deployments are resource-constrained, with limited computation and storage, high network latency, and low bandwidth. The introduction of Edge and Cloud computing provides a method of mitigating these shortfalls. This paper proposes a framework for structuring IoT applications to allow for seamless offloading (based on CPU load) of work from IoT nodes to Edge and Cloud computing resources. The proposed flexible framework utilises software to orchestrate multiple containerised IoT applications for optimal performance within available computational resources. Edge and Cloud servers co-operate autonomously to determine the appropriate resource allocation based on the requirements of running IoT applications in real-time. The result is a framework that is suited to perform with heterogeneous IoT hardware while improving overall computational performance, latency and bandwidth relative to IoT architectures that do not auto-scale. This framework is evaluated using an experimental setup with multiple IoT nodes, Edge nodes and Cloud computing resources. It demonstrates the approach is viable and results in a flexible and scalable IoT solution.

1 INTRODUCTION

The benefits for Internet of Things (IoT) applications of integrating sensing nodes, Edge and Cloud will lead to improved latency and bandwidth for IoT devices and also provide additional resources for resource constrained devices (El-Sayed et al., 2017). Edge and Cloud computing paradigm that is a bridge between IoT, Edge and Cloud computing, data is stored and pre-processed at Edge servers, only a small amount of data is then pushed towards Cloud servers if increased computation is needed. This process of transmitting data between different levels reduces the latency, assists in increase overall computational power along with boosting scalability. There are particular challenges in the convergence of IoT, Edge and Cloud computing for various applications (Biswas and Giaffreda, 2014; Kua et al., 2017),

A 3-tier Edge and Cloud infrastructure can address the limitations of Edge computing but these are difficult to build due to the coordination required. Data needs to be seamlessly transferred between the IoT node, Edge resources and Cloud resources without introducing unnecessary delay. These challenges can be mitigated through well designed frameworks and architectures that allow applications to be built that utilise IoT, Edge and Cloud computing resources. A three-tier architecture builds a hierarchy with inter-layer computation offloading to address this problem (Hwang et al., 2021). To this end, much research has also been done to realize the "Edge-Cloud Continuum" (Milojicic, 2020; Taivalsaari et al., 2021; Aloi et al., 2020).

The aim of the research presented in this paper is to improve the scalability and performance of IoT applications that use a 3-tier architecture. The contributions of this paper are: (i) the development of a framework that allows IoT applications to be built that can utilise Edge and Cloud resources seamlessly through the use of container-based virtualisation, (ii) mechanisms to support the automatic scaling (based on CPU load) of container-based IoT applications across edge and cloud resources, (iii) support for the dynamic addition and removal of IoT, edge and cloud resources in IoT applications, and (iv) an evaluation comparing the approach to one that doesn't automatically scale. The proposed approach allows for improved resource utilisation and improved capability for all nodes in IoT applications.

The remainder of this paper is as follows. Section 2 provides a background of the state-of-the-

Welgama, H., Lee, K. and Kua, J.
A Framework for Seamless Offloading in IoT Applications using Edge and Cloud Computing.
DOI: 10.5220/0011107500003194
In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, pages 289-296
ISBN: 978-989-758-564-7; ISSN: 2184-4976

art in the area. Section 3 proposes a new framework for scalable IoT Applications using offloading. Section 4 presents an experimental evaluation of the proposed framework. Finally, Section 5 presents some conclusions and future work.

2 AUTO-SCALING AND OFFLOADING FOR EDGE COMPUTING

Auto-scaling and offloading allow for IoT and edge applications to be able to meet processing demand. Fundamentally, auto-scaling is about increasing the amount of resources available to the task. In edge computing, auto-scaling represents a variety of challenges and can be implemented in a range of different ways (Taherizadeh and Stankovski, 2017). The work in (Dinh et al., 2017) proposes an approach to auto-scaling in mobile edge computing by scaling the CPU frequency on devices that support it. This is an example of utilising resources to the maximum, whilst potentially incurring costs, such as increased energy use or device life span. Auto-scaling is a core property of cloud computing, through the automatic additional and removal of cloned virtual machines (Lorido-Botran et al., 2014). This approach can be used in IoT, for example, increase the number of supported IoT or sensor nodes (Lee et al., 2010).

Offloading is a form of auto-scaling which utilises additional devices to increase the computational resources available for the task. In edge computing, there are a variety of approaches to supporting offloading of work (Mach and Becvar, 2017). (Zhang, 2018) takes a pricing based approach to choosing between different edge servers to offload data for processing. (Liu and Zhang, 2018) focuses on ensuring achieving low-latency communications through the use of offloading. There is also promising work to improve the battery life of mobile devices by offloading to nearby edge computing resources (Sardellitti et al., 2015). Recent work in container based approaches for IoT allows automatic failure-recovery for IoT edge applications (Olorunnife et al., 2021).

As increasingly IoT applications are being deployed using containers, and more processing is being performed in IoT applications (e.g. video processing), there will be a need for auto-scaling and offloading in these applications. Support for auto-scaling and offloading in container-based deployments is currently extremely limited, with none of the approaches discussed here targeting container-based deployments through the whole of the IoT architecture. As an example, (Wu et al., 2017) proposes the use of offloading in a container-based deployment for mobile computation, but only creates cloud-based containers to offload work directly to, and does not take a hierarchical scalable approach suitable for IoT. (Abdul Majeed et al., 2019) provide an analysis of the performance benefit in offloading critical tasks from edge to cloud, in a container-based environment, but this is focused on normally fixed services than IoT applications, and doesn't take into account the end node. Most container-based auto-scaling approaches, focus on a single layer, such as the edge or fog layer. For example, (Zheng and Yen, 2019) propose an approach to auto-scale Kubernetes containers to balance application performance and resource usage. This paper aims to demonstrate that auto-scaling for IoT applications through the use of offloading is a viable technique in container-based deployments.

3 PROPOSED FRAMEWORK

This paper proposes a framework for building scalable IoT applications with a focus on using Edge and Cloud computing resources. Existing methods (involving the use of container-based virtualization, Edge, Cloud and Fog computing as described in Section 2) often execute computational tasks in a siloed fashion, and often lack seamless cooperation between the various IoT/Edge/Cloud nodes.

The novelty of the framework proposed in this paper is in the real-time cooperation between IoT, Edge and Cloud nodes using a container-based offloading mechanism to satisfy performance requirements.

3.1 Design

Figure 1 illustrates the architecture of the proposed framework that aims to balance and satisfy the requirements. It adopts the popular 3-tier architecture of IoT, Edge and Cloud computing resources. The architecture aims to be scalable through the use of decentralized management, which allows for the introduction of new Edge or Cloud devices at any point in time. An application container can be run as a whole or as sub-tasks, depending on the tasks requirements. Host devices are connected to a private Docker registry and a Message Queuing Telemetry Transport (MQTT) broker. For evaluation purposes the program offers clients the ability to create Docker registries or MQTT brokers upon its first initialisation. MQTT is used as the primary

method of communication between the nodes. All nodes must be connected to the same MQTT broker.

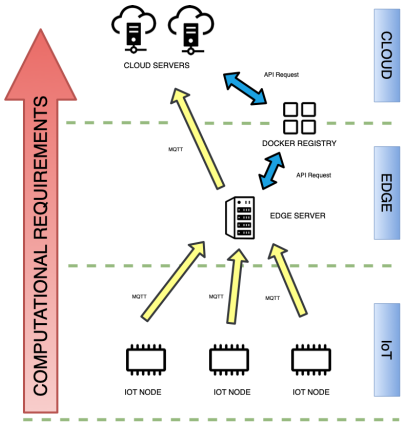

Figure 1: Overview of proposed architecture.

In the architecture, the *IoT node/device* is responsible for data collection and processing within the initial phase. Applications of low latency and minimum computational demands will run on the IoT device until the thresholds are met. Sub-tasks can be partitioned from larger applications and containerised, Each sub-task will have different application and computational requirements. When the sub-tasks increase their demands, they may exceed the processing capability of the IoT device. Under such circumstances, a request will be made to a local Edge server to share hardware resources. Containerised sub-tasks will be transitioned through a Docker registry for processing within the Edge server.

An *Edge node* is situated within the networks Edge. It aims to provide local optimisation through increased computational resources and improved latency for IoT devices. Edge nodes may undertake any task requested from IoT devices requiring additional resources. Furthermore, the availability of Edge nodes can be increased to meet scalability requirements of an application. Situations where a particular Edge node is unable to satisfy computational needs of a sub-task can request resources from the Cloud nodes. If a global optimisation approach is required, the applications can also be processed entirely across the Cloud nodes.

The *Cloud Node* can operate with a similar method to Edge nodes, providing global optimisation and increased computational resources. Sub-tasks that require access from a secure remote location will be controlled through the Cloud tier. As computational resources increase, the availability of Cloud resources provide more utility than constructing new Edge servers. Cloud nodes can communicate with each other to distribute their computational loads effectively, e.g., using load-balancer and auto-scaler (Pan and McElhannon, 2017).

Applications can be separated into sub-tasks and distributed across Cloud or Edge nodes. However, they require a method of orchestration to allow for the whole system to perform correctly. Once an instance of this infrastructure scales, decentralized communication can be deployed to avoid any vendor lock-in or possibilities of information loss. MQTT is a lightweight protocol that offers a flexible Publish/Subscribe model. This architecture allows for any additional Edge or Cloud node to subscribe to open application requests, without requiring databases or centralized management systems. As long as a system is connected to the same MQTT broker, it can scale to 'n' size.

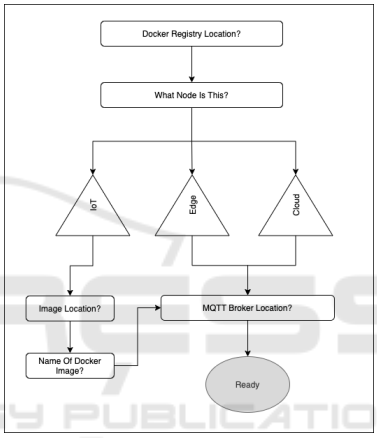

Figure 2: Initialisation of nodes within the framework.

To ensure safe orchestration of applications within the system, a configuration program will run inside the operating system of each node. The program will track the statistics of each IoT service within the pool and relay MQTT messages and Docker API calls when necessary. To improve computational performance of this system, new nodes can be added and initialised.

Node.js is used in the proposed framework. Each node within the system is initialised at the start of each experiment.

- All nodes must have Docker Engine installed
- Required thresholds for application limits need to be decided prior to configuring nodes
- MQTT Broker and Docker Registry are set-up
- Docker application must be saved as .tar file within IoT node prior to execution

A series of system checks are performed prior to setting up each node on the IoT framework. Figure 2 details the required responses upon initialisation.

Figure 3 shows an example of the configuration file created following the initialisation of a node within the framework. This figure describes the necessary information that's required in order to establish a connection between the Docker registry, Docker image, MQTT broker and other nodes within the framework.

```
{
  registryLocation: '192.168.0.0:5000',
  nodeType: 'iot',
  imageLocation: '/image-name.tar',
  imageName: 'Hello-World',
  mqttLocation: '192.168.0.1'
}
```

Figure 3: Node configuration JSON file.

4 EVALUATION

This section presents the experimental setup and evaluation results of the proposed framework. The first two evaluation scenarios focus on the validation of the proposed system while the third evaluation scenario compares the real-world performance of a traditional non-auto-scaling IoT architecture and the proposed framework.

4.1 Experimental Setup and Network Configuration

The experimental test-bed setup consists of the following:

- Raspberry Pi 3B+ with Wifi connection (IoT node)
- x86 Intel i5-3570s @ 3.10GHz, 4GB RAM, 300GB HDD (Edge node)
- 4 Core Intel x86 CPU, 16Gb RAM, 25GB SSD system (Cloud)
- 1 Core Intel x86 CPU, 1GB RAM, 25GB SSD system (Cloud for Evaluation 1)

The network configuration for the experiments is illustrated in Figure 4 and described as follows.

- IoT nodes connected to gateway through Wi-Fi.
- An edge node that uses a Gigabit wired Ethernet connection to the gateway. This has a maximum achievable download speed of 1000Mbits/s
- The network gateway from the selected Internet Service Provider (ISP) is capped at 250Mbits/s download and 20Mbits/s upload.
- A Cloud server is located in close geographical proximity (Sydney Australia)

4.2 Scenario 1: Migration of IoT-Edge-Cloud Nodes

This scenario analyses the impact of node movement based on the available resources by visualising the time between the transfer of a Docker container to a higher level node in the framework. A Linux-based software library *stress-ng* (King, 2017) is used to create artificial load for performance bench-marking and testing. In these experiments, the software stress tests each CPU to 25% on each thread. The command was encapsulated within a Docker image that was containerised and used for each node.

A counter is used to track the CPU percentage from when the Docker image is ran to the moment it has reached its maximum threshold. On the Edge and Cloud nodes, the counter will gather CPU percentage as soon as the MQTT message is received. This is used to calculate the downtime between the transfer of images between nodes. Each node along the system has specific maximum thresholds placed based on their requirements. This is less relevant in this experiment as the primary focus is on observing the transfer of Docker images between nodes.

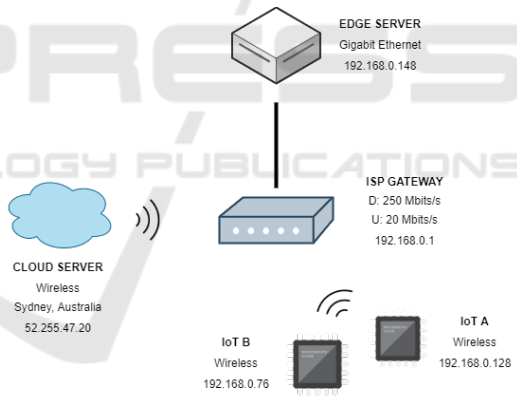

Figure 4: Network configuration for the evaluation.

Figure 5 presents the evaluation results. It is observed that the transfer of Docker image from IoT to the Edge node has negligible downtime. This is due to the location of the Docker registry location being within the Edge node itself, therefore drastically reducing the amount of information that has to be downloaded. The Cloud node has significantly more difficulty in downloading and running the Docker image from the registry, which resulted in an almost 20-second delay in data loss between the moment the Edge node stopped to when the Cloud node started. This delay is heavily dependent on the networking configuration of Edge Server and Cloud Server. This scenario demonstrated the efficiency of

data transmission from the IoT node to the Edge node. As the system requirements become more complex, a more logical approach would be the use of task splitting with the proposed architecture.

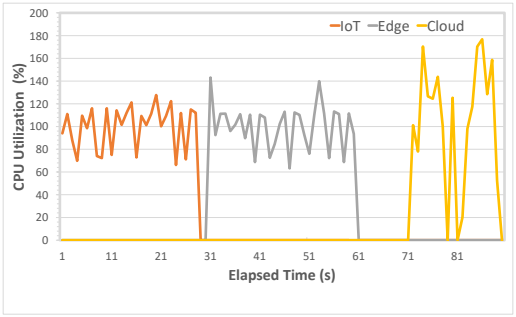

Figure 5: Scenario 1: Application movement when thresholds are met.

4.3 Scenario 2: Aggregation of Sub-tasks

Scenario 2 evaluates the separating of an application into multiple containerised sub-tasks with the goal of mitigating downtime and improving the utilisation of each node. Based on thresholds, sub-tasks are automatically set to migrate to a higher-level node if a threshold is met. This evaluation hypothesises a greater reduction in downtime between tasks, and efficient resource allocation within nodes. The main application is split between three separate tasks, ranging from 25%, 50% and 75% CPU load utilised in each task respectively. Tasks are initially deployed on the IoT node as one application and are moved to a separate node if the threshold is met. In this experiment, the threshold values are not significant, however the movement and CPU% utilisation of each sub-task is important as this value will indicate the successful migration/movement and optimal usage of compute resources.

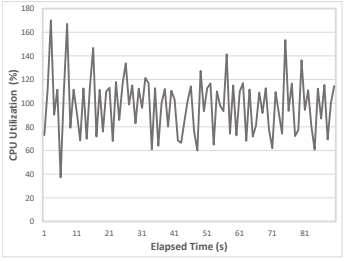

Figure 6: Scenario 2: Sub-task 1, Avg 25% load per thread.

Figures 6, 7, 8 present the CPU utilisation for each sub-tasks at 25%, 50% and 75% CPU load on each thread. Figure 9 overlays all three graph plots and compares the performance across the three sub-tasks.

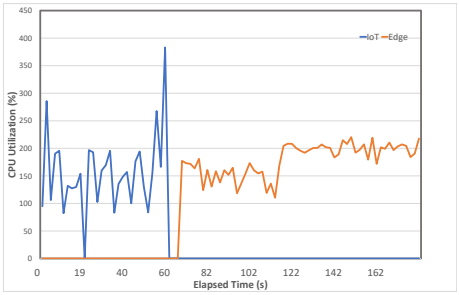

Figure 7: Scenario 2: Sub-task 2, Avg 50% load per thread.

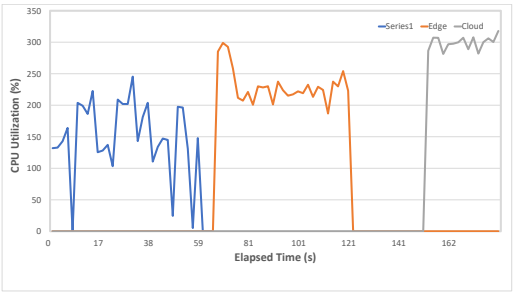

Figure 8: Scenario 2: Sub-task 3, Avg 75% load per thread.

In this evaluation, all sub-tasks successfully run on the most optimal node. The approach uses a 1 minute time window to examine the average CPU usage compared against a set threshold. During each 1-minute interval, each sub-task will re-evaluate its computational requirements and move the application to a higher level node if necessary. The variability of CPU utilisation percentages between each sub-task are influenced by many different factors. This evaluation focuses on the node movement and average CPU load utilisation. The two key takeaways from this evaluation are:

1. The application does not stop running during periods of downtime between transitioning from nodes. This can be observed from Figure 8, as the migration of sub-task from Edge node to Cloud node has caused a 20-second delay. During this period, sub-tasks 1 and 2 are still running at optimal performance as seen in Figure 9.
2. Performance of sub-tasks on each node will be below ideal when the host is executing more than one task running at similar or higher CPU percentage. This can be observed in Figures 7 and 8 where each sub-task is running at a higher CPU percentage. This result indicates that more computational resources are being allocated as the operating system increases the priority level for the particular task, and CPU throttling. Figure 7 demonstrates the increase of CPU percentage following the Cloud migration of sub-task 3.

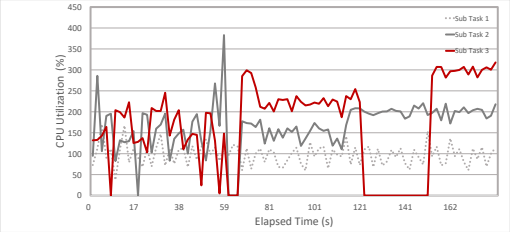

Figure 9: Application view: Aggregation of all sub-tasks.

Scenario 2 validates the framework for multi-task programs and evaluates the effectiveness of utilising IoT/Edge/Cloud nodes for optimal performance.

4.4 Scenario 3: IoT Node Optimised for Image Detection

An IoT framework that autonomously optimises each node to perform at its peak has been validated in Scenarios 1 and 2. Scenario 3 represents a use-case scenario in the real world. This use case involves real-time image detection with Machine Learning on IoT/Edge/Cloud nodes. The application of IoT technologies for image detection is rapidly growing, e.g., the need for robust face/mask detection and social distancing during the COVID-19 pandemic.

This scenario evaluates the performance of each different node to process face detection, and compares the performance to a traditional Cloud approach. The deployed application gathers an image of an artificially generated face database and perform image detection algorithms. This process is repeated many times and the response times between the generation of image to detection are recorded along with the total number of faces processed within the elapsed time frame. Using this metric, the performance of the proposed architecture and its usability in real-world applications can be further investigated. The ability of the proposed architecture to handle multiple IoT nodes, and the performance impact of larger applications can also be evaluated.

An experiment is conducted to evaluate a traditional multi-device IoT architecture against the proposed three tier architecture. This scenario utilises two Raspberry Pi 3B+ IoT devices that work concurrently to perform the same task. Both IoT devices (IoT A and IoT B) vary the amount of threads that are used to perform the application. IoT A uses four threads whereas IoT B uses two threads. This is purposely done in order to maximise the CPU usage where available resulting in the highest possible output for either architecture and showcase variability in the IoT devices used. Detailed specifications for the experimental setup can be observed in Sub Section 4.1. A second experiment is conducted with the same setup, also using the same three-tier architecture. The containerised application runs on two Raspberry Pi 3B+ boards separately. This evaluation initially runs on a traditional IoT architecture, where the device uses HTTP GET to retrieve the image and executes the ML algorithm. Next, the application runs within the proposed architecture.

Figure 10: IoT A (four threads): Traditional approach.

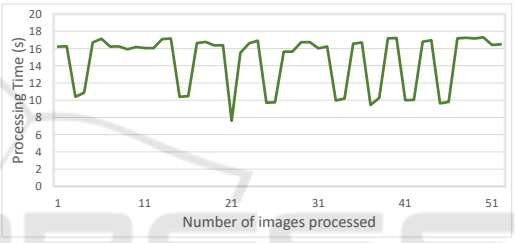

Figure 11: IoT B (two threads): Traditional approach.

The test was performed for 180 seconds, with results presented in Figs. 10 and 11). IoT A had faster processing time given the higher number of threads. The extra two threads resulted in roughly 129% increase in the number of faces processed. The traditional approach gathered 177 processed faces within the time frame.

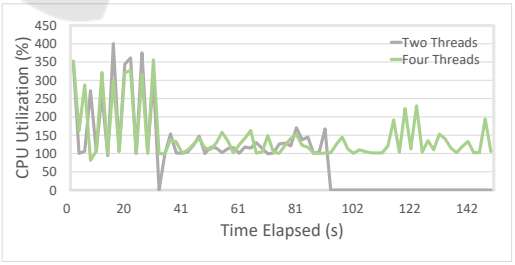

Figure 12: Results with proposed architectural framework: IoT A (four threads) and IoT B (two threads).

The same containerised application was then launched from both IoT devices within the proposed architecture. As illustrated in Figure 12, the application was successful in transitioning the application accordingly based on the CPU usage threshold. It can be noted that the initial erratic CPU fluctuations are due to the lower computational power of the Rasp-

berry Pi, this same pattern is seen on Figure 6. The periods of which either application migrates from one host to another can be observed in sections of the graph where the CPU % dips to 0. During the application, IoT A (two threads) was above the set threshold therefore requiring a migration to the Cloud server. However, this migration was unsuccessful due to major delays in the Cloud host to download the requested Docker container. Furthermore, the 'Four Threads' application remained on the Edge node, most likely due to the increased computational resources available while the 'Two Threads' application remained offline in its attempt to migrate to the cloud node. A conclusion is drawn from this initial testing: The proposed architecture is not designed to be suited for large applications with sizes ranging above 1GB as this induces further strain on the networking capabilities of the involved nodes. The architectural design of the proposed framework will need to be adapted to suit larger applications.

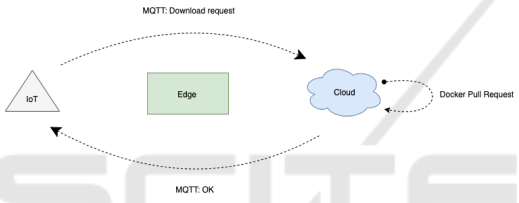

Figure 13: Workflow of 'pre-loaded' approach.

A 'pre-loaded' approach can be taken to ensure that the application is loaded within the Cloud server prior to the initialisation of the IoT node. Figure 13 demonstrates that a possible solution is a Publish/Subscribe protocol to indicate completion of pre-loaded images in the Cloud server. Complex applications may cause high storage and network load when 'pre-loading' multiple applications.

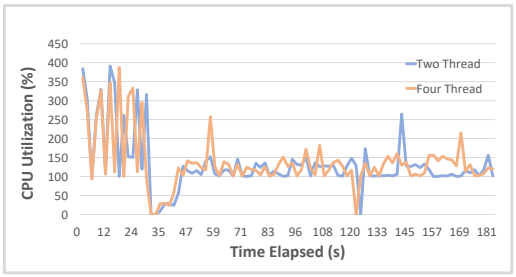

Figure 14: Proposed framework with pre-loading.

The effects of pre-loading images onto the Cloud server is illustrated in Figure 14. The CPU utilisation shows that both applications had made use of all available computational resources as transitions between nodes can be observed through CPU% dipping to 0% as seen in Figure 9 and 8. Successful migration of the containerised applications to the cloud node was made possible through the use of pre-loading the application in the Cloud server, thereby, allowing the two IoT applications to preform optimally with higher computational resources. With comparison to the first evaluation made on the proposed architecture, this evaluation resulted in around 2-second delay between the Edge node and Cloud node. The performance of the application to successfully process faces (images) from the proposed architecture can be compared against the traditional approach of running IoT A and IoT B individually (See Figure 10 and 11).

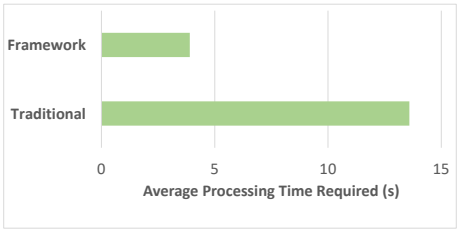

Figure 15: Comparison: Processing time.

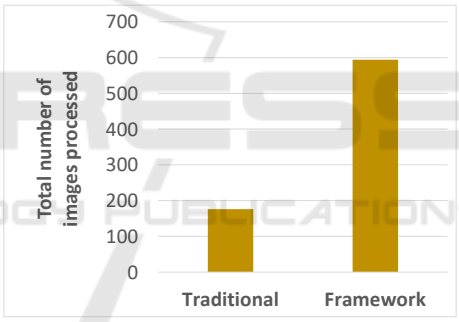

Figure 16: Comparison: Overall performance.

A comparison can be made between the two approaches, by comparing the total amount of images processed and the total processing time. The proposed architecture is able to outperform the traditional IoT system in both processing time and overall performance. This indicates the movement of nodes between IoT to Edge and Cloud servers are successful in providing the most optimal environment for each application to thrive. Figure 15 shows a 10-second gap between processing averages of the two systems. Figure 16 presents a 237% increase in total faces processed through out the three-minute elapsed time. Further evaluations can be performed to assess a many-to-many situation where multiple IoT nodes are sending data to multiple Edge and Cloud servers. This will provide extra validation for the scalability for such architectures in real-world applications.

5 CONCLUSIONS

In this paper, a framework for seamless offloading for IoT applications using Edge and Cloud computing for container-based applications was proposed. It aims to overcome the issues of traditional IoT systems by utilising a sub-task methodology in deploying IoT applications for optimal performance under the proposed architecture. It evaluated in a variety of scenarios to demonstrate offloading to Edge and Cloud resources under load. The evaluation demonstrated that in real world applications, it outperforms traditional IoT applications. However, it is noted that the work presented in this paper is a proof-of-concept demonstration and further work is needed. Our work has not yet taken into account real-world applications that have stringent real-time latency requirements. Future work will seek to address these issues and investigate the seamless integration of the proposed framework, in particular its offloading capability with common IoT development frameworks

REFERENCES

Abdul Majeed, A., Kilpatrick, P., Spence, I., and Varghese, B. (2019). Performance estimation of container-based cloud-to-fog offloading. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, pages 151–156.

Aloi, G., Fortino, G., Gravina, R., Pace, P., and Savaglio, C. (2020). Simulation-driven platform for edge-based aal systems. *IEEE Journal on Selected Areas in Communications*, 39(2):446–462.

Biswas, A. R. and Giaffreda, R. (2014). IoT and cloud convergence: Opportunities and challenges. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 375–376. IEEE.

Dinh, T. Q., Tang, J., La, Q. D., and Quek, T. Q. (2017). Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584.

El-Sayed, H., Sankar, S., Prasad, M., Puthal, D., Gupta, A., Mohanty, M., and Lin, C.-T. (2017). Edge of things: The big picture on the integration of edge, iot and the cloud in a distributed computing environment. *IEEE Access*, 6:1706–1717.

Hwang, R.-H., Lai, Y.-C., and Lin, Y.-D. (2021). Offloading optimization with delay distribution in the 3-tier federated cloud, edge, and fog systems. *arXiv preprint arXiv:2107.05015*.

King, C. I. (2017). Stress-ng. *URL: http://kernel. ubuntu. com/git/cking/stressng. git/(visited on 28/03/2018)*.

Kua, J., Armitage, G., and Branch, P. (2017). A survey of rate adaptation techniques for dynamic adaptive streaming over http. *IEEE Communications Surveys Tutorials*, 19(3):1842–1866.

Lee, K., Murray, D., Hughes, D., and Joosen, W. (2010). Extending sensor networks into the cloud using amazon web services. In *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, pages 1–7. IEEE.

Liu, J. and Zhang, Q. (2018). Offloading schemes in mobile edge computing for ultra-reliable low latency communications. *IEEE Access*, 6:12825–12837.

Lorido-Botran, T., Miguel-Alonso, J., and Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592.

Mach, P. and Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656.

Milojicic, D. (2020). The edge-to-cloud continuum. *Computer*, 53(11):16–25.

Olorunnife, K., Lee, K., and Kua, J. (2021). Automatic failure recovery for container-based iot edge applications. *Electronics*, 10(23):3047.

Pan, J. and McElhannon, J. (2017). Future edge cloud and edge computing for internet of things applications. *IEEE Internet of Things Journal*, 5(1):439–449.

Sardellitti, S., Scutari, G., and Barbarossa, S. (2015). Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103.

Taherizadeh, S. and Stankovski, V. (2017). Auto-scaling applications in edge computing: Taxonomy and challenges. In *Proceedings of the International Conference on Big Data and Internet of Thing*, BDIOT2017, page 158–163, New York, NY, USA. Association for Computing Machinery.

Taivalsaari, A., Mikkonen, T., and Pautasso, C. (2021). Towards seamless iot device-edge-cloud continuum. In *International Conference on Web Engineering*, pages 82–98. Springer.

Wu, S., Niu, C., Rao, J., Jin, H., and Dai, X. (2017). Container-based cloud platform for mobile computation offloading. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 123–132.

Zhang, T. (2018). Data offloading in mobile edge computing: A coalition and pricing based approach. *IEEE Access*, 6:2760–2767.

Zheng, W.-S. and Yen, L.-H. (2019). Auto-scaling in kubernetes-based fog computing platform. In Chang, C.-Y., Lin, C.-C., and Lin, H.-H., editors, *New Trends in Computer Technologies and Applications*, pages 338–345, Singapore. Springer Singapore.