Continuum on ARM

Felix Goosens
Vrije University Amsterdam
Amsterdam, Netherlands
f.goosens@student.vu.nl

Abstract

Continuum attempts to unify edge compute like models to create an easier and more effective solution for developers. In order to come close to these goals, we evaluate Continuum in a more realistic scenario than was done previously.

To achieve this, we developed a design model for evaluating Continuum on a cluster of resource constrained devices. Our design also provides an easy to manage and robust storage solution.

We also implement ARM support in Continuum to increase its compatibility and evaluate Continuum on a cluster of ARM devices. This has exposed some issues with the current implementation of Continuum that were not previously known. Additionally, our results have shown similar trends that were present on the evaluation of Continuum on emulated hardware. These results therefore show that many trends can be identified by evaluating a computing framework on emulated hardware.

1 Introduction

Cloud computing is a popular computing model where the processing of data is offloaded to a cloud service. However, due to the popularity of cloud computing, network congestion has become an increasing problem. To counter this, edge computing is used where small processing tasks are offloaded to a server in close proximity to the data source.

However, to accommodate for the need of different use cases, many edge computing like models exist. This makes it harder for developers to decide which compute model fit their needs best and causes confusion about the notion of edge computing.

Continuum addresses this problem by unifying edge compute-like models into one general framework [1]. By creating a unified architecture, Continuum aims to reduce complexity for developers while providing the required characteristics for the application. Continuum, however, has not been evaluated on small devices which are typically used for edge computing. Additionally, it has no support for the ARM architecture, which is a common ISA for small devices.

We aim to evaluate Continuum in a more realistic scenario by using resource constrained ARM devices. By using Continuum on real hardware, we found some problems and behavior that was not present in an emulated setting. This also allows us to analyze the effectiveness of performance benchmarking on emulated hardware for Continuum.

To accomplish this, we add the functionality for Continuum to also support ARM and perform similar benchmarks that were done in the original paper. We also use network storage, so we can easily change our filesystems and do not have to rely on potentially unreliable storage on our ARM devices.

Our contributions can be summarized as followed:

1. We showcase a design that allows anyone to evaluate Continuum on resource constrained ARM devices with easy to manage and reliable storage.
2. We add ARM support on Continuum for its edge and endpoint nodes.
3. During development and experimentation, we discovered some issues that were not obvious before. We also give some further insights on how to potentially solve these issues to improve Continuum further.
4. We evaluate Continuum on resource constrained hardware that Continuum was designed for. We also compare our results with the results found on the original paper and discuss similarities and differences.

2 Background

2.1 Edge Computing

Edge computing is a computing model where the node is designed to reduce the high network costs needed for cloud computing [2]. The core idea is to lower the amount of data sent to the cloud by performing simple tasks near the source of the data instead of in the cloud. More complicated tasks can still be done by the cloud and the cloud can also act as a centralized server. To accomplish this, edge computing uses a three layered structure.

The first layer is the terminal layer, which are devices at the very end of the network and typically generate data that needs to be processed, such as sensors or cameras. These devices have very limited resources and thus typically offload their data straight to the second layer without processing it.

The second layer is the boundary layer and it is this layer that enables edge computing to reduce the overall network load. Devices on this layer are responsible for collecting and processing the generated data from the terminal layer and to forward data to the cloud for additional processing. Typical boundary devices can be access points, routers and switches. To reduce the network load, these devices must be geographically close to the devices of the terminal layer.

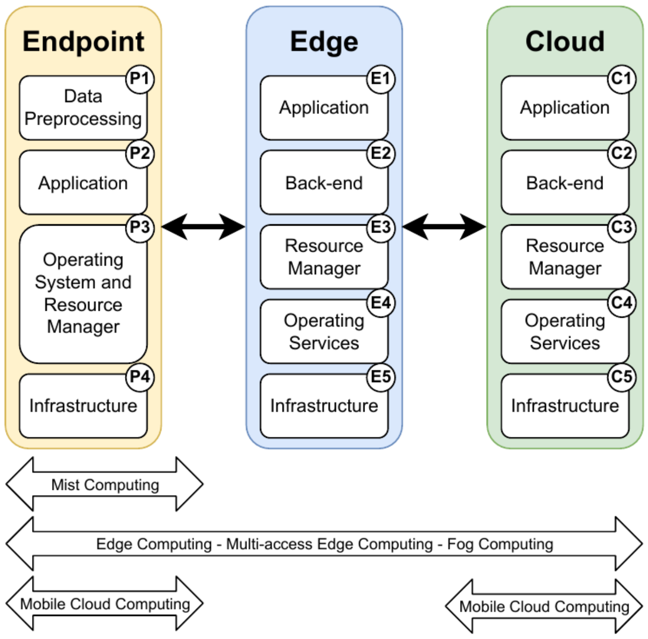

Figure 1. Reference architecture for Continuum. The computing models below show how they span across different components.

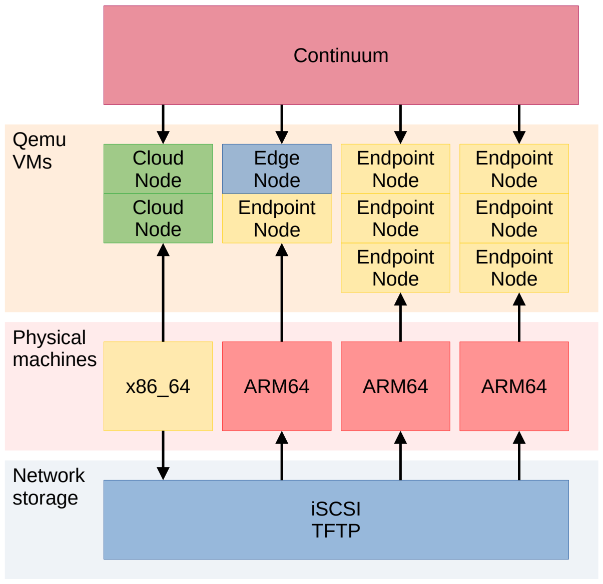

Figure 2. Our design for evaluting Continuum on ARM devices.

The third layer is the cloud layer, which consists of one or more cloud providers. While the first two layers contain inexpensive devices that are in close proximity with each other to reduce network load, the cloud providers in this layer have much more computing resources and can be located far away from terminal and boundary devices.

Using these principles, it is possible to reduce the latency and high long distance bandwidth requirements at the cost of a small amount of extra compute power.

2.2 Continuum

Edge computing is not the only model that combines local processing with cloud computing to reduce the network load. Other models that use this idea have also been made with different pros and cons. Some models focus on working with a high latency or utilize a Peer-to-peer network. However, choosing which model to use can be a daunting task since many applications do not fit exactly into the use case of a specific model.

The Continuum framework attempts to summarize some of these models into one general solution. The framework can be configured to use different aspects of each model, allowing developers to pick and choose different pros and cons across computing models.

Figure 1 represents an overview of the unified architecture that Continuum is based on. Similar to edge computing, Continuum uses three different kinds of compute nodes, endpoints, edge and cloud nodes. The endpoint is similar to the terminal layer in edge computing and consists of devices at the end of the network which typically generates the data that needs to be processed. The edge works like the boundary layer and its responsibility is to collect, process and forward data generated by the endpoints. The cloud is like the cloud layer and represents compute nodes with a lot of resources but may be located far away from the endpoint.

Since Continuum can be configured to fit the needs of different applications, developers do not have to decide on different compute models but can simply specify their needs. This simplifies the developing process and also allows Continuum to better fit the needs of the application.

3 Design

The design of our setup is depicted on figure 2. Our physical machines consist of systems with a x86_64 or an ARM64 ISA. They are all connected on a network and use QEMU with KVM to run their virtual machines.

Continuum has access to every machine and is responsible for starting the virtual machines, installing the required software, and running the benchmarks on them. Continuum also schedules how many cloud, edge and endpoint nodes a machine has to run.

3.1 Network Storage

Instead of each machine having their storage locally available to them, we use storage network protocols to provide the boot and root partition to other machines. In our case, the x86_64 machine holds the filesystems of every ARM64 machine.

We distribute our storage using TFTP and iSCSI. We use TFTP to provide the boot files for our ARM64 machines because it is supported by the default firmware for the raspberry pi 4 b's, which we use as our ARM64 machines. iSCSI is our protocol of choice for distributing the root filesystem due to it's excellent docker support.

With this configuration, we can manage all of our storage from one machine and can easily backup, swap and modify the filesystems for our ARM64 machines. This also means that we do not have to use unreliable storage solutions, such as low quality sd cards for our raspberry pi's.

3.2 Scheduler

Originally, the Continuum framework uses a simple scheduler that automatically allocates the requested nodes to the available physical machines. This process cannot be configured, meaning that it is not possible to specify on which physical machine a certain node should be running. Since we need to run all our edge and endpoint nodes on ARM64 machines and our cloud nodes on a x86_64 machine, this limitation must be addressed.

To solve this issue, we extended the capabilities of the configuration file by also adding the option to specify the nodes for each physical machine. Our configuration file from figure 2 is shown in Appendix A. When the custom_scheduling flag is set to True, each physical machine is allocated a number of different nodes.

4 Continuum on ARM

Here we will discuss some of the issues that had to be resolved while adding support for ARM on the Continuum framework.

Network interface name. Continuum uses Ubuntu-20.04 LTS for its virtual machines. Canonical provides both x86_64 and ARM64 cloud images for this Ubuntu version. However, the default network interface name differs between these two architectures. On x86_64 it is "ens2" and on ARM64 it is "enp2s1"

Since "ens2" is hardcoded in the Continuum framework, this causes issues when generating libvirt configuration files and when configuring the network on the virtual machine. This issue had to be solved by checking the architecture of the machine and changing the network interface name that Continuum uses accordingly.

libvirt ARM64 configuration. The libvirt configuration files that libvirt uses had to be slightly modified to support ARM64. Besides the network interface name change, the aarch64 architecture had to be specified and the cpu host-passthrough mode had to be explicitly stated to enable kvm. Also, an EFI loader had to be specified as well. Fortunately, Qemu comes with EFI firmware that is used by default.

support for both x86_64 and ARM64 docker containers. Continuum uses a docker registry that holds the docker images that will be used by other nodes. However, supporting both x86_64 and ARM64 versions of a docker image on a docker registry is not trivial. In order to distinguish an x86_64 image from an ARM64 image with the same name, a manifest has to be present in the registry that specifies the corresponding platform for each container. An unfortunate downside is that the 'docker manifest' command is experimental and may change without warning [3].

Arch recognition while building netperf. Continuum uses NetPerf to emulate and evaluate networks with certain properties, such as the latency and throughput. By default NetPerf is built with an outdated config.guess script. This causes the building process of NetPerf to fail on ARM64 because it is not able to recognize the platform. To solve this issue, we download the newest version of config.guess and overwrite the old script just before building NetPerf.

5 Experimental setup

For our setup, our x86_64 machine has an intel core i7 870 CPU and 8GB of memory. We are using Debian 11 (bullseye), Qemu and kvm version 5.2.0.

Our ARM devices consist of three raspberry pi's model 4b. We use Ubuntu 20.04.4 LTS (focal), Qemu and kvm version 4.2.1. They use an BCM2835 SoC with 4 cores and 4GB of memory.

5.1 Issues with continuum

While benchmarking Continuum, we experienced some issues. These problems lower the stability and performance of Continuum. However, these issues do not present limitations of the design of Continuum and it should be able to rectify them without significant drawbacks.

Ubuntu auto upgrades. Ubuntu-20.04 LTS has an automatic update service named unattended-upgrades enabled by default. However, since this service can prevent the user from installing packages, this can interfere with the setup of the VMs, causing Continuum to crash. This problem can be solved by either disabling the automatic update service as early as possible or by using another distro that does not ship with an automatic update service by default.

Ram requirements. Continuum uses 1 GB of ram for each vm per cpu core, including on edge and endpoints. However, small ARM devices are usually limited in ram and this requirement can be steep.

Due to our lack of ram, we were unable to run 4 nodes on a raspberry pi's, despite it having 4 physical cpu cores. Even running 3 nodes may trigger the OOM killer to terminate one of the running vms during an experiment.

However, this steep ram requirement may not always be necessary. We have found that reducing the ram requirement for endpoint nodes to 800MB per cpu core causes a considerable increase in reliability when running 3 vms on our raspberry pi's. Further efforts in reducing the ram requirement of Continuum vm's would make it more reliable to run on small ARM devices. Additionally, if the ram requirement can be significantly reduced, it may also be possible to run 4 vms on our raspberry pi's.

Faster setup. Currently, Continuum has a setup phase where it configures the base vms before running the experiments. This setup phase only has to be done once, which lowers the time it takes to run multiple iterations of an experiment.

However, while the experiment itself can finish after 5 minutes, an iteration can still take a long time to finish. This is because, while the setup phase does help, there is still a significant amount of configuration to be done after the setup phase. This remaining part has to be done every iteration and takes the vast majority of time. We recommend optimizing the setup, which can be done by including more into the setup phase for the base vms or by reusing already running and configured vms from previous experiments.

Lowering the time it takes to perform experiments will make it easier to run more iterations and therefore increase the accuracy of the results. It may also make it easier to test and compare different configurations of Continuum, such as the optimal amount of endpoint nodes per edge node.

6 Evaluation

We evaluate the performance of both cloud computing and edge computing in our setup with a variety of endpoints. We use the same application for data generation and processing as in the Continuum paper. That is, we generate data in the form of images by emulating a camera and process it by performing object detection. We also use MQTT to communicate between two application components.

Just like the Continuum paper, we generate data for 5 minutes per experiment. We also ensure that each machine never emulates more cores for each vm than the amount of physical cores.

Figure 4 shows the results for using cloud and edge computing where each endpoint generates 5 images per second. The x-axis shows the amount of endpoints that were used in each experiment, along with its respective configuration. The y-axis shows the system load in linear scale and end-to-end latency in logarithmic scale. The end-to-end latency is the amount of time between an image being sent from an endpoint and receiving the results, after it has been processed. The system load is the total frequency of the images that are

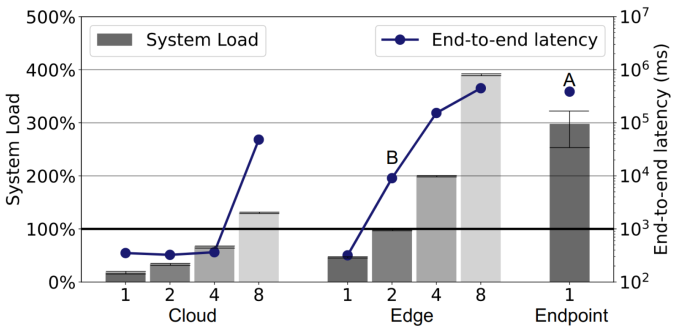

Figure 3. Results from the original Continuum paper, using a frequency of 5 images per second.

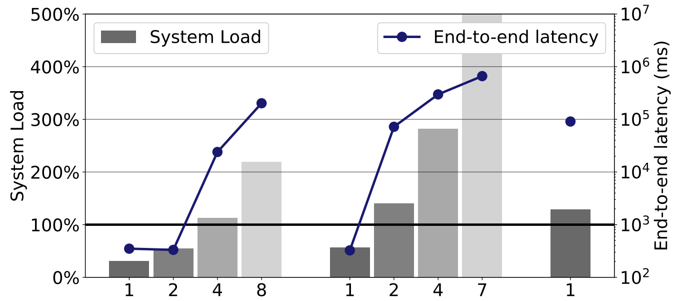

Figure 4. Results from our setup, using a frequency of 5 images per second.

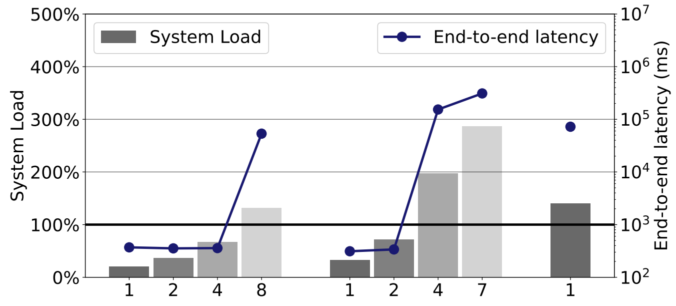

Figure 5. Results from our setup, using a frequency of 3 images per second.

being generated divided by the time it takes to process an image.

We also show the performance of an endpoint processing its own data. Figure 5 shows the same, except that in this case we generate 3 images per second.

Figure 3 shows the results that were presented in the original Continuum paper. These results were made on three Intel Xeon Silver 4210R machines with an image generation frequency of 5 per second. In order to emulate resource constrained devices, they reduced the cpu performance to 66% for edge nodes and to 33% for endpoints.

Results. From the results, we see that the overall performance four our cloud and edge configuration is worse than is shown in the original paper when generating 5 images per second. In this case, our cloud configuration can handle two endpoints. However, when using four endpoints our system load increases beyond 100% and our end-to-end latency skyrockets. As expected, our edge configuration performance worse and we see the same pattern when using two endpoints.

In contrast, the Continuum paper can handle 4 endpoints in their cloud configuration without an increase in latency or going beyond 100% of their system load. Also, while their edge configuration seems to have reached its limit when using 2 endpoints, their latency and system load is significantly lower than our results when using 2 endpoints. However, Our results are not always worse than what was found in the Continuum paper, for our endpoint configuration we perform significantly better than was found in the Continuum paper

When generating 3 images per second, we see a significant performance increase for our cloud and edge configuration. Our edge configuration can comfortably handle 4 endpoints and our edge configuration can handle 2 endpoints without an increase to end-to-end latency or going beyond 100% of system load.

Endpoint configuration. The fact that our endpoint configuration performs significantly better than was found in the Continuum paper, seems to indicate that our ARM machines are performing better as endpoints than the emulated endpoint machines from the Continuum paper. The Continuum paper uses a third of a cpu core to emulate the slower performance for endpoint nodes. Since our results show a large difference in performance, we believe that this measure mostly caused the high system load and end-to-end latency measured from the Continuum paper and that our ARM machines perform better than that.

Cloud and Edge configuration. While our endpoints perform better than was reported in the Continuum paper, our cloud and edge nodes seem to perform worse. This means that the bottleneck must lie within the node that is processing the images. At the same time, the CPU we use for our cloud nodes is significantly older and less powerful than the CPU that was used by the Continuum paper. We therefore believe that our result difference with our cloud configuration is due to less powerful cloud nodes. For the edge configuration then this is likely a similar cause. While a cpu quota for 33% was too low for our ARM machines, a cpu quota of 66% seems much higher than our ARM machines can perform.

Similarities. Our results also show some trends that were found in the original Continuum paper. Just like the Continuum paper, offloading to the cloud gives the lowest system load. We also see that we can determine when the workload becomes too much to perform real-time processing. We can also change the workload based on the image generation frequency to manage more endpoints. Another similarity is that our end-to-end latency increases significantly when the system load goes beyond 100%.

These similarities indicate that performance trends that are discovered on emulated hardware can also be present on real hardware. This result is interesting because we have nodes that perform both better and worse compared to the Continuum paper. Therefore, we argue that using emulated hardware can effectively be used to identify performance trends, although obtaining accurate absolute performance metrics would be much more challenging.

7 Conclusion

We added ARM support for Continuum and developed a design for evaluating Continuum on resource constrained devices. Due to the use of network storage, the storage for our devices is also reliable and easily manageable. We also updated the scheduler for Continuum to provide more control on the allocation of nodes to our machines.

We mentioned some issues that we found while adding ARM support for Continuum and discussed how we solved them. These can help as pointers for other developers who are seeking to provide support of another architecture to Continuum. We also mentioned some issues that we found while evaluating Continuum on ARM devices and discussed possible remedies.

We then evaluated Continuum on resource limited devices and found similar trends in our results. We therefore conclude that benchmarking a system on emulated hardware can show trends in the results that are consistent on real hardware. However, emulated hardware is not ideal for measuring performance in terms of absolute numbers.

References

[1] Jansen, M., Al-Dulaimy, A., Papadopoulos, A. V., Trivedi, A., & Iosup, A. (2022). The SPEC-RG Reference Architecture for the Edge Continuum. arXiv preprint arXiv:2207.04159.
[2] Cao, Keyan, et al. "An overview on edge computing research." IEEE access 8 (2020): 85714-85728.
[3] Docker. "Docker manifest," https://docs.docker.com/engine/reference/commandline/manifest/, accessed: 2022-08-29

A Example configuration

```
[infrastructure]
provider = qemu

infra_only = False

cloud_nodes = 1
edge_nodes = 1
endpoint_nodes = 7

cloud_cores = 4
edge_cores = 2
endpoint_cores = 1
```

```
cloud_quota = 1.0
edge_quota = 1.0
endpoint_quota = 1.0

cpu_pin = False

network_emulation = True
wireless_network_preset = 4g

netperf = False

external_physical_machines = ubuntu@192.168.0.30,
    ubuntu@192.168.0.31,ubuntu@192.168.0.32

custom_scheduling = True
arm_edge = False

[local]
cloud_nodes = 1
edge_nodes = 0
endpoint_nodes = 0

[ubuntu@192.168.0.30]
cloud_nodes = 0
edge_nodes = 0
endpoint_nodes = 3

[ubuntu@192.168.0.31]
cloud_nodes = 0
edge_nodes = 0
endpoint_nodes = 3

[ubuntu@192.168.0.32]
cloud_nodes = 0
edge_nodes = 1
endpoint_nodes = 1

[benchmark]
resource_manager = kubeedge

docker_pull = False
delete = True

application = image_classification
frequency = 3
```