Fluidity: Providing flexible deployment and adaptation policy experimentation for serverless and distributed applications spanning cloud-edge-mobile environments

Foivos Pournaropoulos*,
Alexandros Patras*,
Christos D. Antonopoulos*,
Nikolaos Bellas*, Spyros Lalis*

* *Department of Electrical and Computer Engineering, University of Thessaly, Greece*

ABSTRACT

We introduce Fluidity, a framework enabling the flexible and adaptive deployment of serverless and modular applications in systems comprising cloud, edge, and mobile nodes. Based on a declarative description of application requirements, a custom placement policy, and a formal system infrastructure description, Fluidity plans and executes an initial deployment of application components in the cloud-edge-mobile continuum and performs reconfigurations at runtime based on resource availability and the position of mobile nodes. Fluidity is an enabler for serverless applications, allowing the application developers to focus on the application code itself while abstracting out the infrastructure management. Notably, it permits developers to provide deployment and adaptation policies and switch between them at runtime. Our results show that the core mechanisms of Fluidity can support flexible application execution at a reasonable overhead and experimentation with different deployment policies with minimal effort.
KEYWORDS: Microservices; Flexible Deployment; Runtime adaptation; Edge computing; Serverless computing; Drones

1 Introduction

The proliferation of serverless applications, usually comprised of several microservices, is significantly affecting the operation of modern cloud data centers. This paradigm enables a high degree of flexibility to the application developers who do not have to be concerned about the underlying infrastructure. An increasing number of modern serverless workloads are container-based [dat], allowing effortless deployment and migration. In addition, a plethora of applications are no longer restricted to the cloud but also involve mobile IoT de-

[1]E-mail: {spournar, patras, cda, nbellas, lalis}@uth.gr

vices, such as smartphones and vehicles like cars or even drones. In this case, edge computing can improve application quality of service by moving computations closer to the points where data is produced, while leading to better overall system performance and stability, as it reduces data traffic and resource pressure to the cloud. However, to effectively harness the potential of serverless computing at the network edge, one needs to support flexible and adaptive deployment and orchestration of the application so that edge resources are used, taking into account the dynamically changing position of the mobile nodes. Furthermore, it is important to be able to change the policy that drives such deployment decisions.

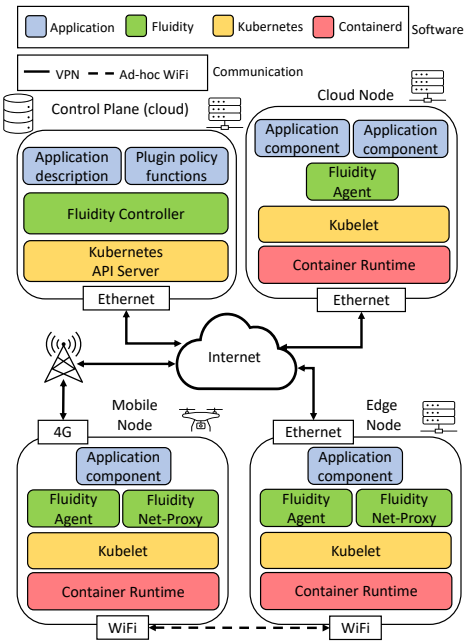
In this work, we introduce Fluidity, a framework that enables flexible application deployment and orchestration across the entire system continuum. We support serverless and modular applications where the developer merely provides the individual components as containers and annotates them with their resource, deployment, and interaction requirements. Based on this information, Fluidity deploys the application components in the cloud, on edge nodes, and mobile IoT nodes, and adapts this deployment at runtime without any intervention from the application owner or system administrator.

The key points of this work are the following: (i) We introduce a framework for the flexible and adaptive deployment and orchestration of serverless applications in the cloud-edge-IoT continuum. (ii) Through a structured API, we separate the core deployment mechanism from the policy that takes application placement decisions, making it possible to plug in and switch between different policies at runtime. (iii) We evaluate the scalability of the core Fluidity deployment and telemetry mechanisms, showing that it is possible to support the deployment, scaling, and monitoring of a large number of application components with reasonable overhead.

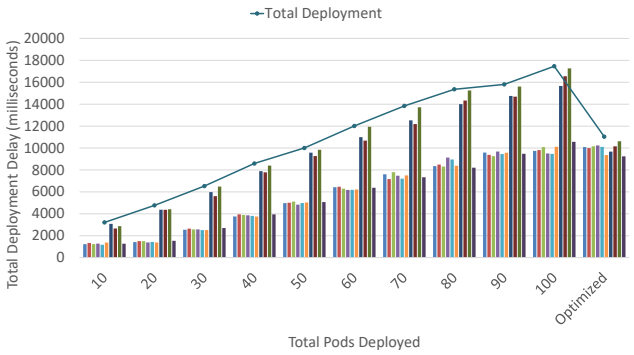
2 Design & Implementation

Fluidity is built as an extension to Kubernetes [kub], which is used as the underlying mechanism for the basic pod/container deployment and health monitoring. Going beyond the standard functionality of Kubernetes, we support flexible application deployment for mobile nodes and edge computing with transparent redirection of application traffic over different network interfaces supporting IP-based communication, which may employ different technologies at the physical layer.

Figure 1a depicts the software architecture of the Fluidity framework on top of a system infrastructure. More specifically, the control plane (called Controller) resides in the cloud, and is responsible for processing application deployment requests, finding a plan with a suitable component-to-node mapping, and executing the deployment plan. The Controller monitors the state of the system and, if needed, adapts the current deployment. The application components are deployed on the selected hosts via Kubernetes as pods with the corresponding containers. We use K3S [k3s], which can run even on low-end devices, while all monitoring information is received via the standard Kubernetes API. Each worker node that can act as a host for application components runs the Fluidity Agent that registers the node with Kubernetes by sending a corresponding resource description, keeps track of the node's state and resource availability, and sends updates to the Kubernetes registry. In a mobile node, the Agent also periodically sends to Kubernetes its current position. Such updates are captured by the Controller, which, in turn, decides if it needs to adapt the deployment of the application. The Agent runs on the side of the Kubelet responsible for the local deploy-



(a) Cluster configuration.



(b) Parallel execution of a scale-out component pattern to 10 nodes for different values of total pods. For a given total number of pods, each bar corresponds to one of the VMs (1 to 10 from left to right).

Figure 1: Fluidity architecture and scalability experiments.

ment and management of the pods. The runtime used to execute the application containers is containerd [con]. In addition to the Agent, mobile and edge nodes with a wireless (WiFi) interface run the Net-Proxy, which implements the redirection of application traffic from the default data path (used for all interactions with the control plane) over a direct wireless link.

3 Evaluation

For an indicative adaptation plan, the average end-to-end adaptation delay is 4.9 seconds, mainly due to the Pod deployment (1.6 seconds) and application traffic redirection (3 seconds), while the overhead of policy switching at runtime is less than 25 milliseconds. We also evaluated the application performance based on 2 adaptation policies: (i) naive policy that always relocates the service-providing component whenever the drone is in range of an edge node, and (ii) data-driven policy which estimates performance based on historical data and relocates only when the predicted performance is beneficial. The policies have an invocation failure ratio (due to component relocations) equal to 14% and 3.7%, respectively. Finally, the end-to-end invocation delay is reduced by 64% at the edge vs the cloud.

We evaluate the ability of Fluidity to execute different adaptation patterns in a parallel way to minimize the respective aggregate delay. We evaluate scale-out and scale-down patterns for a given component and perform measurements for up to 100 instances. The control plane runs in a separate VM, while 10 additional VMs are used as worker nodes that can host application components. The physical machines used for these experiments have sufficient resources to host the aforementioned VMs, and are interconnected via a high-speed 1 Gbps LAN. Figure 1b depicts the total overhead for the scale-out experiments. Each column corresponds to the overhead on each node, while the solid line marks the total/aggregate overhead of the respective parallel execution by Fluidity. Pods are equidistributed to the nodes.

In addition, the right-most column group shows the per-node and total delay for the scale-out to 100 pods, provided we perform a more efficient (rather than equi-) pod distribution, based on the speed/performance of each node, so as to minimize the total delay. As expected, the total delay is determined by the deployment task/job with the highest completion time. The latter is affected by the node's kubelet performance for deploying a given number of pods. For example, we can observe that nodes 7 to 9 are consistently slower due to worse disk performance compared to the rest nodes. The control plane, on the other hand, is not the bottleneck; it can distribute and execute overlapping deployment tasks efficiently while having a maximum CPU utilization of 18%.

The results confirm the scalable implementation of the Fluidity deployment mechanism. More specifically, in the case where 100 pods are deployed (10 to each node), the total delay is just 17.4 seconds. This yields a speedup equal to 6.8x compared to the sequential deployment of 100 pods to the respective nodes which would require at least 118.7 seconds (based on the individual deployment delays of 10 pods per node). Notably, if we optimize the placement of 100 pods across the 10 nodes, the delay drops to 11 seconds, corresponding to a speedup of 10.76x compared with the sequential deployment. Similarly, the speedup of the parallel scale-down for 100 pods is equal to 3.6x compared to the serialized removal which would have a total delay of approximately 1.5 seconds. To this end, the core Fluidity mechanisms are scalable to applications with a large number of components.

4 Conclusions & Future Work

We have introduced Fluidity, a framework for the automated and adaptive deployment of serverless and modular applications in systems with cloud, edge, and mobile nodes. We discussed its architecture and implementation details. The provided flexibility is necessary for next-generation compute systems with complex serverless applications operating on multi-layer, heterogeneous infrastructures, with varying characteristics. Our results confirm that Fluidity can execute adaptation plans produced by a policy in a scalable way. As future work, we plan to support hierarchical policies where Fluidity agents will accommodate policies based on their respective layers.

Acknowledgments

This work has received funding from the Horizon Europe research and innovation programme of the European Union, under grant agreement no 101092912, project MLSysOps.

References

[con] Containerd. https://containerd.io/.

[dat] Datadog, State of Serverless Report. https://www.datadoghq.com/state-of-serverless/.

[k3s] K3S. https://k3s.io/.

[kub] Kubernetes. https://kubernetes.io/.