Towards the Optimal Placement of Containerized Applications in a Cloud-Edge Model

James Nelson, Jonathan Lwowski, Patrick Benavidez, John J. Prevost, and Mo Jamshidi
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249

Abstract—Cloud computing is now a global standard computing topology and has been widely studied for many years. Less frequently researched is the use of cloud and edge computing to optimize the performance of a system as a whole. One important aspect of cloud and edge computing is managing the placement of the applications in the network system so as to optimize each application’s runtime, given the resources of system’s devices and the capabilities of the system’s network. The properties of containerized applications now make this possible. The process of containerization creates a lightweight, mobile, packaged application for each of the algorithms of a system. These applications can then be easily and quickly deployed on any layer of the cloud and edge computing architectures. In this research, a fuzzy placement control system is designed to place applications on the cloud-edge model. As verified from simulated results, our fuzzy placement control system reduces the total runtime of the applications by placing applications near their optimal location.

Index Terms—cloud computing, edge computing, fuzzy control, containerization, expert systems

I. INTRODUCTION

Cloud computing is the use of computational services such as servers, storage, or analytics over a network such as the internet [1]–[5]. These services exist mainly in two categories of the cloud: private cloud, or public cloud. Private cloud is a cloud that is owned and managed by the user using open-source cloud software such as OpenStack [6]. Public cloud is owned and managed by a private company that charges the customer based on data usage, for example Amazon’s AWS or Microsoft’s Azure [7], [8]. Since cloud architectures are highly scalable, they are considered a source of infinite computational power. Also, because these cloud systems are so powerful, they are being widely researched in academia while also being employed as a commercial tool in industry [9]. Typically, cloud computing is used to analyze large amounts of data, but cloud computing can also be used to interconnect systems, allowing them to act as one system [10]–[13].

Although cloud computing has virtually infinite compute power, there are some disadvantages to using the cloud. One of these disadvantages is the increased latency associated with increased internet traffic [14]. Since cloud servers can be thousands of miles away, it can take a relatively long time to send the data to the cloud server for analysis. In many applications this is not an issue, but in fields such as swarm robotics this increased latency can be detrimental.

To solve some of the problems associated with cloud computing, edge computing emerged. Edge computing is used to improve the speed at which a cloud system can respond to data [15]–[18]. Edge computing takes applications that would otherwise be hosted on the cloud and moves them to edge devices, such as routers or network switches. This is done to decrease network latency and to lower network traffic. An example of an application of edge computing is “internet of things” (IoT) systems. In IoT systems, the local router may be unable to send large amounts of data to the cloud system. Instead of the router sending the information to the cloud, the router will perform some computation locally, which can significantly reduce network latency [19]. Just like cloud computing, edge computing also has some drawbacks that limit its abilities in real world systems [20]. Most edge and end devices typically don’t have a lot of computational power. Therefore, the end and edge devices can only run relatively simple applications. This limits which applications can be placed on the edge devices, for this lack of processing power can cause many applications to run significantly slower on the edge device than on the cloud.

Since both cloud and edge computing have benefits and drawbacks, many researchers have started to combine them into one system. The combination of cloud and edge computing, the cloud-edge model, takes advantage of the computational power of the cloud by placing computationally expensive applications on the cloud, while placing computationally inexpensive
applications on the edge device. A less researched topic is the development of a mechanism that automatically decides the optimal location for an application in this model. There are many factors that need to be considered when deciding where to place an application in the cloud-edge model such as computational power of edge device, or the computational expense of the application. A few people have tried to solve this problem. For example, Taneja and Davy used iFogSim to demonstrate that, in some cases, the adoption of the fog computing paradigm can reduce energy consumption and network traffic [21]. Also, Tran. et. al used optimization methods to maximize the amount of applications on the edge devices while keeping all of the applications’ delays under a certain value [22]. Their results are promising, and although they consider transportation systems, there is little implication for more short-run dynamic and stochastic systems [22]. Additionally, Faticanti et al. also attempted to solve this problem by using mixed-integer nonlinear programming (MINLP) [23]. This MINLP is used to maximize the profit the edge-infrastructure owner will make. Their method maximizes the profit of the edge-infrastructure owner, but their algorithm has a runtime complexity of $O(U^2K^3)$, where $U$ is the number of applications, and $K$ is the number of regions in the cloud-edge model. This runtime can be very slow with large numbers of applications, which can make it unusable in environments that are changing very quickly.

Because of the multitude of network configurations, and consequent combinatorial explosion, the cloud-edge model has issues with optimization at scale. Randomness and non-negligible exogenous shocks further complicate the analysis, but more importantly, are not known a priori, and typically can only be classified post hoc. Accordingly, a fuzzy logic application placement controller is developed, for it does not depend on a model.

The rest of the paper is structured as follows. Section II describes the cloud-edge model, along with the necessary considerations for deciding where to place an application in the cloud-edge model. The following section, Section III, describes our proposed fuzzy placement control system that decides where to place applications in the cloud-edge model. The results are then discussed in Section V. Finally, our conclusions and future work are discussed in Section VI.

II. CLOUD-EDGE MODEL

A. Modeling the Cloud-Edge Network

Each node on the network has to consider a multitude of factors in generating an estimate for the time it will spend or save by moving the application to that node. Some of these factors include the device’s local network usage, processing power, random access memory (RAM), and load management or operational overhead processes. While many of these parameters may be selected when specifying the machine running on the cloud, the operational overhead and local net usage are, in general, not available. Further, due to short-term fluctuations in these parameters, their values need to be estimated, and these estimators necessarily must be conditioned on the knowledge set of the predictor. As a result, estimation will not condition on confounding variables such as, for example, the emergence of aggressor nodes in the local network.

B. Cloud-Edge Model Considerations

There are a few things that need to be considered when choosing the optimal locations to place an application in the cloud-edge model:

1) Computational Power of Edge Device: The first thing that needs to be considered is the computational power of the edge device. Currently, the processing power of edge devices are increasing, but still are not nearly as powerful as small micro-computers, such as the Raspberry Pi 3. For example, the Linksys WRT AC1200 router is currently one of the most computationally powerful routers, but it only has two 2.0GHz processors, and 512MB of RAM [24]. This severely limits the number of applications, and the computational need of an individual application that can be place on the edge.

2) Networking Parameters: There are a couple of factors that affect the speed at which information can be sent from end devices to the edge devices, and to the cloud. These parameters include network bandwidth, and network latency, neither of which are known a priori and therefore must be predicted. Seasonal Autoregressive Integrated Moving Average (SARIMA) models can be used for forecasting how bandwidth changes over time; however, by construction, these models do not condition on acyclical exogenous shocks to network demand [25]. As such, fluctuations of usage of the cloud or fog during the day among other acyclical shocks are unaccounted for in the predictive modelling of these parameters.

3) Intra-Application Communication: The usage of the network is not strictly for outsourcing computations. In many cases, the application manages the communication among devices on the local network. When this is the case, there is no added time due to network delay, for the relevant messages still must be sent over the network in the counterfactual. Therefore, if an edge device has sufficient compute power, it may be more expedient to put some tasks on that device.

4) Application’s Communication Load: The final consideration when choosing where to place an application is the communication load the application will have on the network. For example, if an application needs to send large images to an end device, then it may be beneficial to place that application on the edge device.
This is because sending images from the cloud to the router then to the end device may take a long time. However, if the application is sending small packets of data, then placing the application in the cloud may not be an issue, for it may only take a very small amount of time to send the small packets of data from the cloud to the end device. Further, if the burden the application imposes on the network is non-negligible, then this creates a second-order effect that complicates the predictive capacity of the aforementioned SARIMA models.

III. Fuzzy Placement Control System

Our proposed fuzzy application placement system, as seen in Figure 1, consists of three subsystems. The first two subsystems are used to estimate the reduction in runtime by placing the application on the cloud and the reduction in runtime by placing the application on the edge device. The first two subsystems run in parallel and feed into the third subsystem to give the estimation where to place the application.

Fig. 1: Overview of the fuzzy placement control system

A. Cloud Runtime Reduction Estimator

The first subsystem, the cloud runtime reduction estimator, is used to estimate how much time is saved by placing the application on the cloud, excluding communication time. The inputs of this subsystem are the estimated speedup of placing the application on the cloud, and the magnitude of the runtime of the application. The speedup is used to estimate the performance increase of placing the application on the cloud relative to placing the application on the end device. In determining the speedup, both the effects of parallelization and the difference in hardware capabilities are considered. The membership function for speedup can be seen in Figure 2.

By combining speedup with the runtime of the application, this subsystem can estimate how much time will be saved by placing the app on the cloud. The membership function for the output of this subsystem can be seen in Figure 4.

B. Edge Runtime Reduction Estimator

The second subsystem, the edge runtime reduction estimator, is used to estimate how much time is saved by placing the application on the edge, excluding communication time. This subsystem works the same as the cloud runtime reduction estimator, but the speedup is the speedup from placing the application on the edge instead of the cloud. This allows the subsystem to estimate how
much time will be saved by placing the application on the edge.

C. Container Placement Estimator

The last subsystem, the container placement estimator, uses the output of both the cloud and edge runtime reduction estimators, along with various networking parameters such as latency, bandwidth, throughput, and if it requires information from other applications or devices. The cloud and edge runtime reduction estimators allow the container placement estimator to predict how much time will be saved by placing the application on the edge or the cloud. The networking parameters will allow the containerized application placement controller to predict how much time it will take for the information needed by the application to be sent across the network. The current network latency allows the system to know how long it will take to send information through the network, and the bandwidth will allow the system to know how much information can be sent through the network in a given time. The application’s network throughput will allow the system to know the rate of the information being sent and received by the application. Finally the container placement estimator needs to know if the application needs information from other applications. This is important because if the application needs either to only send or only receive information to or from multiple devices, and if the application is placed on the edge device, the latency of collecting the data from the other applications or devices will be reduced. This is because placing the application on an end device elongates the application loop by forcing the data to be collected at the router, then sent to that end device for processing, thereby introducing delays from network latency and bandwidth limitations. By using all of the information described above, the container placement estimator will estimate the best location to place the application in the cloud-edge model.

IV. Simulation

A. Simulation Environment

The performance of this application placement controller was evaluated using Yet Another Fog Simulator (YAFS). YAFS is a highly configurable, Python-based simulator for cloud, edge and fog computing [26]. Using conventions from graph theory, each computational device in the network is represented by a node, and each connection between two computational devices is represented by a link. For the intents and purposes of this research, the network is modelled as an undirected graph, and the two end devices are identical except for the sensors and actuators they may contain.

Our network was divided into three regions: cloud, edge, and end. For each region, the number of constituent nodes was determined. In our case, this was one cloud node, one edge device node, and two end device nodes. The network is connected as shown above in Figure 5, where node 0 is on the cloud, node 1 is an edge device and nodes 2 and 3 are end devices. For each node, the amount of instructions that can be executed per second (IPT) and random access memory (RAM) were set; for each link, the bandwidth (BW) and base latency (PR) were set. Each app has a workload defined by the number of instructions to be executed (INST) and the size of the messages (M) to be sent to and from the application.

B. Calculating Optimal Placements for Applications

This app placer discussed is mainly designed for multi-agent robotics, and as such, the valued criterion was reduction of the overall loop time. The loop time is defined as the total amount of time it takes for the application to receive data from the sensor on an end device, process that data, and send the results back to the actuator on an end device. The communication delay from device $i$ to device $j$, $t_{ij}$, is calculated using Equation 1, given the size of the transmissions $M_k$ to and from application $k$. $R_{ik}$, the runtime of application $k$ on device $i$ is given by Equation 2 [26]. The full loop
delay or round-trip time $T_{loop}$ is defined by the sum of the link delays $t_{mnk}$ on the transmit path $P_t$, the link delays on the receive path $P_r$, and application runtime on device $i R_{ik}$, as shown in Equation 3.

$$t_{ijk} = \frac{M_k}{BW_{ij}} + PR_{ij}$$

$$R_{ik} = \frac{INST_k}{IPT_i}$$

$$T_{loop} = \sum_{(m,n) \in P_t} t_{mnk} + \sum_{(m,n) \in P_r} t_{mnk} + R_{ik}$$

C. Relating Simulation and Fuzzy Placement Controller

While not all the input parameters for the fuzzy placement controller directly map into this simulator, the simulation parameters can be obtained by, at most, a first order derivation and mild simplifying assumptions. The parameters given to YAFS were first generated as inputs to the fuzzy placement controller, and then the analogous parameters were mapped into YAFS. For the purposes of this research, cloud speedup is determined by the product of both the hardware speedup and the speedup from parallelization.

$$s_i = \frac{IPT_i}{IPT_2}$$

The cloud speedup $s_0$ and the edge speedup $s_1$ are determined by the ratios of the devices’ $IPT$ values and those of the end device, as shown in Equation 4. As, previously stated, the runtime of application $k$ on device $i$, $R_{ik}$ is given by Equation 2, and because the speedup parameters were calculated with respect to the end device, the runtime that the placement controller is interested in is the runtime when device $i$ is the end device. The latency parameter used was the base latency between the cloud and edge nodes $PR_{01}$, for this represents the maximum base latency in the network. The bandwidth used was the bandwidth between the edge and end nodes $BW_{12}$, because the connection between the edge and the end device is set to have the smallest bandwidth in the network. The throughput is a realizable upper bound on the bit rate of the messages being sent and scales the bandwidth parameters of the links accordingly. The input parameter entitled “Requires Other’s Info” requires a slight modification to Figure 5. If this value is true, then the actuator on node 2 is moved to the other end device, node 3, thereby forcing communication through the edge device. Because of our treatment of the end devices, this is equivalent to moving the sensor from node 2 to node 3 by symmetry.

V. Results

The inputs to the fuzzy placement controller were generated by scaling and shifting standard uniform random variables such that they would only span across the universes of the membership plots that were explicated in the placement controller algorithm. This was done so as to bias the test cases to be closer to the boundary conditions of the decisions made by the placement controller. In effect, this is biased in order to produce a more conservative estimate of the placement controller’s performance. Moreover, the addition of this bias also is more reflective of the cases where this placement controller is more likely to be employed, for, in more extreme cases, the optimal placement decision will tend to be more obvious to the user, and these test cases were intended to test the performance of the controller in the conditions where the correct decision is less obvious.

For each configuration, all placements were tested, and the placement with the lowest value for $t_{loop}$ was selected as the optimal or “expert” decision. Once the optimal placement was found, the network parameters were put into the fuzzy placement controller. The $t_{loop}$ for the controller’s placement decision and its percent difference from the $t_{loop}$ of the optimal location were then calculated and grouped in 20% intervals for visualization, as shown below in Figure 6.

![Fig. 6: Test Case Error Margins by Percent Intervals](image)

In this more conservative performance estimate, the fuzzy placement controller placed the containerized application on the optimal device 64% of the time, and the placement took no more than 20% more time than optimal in 80% of the trials. For a more concrete example, suppose there were 100 trials, and for each trial, the network was configured randomly but in such a way that the total loop delay for the optimal placement was 10 seconds. It would be the case that, for about 80 of the 100 trials, the placement decision would result in a runtime that was less than 12 seconds. From all the trials, the average percent error was 12.4%. This implies that the runtime of the applications placed on the network according only to the controller is empirically expected to be 12.4% longer than the true optimal values.
VI. CONCLUSION

In this paper, a fuzzy logic application placement controller was developed for placing containerized applications on the cloud-edge network. Due to the nature of this network, models for the system are either not well explicated, or do not scale to an operational level. Accordingly, a fuzzy logic controller was chosen because of its ability to work in the absence of an explicit model of the system. The fuzzy logic controller takes in estimates of the various network parameters, and returns a decision of where the application should be placed on this network to minimize the application’s total loop delay.

Though using this fuzzy logic-based application placement controller is able to provide a rough classification of where to place apps on a multi-agent network, much can be done in the way of improving it. Machine learning can be used to tune the membership functions of the application placement controller through the use of the simulated annealing algorithm or the genetic algorithm. Alternatively, the use of neural networks and adversarial reinforcement learning is under investigation as an alternative to this fuzzy logic-based placement algorithm.

REFERENCES