TDD4Fog: A Test-Driven Software Development Platform for Fog Computing Systems

Rui Li[1], Xiao Liu[1], Xi Zheng[2], Chong Zhang[1], Huai Liu[3]
[1]School of Information Technology Deakin University, Geelong, Australia
[2]Department of Computing Macquarie University, Sydney, Australia
[3]School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia
rui@deakin.edu.au, xiao.liu@deakin.edu.au, james.zheng@mq.edu.au, zhangcho@deakin.edu.au, hliu@swin.edu.au

ABSTRACT

As an ideal infrastructure for smart services, Fog Computing is becoming the next wave of IT investment harnessing the successful models of Cloud Computing and latest technologies such as 5G and Internet of Things (IoT). However, the development of Fog Computing systems is a big challenge due to its complex, heterogeneous and distributed nature. Currently, there are a few SDKs released by some public Cloud service providers to support the development of Fog services in a top-down fashion as the key motive is to leverage their business Cloud services. However, Fog Computing systems are usually designed in a bottom-up fashion as the major functionalities are centred around the Edge Nodes and the End Devices. Meanwhile, significant efforts are required to verify the conformance of software behaviours as the collaboration between the End Devices, Edge Nodes and Cloud Servers is vital to the success of a Fog Computing System. Therefore, a holistically designed software development platform is urgently required. In this paper, we propose TDD4Fog, a test-driven software development platform for Fog Computing systems. Following the Test-Driven Development (TDD) methodology and a bottom-up design fashion, TDD4Fog supports the microservice architecture and provides the Test-Driven utilities such as metamorphic testing, mutation testing and random testing for the whole software development lifecycle of Fog Computing systems. To demonstrate the feasibility of TDD4Fog, we have presented some preliminary results on the key components of TDD4Fog and discussed some important future research directions.

KEYWORDS

Test-Driven Development, DevOps, Microservices, Fog Computing, Smart Services

1 Introduction

With the rapid development of Internet of Things (IoT), we are now living in a world of smart services such as smart transportation, smart manufacturing and smart health [1, 2]. As most of the smart services are time sensitive, Fog Computing has been proposed as the latest computing paradigm to overcome the latency issue of Cloud services over the Internet. Fog Computing creates an extra layer of Edge Nodes between the layer of End Devices and the layer of Cloud Servers so that computation resources can be provided much closer to the End Devices [3]. However, as Fog Computing systems are normally complex in functionality, heterogeneous in computing resources, and distributed in system architecture, the development of Fog Computing systems is a big challenge. For instance, an autonomous vehicle system (e.g. the Baidu Apollo Platform[1]) requires toolkits to design, develop, test, and deploy software components for sensor collection, sensor fusion, perception, planning, controlling and all these components need to be thoroughly tested in both simulated and real-world environments throughout the whole Software Development Lifecycle (SDLC). Therefore, a software development platform with a holistic design for the whole SDLC of Fog Computing systems is essential and urgently required. Specifically, we identify the following two major requirements in the development of Fog Computing systems:

1) The design of Fog Computing systems needs to follow a bottom-up fashion. Unlike many Cloud Computing systems which are usually designed in a top-down fashion as majority of the data and computation are residing on the Cloud, bottom-up design allows the developers to focus on the communication and computation taking place at the Edge as the major system functionalities are implemented via the collaboration between the End Devices and Edge Nodes.

2) The development of Fog Computing systems requires the continuous quality assurance, especially the conformance with expected system behaviours as the collaboration between the End Devices, Edge Nodes and Cloud Servers is vital to the success of a Fog Computing System. However, it is extremely difficult, if not impossible, to validate and verify the whole system after most development work has been conducted. In other words, the philosophy of continuous testing must be followed in the rapid development of Fog Computing systems. Various testing activities must be systematically integrated in SDLC such that bugs and runtime errors can be fixed and handled as quickly as possible.

As shown in Table 1, currently there are a few Fog development platforms/SDKs released by some public Cloud service providers to support the development of Fog services, but they are usually designed in a top-down fashion and their key motive is to leverage

[1] http://apollo.auto/developer.html

their business Cloud services. For example, Microsoft Azure IoT Edge[2] allows developers to employ Azure AI services and deploy their own codes on the Edge Nodes. It consists of three components: IoT Edge modules which are containers for services running at the Edge Nodes; IoT Edge runtime which runs at the Edge Nodes and manages the communication between the End Devices, the Edge Nodes and the Cloud Servers; the Cloud-based interface which enables the remote monitoring of Edge Nodes. An IoT Edge simulator is also provided for debugging modules running at the Edge. Huawei Intelligent EdgeFabric (IEF)[3] focuses on the management of Edge Nodes and the push-down of Huawei Cloud services in FunctionGraph to the Edge. The developers can pack their Edge services into containers and upload to Huawei Cloud SWR (Software Repository for Container), and then create Edge application templates in IEF before they can be deployed and running on the Edge Nodes. IEF has provided a set of APIs for the management of Edge Nodes but no complete SDKs are available yet. ThingWorx[4] is a platform designed for the industrial Internet of Things. ThingWorx WebSocket-based Edge MicroServer (WS EMS) works with Edge Nodes to ensure the secure and efficient communication back to the ThingWorx platform. Through its Protocol Adapter Toolkit, ThingWorx supports the connection to both Azure IoT Hub and AWS IoT. There are also a few open-source projects. KubeEdge[5] is an open-source system built upon Kubernetes to extend native containerised application orchestration capabilities to the Edge. KubeEdge consists of a Cloud part and an Edge part. Developers can develop containerised applications and run them anywhere on the Edge or in the Cloud. They can also manage devices and monitor app and device status on Edge nodes, the same as in a traditional Kubernetes cluster.

Table 1: Comparison of Major Fog Development Platforms

Fog Platform	Type	Cloud Platform	Design Fashion	Microservices Architecture	Lifecycle Support	Testing Utilities
Azure IoT Edge	Commercial	Azure Cloud	Top-Down	YES	N/A	Simulator for Edge Runtime
EdgeFabric	Commercial	Huawei Cloud	Top-Down	YES	N/A	N/A
ThingWorx	Commercial	No Limit	Top-Down	YES	N/A	N/A
KubeEdge	Opensource	No Limit	Bottom-UP	YES	N/A	N/A
TDD4Fog	Opensource	No Limit	Bottom-UP	YES	YES	Comprehensive

In summary, all major platforms adopt the container technology to support microservice architecture, but none of the existing major platforms provide SDLC support or comprehensive testing utilities. However, Fog Computing systems require tailored toolkits to support the SDLC in a bottom-up fashion and usually such software components are mission-critical thus requires test driven utilities. Therefore, we propose TDD4Fog, a test-driven software development platform for Fog Computing systems which follows the Test-Driven Development (TDD) methodology and a bottom-up design fashion. TDD4Fog provides comprehensive Test-Driven utilities such as metamorphic testing, mutation testing and random testing for the whole SDLC of Fog Computing systems. Details about TDD4Fog are described in the next section.

2 TDD based Software Development Platform

TDD is a mainstream practice for rapid software development. It aims at achieving short and efficient development cycle based on self-checking tests that represent new features to be added into the user requirements. Since Fog Computing systems normally require the agility and independence for its development and deployment, the microservice architecture provides an excellent solution by delivering micro-sized independent processes implementing specific functionalities [1]. In the development of microservices based ecosystem, TDD is particularly focused on the development and testing of new behaviours (corresponding to new features in the general context) added in each development cycle [4]. More specifically, a test suite is first created to represent those specific behaviours that the development team intend to introduce into the Fog Computing system. All tests in the suite must be validated such that they can fail the ecosystem. Then, some simple code is written to implement the added behaviours. The coding will be continued until the system can pass all tests. Last but not the least, refactoring is required to improve the readability and maintainability of the code. Such a cycle will be repeated for the continuous introduction of more and more behaviours into the Fog Computing system. In a word, the microservice architecture enables the bottom-up design while TDD empowers the development of individual microservices for the whole Fog Computing system. Based on such a microservice architecture and TDD, we present the overview of our TDD4Fog software development platform as depicted in Figure 1.

TDD4Fog is mainly targeted towards those intelligent and distributed systems (e.g. autonomous vehicles, smart farms, smart home, smart health) where Fog Computing is urgently required to solve issues with big data, network latency, scalability and reliability [1]. The left side of Figure 1 is the widely accepted 3-layer Fog Computing architecture which consists of End Devices layer, Edge Nodes layer, and Cloud Servers layer [3]. For instance, in autonomous vehicles application, the End Devices are those vehicles and Edge Nodes are road-side units (RSUs). We also propose a high-fidelity emulator for the underlying Fog Computing environment, which is shown in the middle of Figure 1. The emulator can reflect not only the network configuration but also the physical attributes of each key component for the underlying Fog Computing environment. For instance, in autonomous vehicles, the emulator shall be able to provide highly accurate emulation for the vehicle networks along with physical emulation for each vehicle [2]. To achieve extensibility of the emulator, the emulator has an interface which allows to add or remove emulation component for a specific Fog Computing system.

[2] https://azure.microsoft.com/en-au/services/iot-edge/
[3] https://support.huaweicloud.com/ief/index.html
[4] https://support.ptc.com/help/thingworx_hc/thingworx_edge_sdks_ems/
[5] https://kubeedge.io/en/

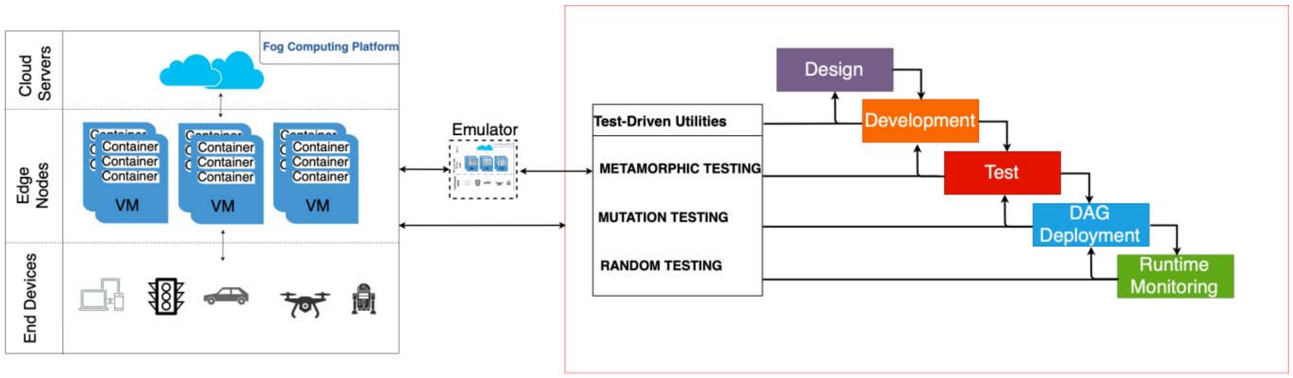

Figure 1: A Test-Driven Software Development Platform for Fog Computing Systems

On the right of Figure 1, we illustrate the TDD based software engineering development life cycle (TDD-SDLC) for the development of microservice based Fog Computing systems from a bottom-up design perspective. Specifically, the typical test-driven utilities include metamorphic testing, mutation testing and random testing. As discussed in Section 1, due to the lack of lifecycle support in major Fog development platforms, we propose to create SDLC tools and metrics specifically for Edge Nodes and End Devices. For instance, in autonomous driving application, we need to develop distributed data analytics and machine learning on RSUs for autonomous driving, and we also need to develop specific components on vehicles itself which include sensor collection, sensor fusion, and decision layers [2].

Here, we present more details on the test-driven utilities in TDD4Fog. As discussed in Section 1, TDD for microservices based system is particularly focused on the system behaviours. As such, the test-driven utilities used in our TDD4Fog framework must be able to create tests that can effectively represent and check the expected behaviours. For example, metamorphic testing (MT) constructs tests based on some necessary properties of the expected system behaviours, represented by the so-called metamorphic relations (MRs) among multiple system inputs and their corresponding outputs. Apparently, tests created by MT can directly represent the added behaviours. In addition, the MRs in MT can provide an easy-to-automate mechanism for checking test results through the comparison of multiple outputs against MRs. In other words, tests constructed in MT also support self-checking specifically against critical properties of the added behaviours. In addition to testing, MT can be used to guide other stages in the development lifecycle. MRs can facilitate the design to guarantee the consistency with user requirements by providing necessary properties of the intended behaviours. Tests generated by MT will be applied as the guidelines for writing the code (the Development stage). Moreover, MRs can serve as reference points for evaluating the acceptability in deployment. MT tests can further be used to guide the monitoring to check whether the behaviours still hold in runtime. Take the development for autonomous driving as an example. One necessary property for driving behaviours, represented by an MR, is that randomly scattered points added outside the so-called Region of Interest (ROI) “should not cause an obstacle on the roadway to become undetectable” [8]. According to our TDD4Fog platform, such a property must be fully considered throughout the whole SDLC. For instance, the coding should distinguish the executions for inputs inside and outside ROI, and the real-time monitoring ought to check whether the MR is violated or not. In addition to MT, other possible driving utilities include mutation testing (µT) and random testing (RT), which can represent and check service behaviours from different perspectives. µT, based on the so-called mutants with intentionally seeded faults, can facilitate the proactive avoidance of specific fault types in the intended behaviours. RT, by providing some “fuzzy” tests, is particularly suitable for detecting safety-critical issues for the intended behaviours, such as crashing of a microservice or even the whole ecosystem.

3 Preliminary Results

In this section, we present some of our preliminary results for the investigation on TDD4Fog.

3.1 Metamorphic Testing of Microservices

A preliminary study has been conducted to apply Metamorphic Testing (MT) into the verification of microservices [5]. Due to unique characteristics of microservices, it is extremely difficult, if not impossible, to verify test results given any possible input to a certain microservice. This so-called oracle problem is a fundamental problem in software testing and verification. MT has been demonstrated as a simple yet effective approach to the oracle problem in the community of software testing. It has helped detect different types of real-life defects in various application domains. In our preliminary study, we integrated MT into the verification of microservices. An experimental study was conducted based on three typical microservices, for which seven simple metamorphic relations (MRs) were constructed. Despite the simplicity of MRs, MT was justified to have a very high effectiveness in detecting various faults without the need of complete oracles. As one of the test-driven utilities, MT will play a significant role in TDD4Fog.

3.2 Deployment Framework for Microservice

We have done a preliminary investigation to the deployment of microservices [6]. SmartVM is a business Service Level

Agreement (SLA) aware, microservice-centric deployment framework. SmartVM splits and organises microservices into a multi-tier architecture based on the intensity of their workload. The traditional library functions and middleware are turned into separate API microservices that can scale independently. Business features are grouped into Business microservices by their patterns of API access, resulting in less conflicting resource requirements. Furthermore, SmartVM maintains SLA compliance by utilising application-level metrics for autoscaling. This allows SmartVM to more accurately and dynamically scale to accommodate fluctuating user traffic of SaaS (Software as a Service) applications while saving costs by reducing over-provisioning. Our experiments show up to 66% cost reduction compared with the state-of-the-art uniform microservices approach with reduced SLA violation. SmartVM will play a significant role in the deployment of microservices in TDD4Fog.

3.3 Simulation Tool for Fog Computing Systems

A preliminary simulation tool has been created to evaluate the runtime performance of Fog Computing systems. FogWorkflowSim [7] is an efficient and extensible toolkit for automatically evaluating resource and task management strategies in Fog Computing with simulated user applications defined in DAG (Directed Acyclic Graph), namely workflows. FogWorkflowSim is able to: 1) automatically set up a simulated Fog Computing environment for workflow applications; 2) automatically execute user submitted workflow applications; and 3) automatically evaluate and compare the performance of different computation offloading and task scheduling strategies with three basic performance metrics, including time, energy and cost. FogWorkflowSim can serve as an effective experimental platform for researchers in Fog based workflow systems as well as practitioners interested in adopting Fog Computing and workflow systems for their new software projects. As an important part of the Emulator, FogWorkflowSim will play a significant role in the testing and runtime monitoring of TDD4Fog.

4 Conclusions and Future Work

The market for Fog Computing is rapidly growing with the increasing popularity of smart services. However, the design and development of Fog Computing systems are posing new challenges to the Software Engineering community due to its complex, heterogeneous and distributed nature. In this paper, we have presented the new idea about TDD4Fog, a test-driven software development platform for Fog Computing systems which is to provide a holistic design to support the software development lifecycle for Fog Computing systems. To demonstrate the feasibility of our proposed solution, we have also presented some of our preliminary results such as the testing of microservices, the deployment of microservices, and the simulation tool for Fog workflow applications.

The next step is to complete the development of TDD4Fog and evaluate its effectiveness in the development of real-world Fog Computing systems. There are three major research directions that we aim to investigate in the future based on TDD4Fog:

1) The extension of our preliminary study [5] from MT in testing to the more general context of whole SDLC, including MR-guided design and maintenance (note that such work is not only novel in fog computing and microservice, but also the first of its kind in software engineering); in addition, integration of MT with µT and RT in TDD (note that such integration is not new in software testing). Another possible direction is the combination of studies of [5] and [6], that is, using MT in the SLA-aware microservice deployment framework. Similarly, it is also possible to use MT in the V&V for simulation tool.

2) The extension of basic task offloading in our preliminary study [7] to more efficient service migration strategies for microservices. In Fog Computing systems, due to the mobility of End Devices and the dynamic nature of Edge Nodes, it often requires fast service migration from one Edge Node to another Edge Node or a Cloud Server to ensure the low latency and high availability of Fog services. Therefore, efficient service migration strategies based on container technology will be designed for TDD4Fog. We will also look at the opportunities brought by 5G and 6G which will significantly increase the data transfer speed between End Devices and Edge Nodes.

3) The extension of support for intelligent services. Most Fog Computing systems are running intelligent services. For example, distributed deep learning models can be deployed into a variety of computing nodes in the Fog Computing system including Edge Nodes, Cloud Servers and even End Devices [9]. TDD4Fog will provide SDLC support to the development of intelligent Fog services and one of the important future works is the efficient testing of intelligent Fog services such as deep learning models.

REFERENCES

[1] R. Naha, S. G., D. Georgakopouols, P. Jayaraman, L. Gao, Y. Xiang, R. Ranjan Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions. IEEE Access, 6 (2018), 47980-48009.

[2] P. Jin, D. F., A. Hall and C. Walton Emerging Transportation Technologies White Papers. City, 2015.

[3] Consortium, O. OpenFog Reference Architecture for Fog Computing. City, 2017.

[4] Gao, M. R. a. J. A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development. In Proceedings of the 2015 IEEE Symposium on Service-Oriented System Engineering, San Francisco Bay, CA, USA, 2015.

[5] G. Luo, X. Z., H. Liu, R. Xu, D. Nagumothu, R. Janapareddi, E. Zhuang and X. Liu. Verication of Microservices Using Metamorphic Testing. In Proceedings of the 19th International Conference on Algorithms and Architectures for Parallel Processing, Melbourne, Australia, 2019.

[6] T. Zheng, X. Z., Y. Zhang, Y. Deng, E. Dong, S. Yan, R. Zhang, X. Liu SmartVM: a SLA-aware Microservice Deployment Framework. World Wide Web, 22 (209), 275–293.

[7] X. Liu, L. F., J. Xu, X. Li, L. Gong, J. Grundy, Y. Yang. FogWorkflowSim: An Automated Simulation Toolkit for Workflow Performance Evaluation in Fog Computing. In Proceedings of the 34th IEEE/ACM International Conference on Automated Software *Engineering*, San Diego, California, United States, 2019.

[8] Z. Q. Zhou and L. Sun. Metamorphic testing of driverless cars. Communications of the ACM: 62(3): 61-67, 2019.

[9] S. Teerapittayanon, B. McDanel, H.T. Kung, Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices, 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 5-8 June 2017, Atlanta, GA, USA.