

Creating A Scalable Smart Healthcare System Through The Utilization Of Edge/Fog Computing

Rhaed Khiati*

*Department of Computer Science, Toronto Metropolitan University, 350 Victoria St., Toronto, ON

rkhiati@torontomu.ca

Abstract— In the era of digital transformation, healthcare systems are evolving towards smart infrastructures to provide personalized, efficient, and timely services. However, the scalability and real-time processing demands of such systems pose significant challenges. This paper proposes the integration of edge and fog computing paradigms to address these challenges and create a scalable smart healthcare ecosystem. By leveraging edge devices and fog nodes situated closer to the point of data generation, computational tasks can be offloaded from centralized servers, reducing latency and bandwidth requirements. This distributed architecture enables real-time data analysis, facilitates timely decision-making, and enhances the scalability of healthcare services. Through a comprehensive review of existing literature and case studies, we elucidate the benefits and challenges of adopting edge/fog computing in healthcare settings. This analysis underscores the potential of this approach to revolutionize healthcare delivery, improve patient outcomes, and drive innovation in the field.

Keywords— Smart Healthcare, Edge Computing, Fog Computing, Distributed Systems, Scheduling Algorithms

I. INTRODUCTION

The production and advancement of smart cities in the 21st century is a foregone conclusion rooted in the development of our ever-growing technological landscape, and this is especially true with regard to smart healthcare systems and subsequently, the development of intelligent hospitals. Through such developments, we are now able to design and program rapid and smart AI-driven systems, interconnected through the Internet of Things (IoT) on devices like smart hospital beds and hospital room controls, to name a few, all collecting copious amounts of data through both active and passive patient interactions and providing feedback, all in an attempt to improve the overall quality of life of the patients residing in the hospitals or relying on these smart systems in general. Unfortunately, as these systems increase in scope and complexity, so does the number of patients using these systems, and with it comes an increase in both the production and usage of this patient data, which will definitely provide a future obstacle for system developers and on a more general level, for hospitals at large. Cloud computing has been proposed as a model and methodology by which to circumvent these issues, but the cloud also deals with its own fair share of issues, ranging from high latency to increased resource usage and overall cost, among other points, with the overall takeaway here being that cloud computing cannot solve the

smart hospital dilemma alone, hence the proposal of a more sustainable model for these smart healthcare systems using a combination of edge and fog computing as the basis for this new model in conjunction with the cloud.

The concept of developing a smart healthcare model based around edge and fog computing is hardly a novel idea as the idea has been floated around and researched due to the benefits provided by edge and fog computing with regards to these systems in particular, namely, in the fact that the processes headed by fog computing occur closer to the development and usage of the obtained data from the IoT devices. Both edge and fog computing provide an additional level of processing and filtering of patient data as it comes in from the various sensors and other IoT devices, while also facilitating a communication line between the entire network leading to the cloud, which reduces latency, provides an additional level of fault-tolerance and in the long run, can also reduce the overall resource cost, which in turn provides both doctors and patients with quick and accurate access to their data, and feedback on how to proceed on any unexpected medical emergencies, among other benefits.

Ultimately, the addition of edge and fog computing does not make this project particularly ambitious or unique on its own, seeing as at least in theory or in minor practice, the idea of applying fog computing to smart healthcare has already been done, and fog computing itself has its limitations when it comes to managing data streams, among other issues, hence the application of Apache Beam, a distributed streaming framework, to the system as well to help alleviate a sizable number of the concerns that come with edge and fog computing. As will be explained later, Apache Beam is effectively the latest streaming framework from the ever-expanding Apache banner that offers a lot of the advantages of its predecessor Apache Flink, like event-time bookmarking and improved latency, among others, while also offering other unique benefits like the utilization of a unique and singular API; A custom static scheduler will be implemented with Beam to schedule tasks and parallelize the entire process, with the intent to make a more dynamic scheduler down the line once successful.

The rest of this proposal is structured as follows. Section 2 will contain the literature review, highlighting the related works, while section 3 will highlight the system methodology, including the project objectives, intricacies of the system design, and a look at the scheduling algorithm. Section 4

offers an insight into the parameters used for testing as well as further information regarding the results and outcomes, before concluding this proposal in section 5.

II. LITERATURE REVIEW

The key point about this project is essentially the idea that this project is an amalgamation of several tried and tested concepts and systems that have been combined to form a singular and novel entity that has not been tried and tested to date. As seen in this section, a number of the related works concern themselves with one or a couple of the ideas that are presented in this proposal, namely the usage of fog/edge computing in smart cities, the use of Apache Beam, the scheduling of tasks in smart healthcare systems, and so on, but none have unified the above into a single novel entity. The following represent the aforementioned works that are relevant for the purposes of this project.

The first area that we focused on is the concept of fog/edge computing and its relation to smart cities, a topic that has been widely discussed and built upon, though not as frequently in the area of smart healthcare. The general idea of this trend is to take full advantage of the versatility and processing capabilities of fog computing to lighten the load on cloud computing, thereby providing a quicker and more efficient processing and analysis of data obtained from the various IoT sources. Hassan et al. [1] suggested the use of fog computing architecture for the deployment of a remote pain monitoring system, with the ultimate goal being the development of an enhanced fog paradigm to reduce latency and network consumption. To analyze the effectiveness of their proposed approach in reducing delay and network resource cost/usage, simulations were carried out in iFogSim and the results were compared when concerned with the effectiveness of edge computing and a linear cloud computing system with no other nodes involved. The results of the simulations carried out in this research state, according to the authors at least, that a reduction in both latency and network utilization can be achieved through their proposed approach. To summarize their approach in more detail, they proposed a 3-tier fog architecture, consisting of the sensors layer which collects data from a variety of sensors attached to patients in a hospital (namely sEMG and ECG signals being sent), the fog layer which is seen as the central layer where all of the data gathered through the sensors is sent to, and where the data is processed, and finally, the cloud layer which acts as sort of a big data warehouse while also being the place where the more copious amounts of big data are processed. The sensors are all interconnected through a Wifi module, and continuously feed their data to the fog layer. The authors utilized fog computing because of its interoperability with various IoT devices, in this case healthcare devices like heart monitors, sensors and the like, and its use of signals to help differentiate between the various patients and their level of pain. The cloud layer is there to provide extra processing firepower and storage, and honestly, could possibly even be avoided, which is how this proposed architecture reduces latency and network

consumption, as its data is submitted straight to the fog devices to process said data, while leaving the cloud to handle the bigger matters.

Tuli et al. [2] proposed HealthFog, a system designated primarily for heart patients and other users that need full-time access to their respective data, combining an edge-fog-cloud computing approach with ensemble deep learning techniques, and their contributions included creating a lightweight generic architecture, using FogBus as a base for the backend, demonstrated and analyzed throughout the paper. HealthFog consists of a number of components, including a data filtering component which preprocesses data to extract key patient info, a resource manager to allocate tasks, a deep learning module to train neural networks to classify patient data and predict results for content that arrives from the gateway, and finally an ensembling module (located in the FogBus assigned this task) which based on the results it obtained from all different models, the system decided (through voting) whether or not the patient has heart complications. For their results, the authors found that prediction accuracy was tied to the number of nodes used, as each node receives a subset of training data, so more nodes means less accuracy. Prediction confidence was tied to correct prediction accuracy. Latency and bandwidth usage varied depending on the results scenario from earlier.

There is legitimately a lot to digest from that previous paper and of the related works, but the key point necessary for the project is the usage of ifogsim, the fog simulator tool that was developed recently. Ifogsim [3] is a high-performance open source toolkit for fog computing, edge computing and IoT, which is used to model and simulate the networks of edge computing, the Internet of Things and fog computing. iFogSim enables the modeling and simulation of fog computing to evaluate resource management and scheduling policies across edge and cloud resources under different scenarios. Long story short, ifogsim allows developers to simulate a fog computing environment, as they have done in [1], to analyze and assess the effects of fog computing in an IoT/cloud environment and to gauge the benefits of fog computing in comparison with other methods. Just recently, Mahmud et al. [4] actually developed an updated version of ifogsim, known unsurprisingly as ifogsim2, or rather an “Extended iFogSim Simulator for Mobility, Clustering, and Microservice Management in Edge and Fog Computing Environments” as they put it. The idea or problem that the authors were trying to solve with this enhancement focused more on the scalability of the edge/fog computing environment rather than efficient application of said environment, which the authors feel has been thoroughly neglected by developers. The authors proposed additions that include a service migration simulation model that can operate across multi-tier infrastructure and support simplified integration of real-world datasets (mobility), a proposed dynamic distributed cluster formation among the multi-tier infrastructure of edge/fog computing, to provide a higher QoS (clustering), and the development of an orchestration model for microservices deployed across multi-tier infrastructure, which enables placement policies to dynamically scale

microservices among federated Edge/Fog nodes to improve resource utilization (microservices).

Of course, the experimentation with ifogsim, and subsequently the newer model ifogsim2, still raises a few concerns, including the capacity to handle a larger, more secure batch of patient data, especially with time as the number of patients increases and smart technology continues to develop. For this, we are proposing the integration of Apache Beam, a distributed streaming framework that is the most recent streaming engine developed by an open source community under the umbrella of Apache, though initially developed by Google, one capable of real-time data processing of millions of events per second, using the least amount of resources, or at least as many as required. Beam [5] has a variety of features and solutions which makes it unique in comparison to prior streaming frameworks developed by Apache, including the usage of a singular, unified API for all batch and streaming use cases, the ability to execute pipelines in a variety of runtime environments, and the ability to write and create custom SDKs and transformation libraries, among others, with further detail regarding its benefits to this PhD research to be described in the progress section. Perhaps due to its recent release, there aren't a large number of works on the topic of integrating Beam with smart healthcare or even fog computing, but there was one that struck my attention as relevant to this project. FogFlow [6] was designed as a development model for dataflow applications that could span all fog nodes as well as the cloud, a model that could enable the defining of data processing pipelines split further into sub-pipelines that run on heterogeneous IoT environments, a similar concept to Apache Beam. It has two main requirements in mind, namely that the application definition should only contain the logic of data processing as well as be independent of any infrastructures, and its execution should be possible over multiple platforms. Consists of three sections: The API, which creates the pipelines, the graph module which transforms and modifies data to be represented as a graph and the execution module responsible for translating the graphs into executable targets for the heterogeneous environments. For the execution of FogFlow's infrastructure, Apache Flink, a slightly older and simpler streaming framework compared to Beam which will be described in the next section, was used to run applications in the cloud, while Edgent, a stream processing model from Apache built specifically with the lower processing capabilities of edge computing in mind, was used in edge nodes due to more limited resources, and a simple C based runtime was used for the lower-level microcontrollers. The execution module was divided into submodules for each of the three sections. Testing was conducted using 2 schemas: Testing on the cloud alone, and testing on the whole framework. Testing on the cloud revealed a bottleneck issue, while in the second test, Apache Edgent was revealed to have reduced data processing in the cloud, even in an environment with constrained resources.

The last area related to this project is with regards to the usage of schedulers in the context of setting priorities for the processing and analysis of healthcare data in the context of

smart systems. Jamil et al. [7] wrote a paper to create a potential solution to the resource allocation and job scheduling of fog computing, with the primary goal being the reduction of the response time and energy usage. Applying their work to healthcare as a case study, they gathered data from sensors and sent them to the fog nodes in the form of tuples. Once there, the tuples were sent to the scheduler, which checked if it was the first tuple in the device or not. If it was, it was executed and if not, it was sent to the queue found in the fog node. They tested on a number of different parameters, including loop delay, energy configuration and network usage, and found that the delay was reduced, while energy configuration and network usage increased by 32% and 16% respectively. Alsamadi et al. [8] attempted to curate and develop a scheduling algorithm for use in smart cities, using a weighted round-robin (WRR) scheduler to schedule tasks from one fog node to the next (eventually leading to the cloud) based on incoming data from the IoT. Communication between the layers was conducted over a 5G network to allow for rapid transmission of data, and the WRR algorithm, where tasks are executed in a cycle, and more time is allocated to tasks that can be done in more time, with tasks in each cycle being allocated based on job-priority proportion rather than what the system dictates, meaning that ultimately the response time is longer for smaller jobs to ensure fairness. Results saw the authors achieve a greater throughput as well as a lesser delay and computational complexity, as a result of using the WRR (as well as the 5G network and a spanning tree routing protocol). Verma et al. [9] proposed a scheduling algorithm for fog computing that utilizes a load balancing technique based on task deadline, network utilization, and increased throughput. Scheduling algorithm balances the load through the execution of the available network bandwidth, executing tasks before their deadline and offering a rapid response to users. Testing was run on CloudSim, with a variety of different workloads being experimented on, trying to emulate a real-time effective workload scheduling algorithm and comparing it with other known, static algorithms like priority scheduling and first come, first serve. Results highlighted a minimal execution time, fast response time and completion of tasks prior to their deadline, all while maintaining a level of data consistency. Aladwani et al. [10] proposed a model to schedule healthcare task priorities prior to running them through a MIN-MAX scheduling algorithm over a wireless sensor network. They prioritized tasks based on importance regardless of length, using a new method they coined as a "Tasks Classification and Virtual Machines Categorization (TCVC)", which classified tasks as high, mid and low priority based on patient status. Simulations were run on CloudSim, with impact measured based on execution, waiting and finishing time, and tasks were split into three different categories: Class 1 received general, low priority tasks run on a low performance VM(s), Class 2 received important, mid priority tasks run on VM(s) with medium performance and Class 3 received critical, high priority tasks run on VM(s) with high performance. Results indicated better performance in all

three measured parameters while also offering fairness between tasks and improved load balancing.

III. METHODOLOGY

A. System Objectives

The goal of this project is to create a more scalable and reliable smart healthcare system through the usage of some of the more prominent systems utilized in IoT and smart technology today, namely edge/fog computing, streaming frameworks, and task scheduling to create a unique and innovative system designed to provide a blueprint for not just smart healthcare systems, but smart technology in the long-term. As mentioned earlier, hospitals today rely on vast amounts of big data, an amount that is increasing by the day as the number of patients and the population increases, and both new ailments and new cures are found globally, as people seek more reliable methods of treatment. To handle this massive increase in data, we are proposing a scalable system that utilizes edge and fog computing in conjunction with task scheduling based on task deadline for quicker and more efficient results, and Apache Beam which offers a distributed framework that can manage all of these changes efficiently.

Further details are included below. To achieve this goal, the following objectives must be met:

- The characterization of a typical intelligent health care application, and obtaining an open source patient dataset for use in initial testing and simulation of the aforementioned application. There are a number of online sources that can be utilized for this project, which is necessary to simulate a healthcare system and its stakeholders' data (patient, doctor, nurse, admin, etc.) can access the system globally in a secure and stable and scalable manner
- After characterization of the workload and data, scheduling the tasks is the next step. The development and implementation of a custom scheduler that will be used to assign tasks and resources based on the task's deadline, is the main objective of this step which will be explained in a later section. The following work prioritizes a more static approach to scheduler design, intending to switch to a more dynamic scheduler by exploiting the advances of ML and Deep learning methods
- Connecting the scheduler to Apache Beam to manage the big streaming data. This requires analyzing both the data coming in and the tasks that have been scheduled, in preparation for pipelining them to their respective edge/fog nodes where said data will be processed or tasks will be implemented. Massive scale tasks/data will be sent straight to the cloud

B. System Design

The idea of implementing edge/fog computing in conjunction with smart healthcare systems has been done quite a bit up to this point, as evidenced by the literature

review, but given some of the known limitations that edge/fog nodes have with regards to the increased complexity on a system, as well as the fact that only smaller scale testing has been conducted in this area, Apache Flink could be viewed as a method to help mitigate such issues. Flink [11] offers a number of advantages when handling large data streams like its use of event bookmarking to deal with fault-tolerance issues by rolling back to a specific 'event' determined by the parameters that were set, and the ability to handle data in real-time, among other benefits. I had worked with Flink before for a previous work that I wrote [12], and have seen the benefits it could potentially offer healthcare systems and providers at large, not to mention that it could easily be integrated into a fog node, and in fact, there has been a work that attempted this idea previously through the platform known as FogGuru [13], designed to facilitate the running of applications in a fog environment, using Apache Flink as a medium of processing the data from the IoT to the fog node itself.

Unfortunately, a small problem reared its ugly head in the form of a bottleneck, amplified by what I soon found out was one of Apache Flink's glaring flaws, that being that Apache Flink was incredibly individualistic, and would only run within the confines of the edge/fog node it was set up in, so if you wanted to utilize Flink, you would have to set it up across all if the nodes that would be utilized, which in the case of this work, is at least several of them. On top of that, the Flink APIs cannot communicate with each other to relay information about any data that had been analyzed, meaning that parallelization was impossible to achieve, and a bottleneck was formed. Results were also contradictory and often overlapped with the different results for certain tasks that were illogical (one node declared that a patient had no issues, while another declared that a patient was going through internal bleeding for example). On top of that, with no task scheduling available, some nodes were often overworked while others were not given any tasks, and latency increased as a result as well as unnecessary resource usage. This was the end result of my initial testing, and although some of the above literature did not run into this problem, that was because a number of them used a single-tier fog architecture with only one fog node involved, which is not ideal for the type of work intended for this project, and this is without mentioning the hardware, latency and bandwidth limitations associated with adding Flink to each edge/fog node. Ultimately, Apache probably realized the same limitations, and as a result, developed an enhanced version in the form of Beam.

As mentioned in the literature review section, Beam corrects these faults that Flink has through the usage of a unified API, meaning you only need to place Beam once and that's it, and with Beam being portable and able to run in any environment, it can be placed anywhere in the system, like say between the data pool and the edge/fog nodes. The best part about this is that you don't even have to compromise or remove Flink; You can still keep Flink in the nodes, but it will operate based on instructions provided by Beam in a sort of master-worker relationship, allowing us to get the benefits of

both frameworks. Having made that adjustment, the final design can now be developed, as seen in Figure 1. Data is collected from a variety of sources, primarily from IoT devices like smartphones or smart hospital beds, as well as a variety of sensors, and pooled together into a shared pool. Data is then pipelined to Beam, where data is analyzed and processed, and simultaneously, the scheduler pre-schedules the tasks to be executed based on the expected deadline of the task. The reason for choosing this parameter in particular is due to the necessary urgency that data needs to be analyzed and tasks need to be executed, so that doctors can obtain patient data as quickly as possible for example, or obtain recommendations for treatments for their patients, and these are just a couple of the many examples for this case. Once Beam has analyzed the data and the tasks have been scheduled, data is then pipelined to the various edge/fog nodes, as well as the cloud, and in each node, data is analyzed and tasks are executed with the aid of Flink based on their priority deadline. Once completed, data is sent to the cloud for final analysis, while also sent to a distributed storage in case of errors or faults.

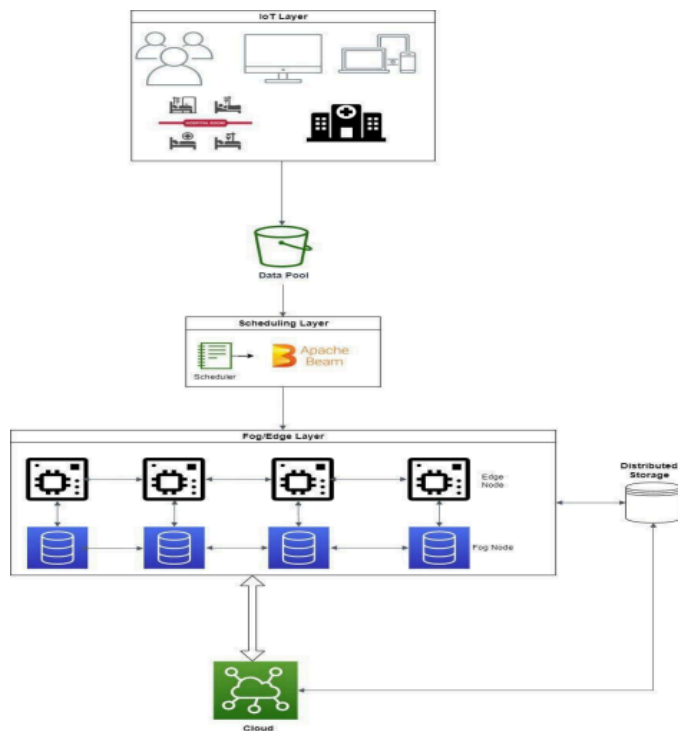


Figure 1. Overview of the proposed system model

Internally, each edge and fog node receives their assigned tasks and data stream segment from the scheduling layer, which are received by the messaging queue, whose job is to receive the assigned tasks from Beam. The queue then relays the information to the processor of the edge/fog node, which in this particular instance, is Apache Flink. Flink receives the information

and begins to process the data, while communicating with the local resource manager regarding the allocation of resources for each of the assigned tasks, as seen in Figure 2. Once analysis of the data and the submitted tasks are completed, Flink send the data back to the messaging queue, which maintains constant communication with both Flink and the resource manager, who then relays this to any other edge or fog nodes, as well as the cloud, while also saving a copy of this information in the distributed storage. Each node will come with its own storage where information will constantly be stored in case of an error or unexpected situation, but its inclusion is optional, and can be removed with little to no consequence on the system.

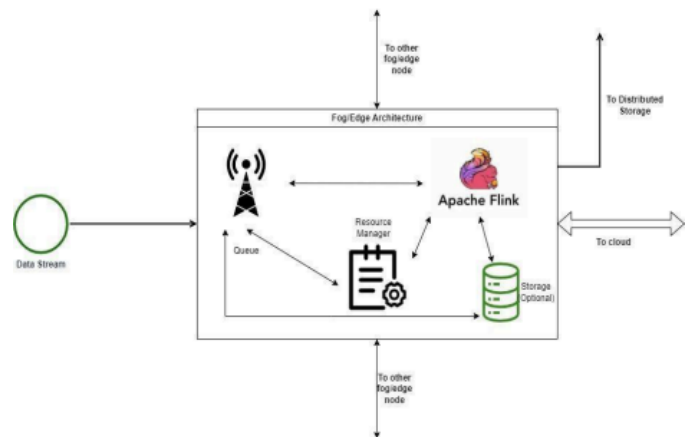


Figure 2. Internal view of what occurs inside each edge or fog node device

C. Scheduling Design

With regards to the scheduler, there was a debate between the usage of a static scheduler and pre-scheduling the tasks to make it work according to our specifications, or the use of a dynamic scheduler that will prioritize the tasks deadline while running, and that is just one of the many factors that have to be considered for the creation of a unique and worthy scheduler, including:

- Fog and edge nodes have different processing capabilities and loads, so a scheduler must take that into account when assigning or delegating tasks to each node
- Depending on the priority level of certain tasks, they might have to be executed at different execution times or within a specific timeframe, and typically, must also be completed within a set or specified period
- Given the variety and complexity differences surrounding all potential tasks, different resources must be allocated by the scheduler, which would also determine the node said task would be sent to
- One major point to consider is that nodes should be able to push tasks onto another node that is free at the moment in case the node in question is handling a sizable number of tasks. Likewise, a node that has no

or a limited number of tasks assigned should be able to pull tasks from other nodes that are dealing with several at a time, taking into consideration the processing capabilities of said node

- One major point from [8] that should also be considered in a scheduler, particularly one that involves fog and edge nodes, is the consideration of MIPS

IV. EXPERIMENTAL PARAMETERS

A. Dataset

At the moment, I am currently using open source medical records from OpenEMR, a free to use software that offers a variety of different medical databases for use in different scenarios (<https://www.open-emr.org/>). The datasets themselves have been simplified for the most part, but there are modules that can be added to offer different scenarios or complications (like urgent ER situations and the like) as required through testing. Primary fields for use in simulation include the more relevant patient details like name, prior medical history, current condition, current physician diagnosis, and current treatment if applicable, among others. Other open source datasets, as well as data synthesized in house are also in consideration for usage when needed. As mentioned earlier, the next goal if successful is to collaborate with a hospital or clinic to provide a more accurate and realistic dataset for testing, but that is for future consideration.

B. Simulation Platform

All simulations will be run on the ubuntu virtual machine, utilizing the aforementioned ifogsim to simulate fog and edge nodes in action. I've spoken about the details of ifogsim in a previous section, but its use for the purpose of this project is to provide us with a platform that can simulate several nodes at once, essential to testing or simulating a more small-scale healthcare system, and can simulate both edge and fog nodes. It also allows us to test our system against a number of key metrics like latency, network usage/bandwidth and, crucial to the future of this system, cost and complexity.

Physically, work will be done using both the computers available in the DSMP lab as well as my own personal laptop when working remotely, though primarily, work will be done on the latter. Both systems have a Windows 10 operating system, and an Intel i7 processor. Programming will be done using Java and Python, Java for the frameworks and Python for the nodes, though the idea is to potentially stick to Python for flexibility and ease-of-use.

C. Experimental Results and Future Outcomes

As of now, this is still a work in progress, and testing is being conducted. For the final camera-ready version and presentation, results will be present.

V. CONCLUSION

Healthcare systems are and will continue to be a relevant topic of interest for governments and developers for years to

come, especially as technology continues to progress, and smart systems in particular are going to be the way of the future in that regard. The idea of merging fog computing and edge computing have a number of clearly defined benefits to doing so, as is the case with the addition of a scheduler and the addition of Apache Beam and Flink. What hasn't been accomplished, however, is the combination and adoption of several of these areas into one unique and compact idea that I believe has the potential to bring significant change to the industry, and while there are a lot of factors and components to figure out, I plan to test the scalability of this system through running simulations on ifogsim.

The ideas mentioned above are of course only the beginning of the road. In future developments, I plan to utilize machine learning to help simplify and automate the process, which is especially critical for the development of smart systems. Initial research and design suggests that deep learning might be the ideal method to integrate with this system, but more will become clear as the project moves forward. As mentioned earlier, a dynamic version of the scheduler will eventually be implemented to aid in this endeavor, constructed by using deep learning or other AI techniques like reinforcement learning to further increase the scalability of the system. Once testing has yielded some consistent results, one of the end goals of this work is to secure the aid of a hospital or major clinic to provide more accurate and realistic datasets for testing, and if that fails, then hopefully volunteers will be brought in to help with the experiment. It is my biggest hope that this project can get off the ground, and provide both a landmark in the field, as well as a platform that can be built upon by others in the future.

By applying and completing the following goals, this project should be helpful to all manner of healthcare providers/stakeholders and potentially even save lives along the way. The ultimate goal is to make life easier for doctors and patients primarily and provide a comprehensive system that not only builds upon some of the features that have been designed in the past but also to provide a benchmark for future healthcare projects that can be followed upon in future developments.

ACKNOWLEDGMENT

I would like to acknowledge the aid of the Distributed and Cloud Computing Lab at Toronto Metropolitan University for providing me with the resources required to design and test the ideas highlighted in this work. I would also like to acknowledge the CS department at TMU for their valuable aid and support.

REFERENCES

- [1] Hassan, S. R., Ahmad, I., Ahmad, S., Alfaify, A., & Shafiq, M., *Remote pain monitoring using fog computing for e-healthcare: An efficient architecture*, Sensors, MDPI, vol. 20, no. 22, p. 6574, Available <https://doi.org/10.3390/s20226574>, 2020
- [2] Tuli, S., Basumatary, N., Gill, S., Kahani, M., Arya, R., Wander, G. and Buyya, R., *HealthFog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments*, Future Generation

- Computer Systems, Elsevier, vol. 104, pp.187-200. Available: <https://arxiv.org/ftp/arxiv/papers/1911/1911.06633.pdf>, 2020
- [3] Gupta, H., Dastjerdi, A. V., Ghosh, S. K., and Buyya, R., *IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments*, Software: Practice and Experience, Wiley Online Journal vol. 47, no. 9, pp. 1275–1296, 2017
- [4] Mahmud, R., Paliewtata, S., Goudarzi, M., & Buyya, R., *IFOGSIM2: An extended ifogsim simulator for mobility*, Arxiv, Cornell University. Available: <https://arxiv.org/pdf/2109.05636v2>, Nov.24, 2021.
- [5] Apache, *Apache Beam Overview*, Available: <https://beam.apache.org/get-started/beam-overview/>, 2021
- [6] Sendorek, J., Szydlo, T., Windak, M., and Brzoza-Woch, R., *FogFlow - computation organization for heterogeneous fog computing environments*, Lecture Notes in Computer Science, Springer Science + Business, pp. 634–647, 2019
- [7] Jamil, B., Shojafar, M., Ahmed, I., Ullah, A., Munir, K., and Ijaz, H., *A job scheduling algorithm for delay and performance optimization in Fog Computing*, Concurrency and Computation: Practice and Experience, Wiley Online Journals, vol. 32, no. 7, 2019
- [8] Alsmadi, A. M., Ali, R. M., Aloglah, N. J., Abu-darwish, A., Smadi, A., Alshabanah, M., Alrajhi, D., Alkhaldi, H., and Alsmadi, M. K., *Fog computing scheduling algorithm for smart city*, International Journal of Electrical and Computer Engineering (IJECE), vol. 11, no. 3, p. 2219, 2021
- [9] Verma, M., Bhardwaj, N., and Yadav, A. K., *Real time efficient scheduling algorithm for load balancing in fog computing environment*, International Journal of Information Technology and Computer Science, MECS Press, vol. 8, no. 4, pp. 1–10, 2016
- [10] Aladwani, T., *Scheduling IOT healthcare tasks in fog computing based on their importance*, Procedia Computer Science, Science Direct, vol. 163, pp. 560–569, 2019
- [11] DataFlair Team, *Why Apache Flink - Best Guide for apache flink features*, Available: <https://data-flair.training/blogs/apache-flink-features-why-flink/>, 2021
- [12] Khiati, R., Hanif, M., Lee, C., *Adapting Distributed Stream Processing Technologies for the Automation of Modern Healthcare Systems*, The 22nd International Conference on Advanced Communications Technology, 2020
- [13] Battulga, D., Miorandi, D., and Tedeschi, C., *Fogguru: A fog computing platform based on Apache Flink*, 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2023

Rhaed Khiati is a PhD candidate in the department of computer science at Toronto Metropolitan University. He earned his Masters degree at Hanyang University in Seoul, South Korea in 2020. His research interests include Smart Healthcare Systems, Cloud Computing, Edge and Fog Computing, and Distributed Streaming Optimization. His email address is: rkhiati@torontomu.ca.