Programming Support for the Cloud

Konstantinos Kallas
University of Pennsylvania
Philadelphia, PA, USA
kallas@seas.upenn.edu

ABSTRACT

The way we write programs is going through a fundamental shift due to the advent of the service-oriented software economy that is powered by cloud providers. Functionality is often deployed and offered as a service, instead of packaged as a library that is imported by the user program. Applications themselves are often broken up in different services, each of which implements different functionality. This is in part due to the increasing offloading of deployment and administrative tasks to cloud providers, which can be done more seamlessly when applications are structured as services. The increasing popularity of serverless indicates that these changes are here to stay.

On the surface, using services instead of libraries makes no difference from a developer's perspective, since they both provide interfaces that *in theory* adequately describe how to use them and what to expect from them. However, there are multiple problems that are caused by this shift. Cloud applications that compose different services are complex distributed systems (even though they might not look like it), the code of external services is often not accessible, and their APIs do not express all the necessary information for the composition to be correct. This prohibits the use of established formal methods techniques to improve software reliability and guarantee correctness. As an example, idempotence and consistency guarantees are often not part of service APIs. What should a user expect when their application crashes after it called an external service? Should the call be repeated? This problem is exacerbated by composition. What are the guarantees of an application that composes different services with different, potentially non-aligned, guarantees. Do we expect cloud programmers that want to implement a simple web app on serverless to be distributed systems experts?

There are two different paths that we can take to resolve these problems, both of which have their benefits. First, we can try to hide complexity from the users, which is what most recent work and proposals on programming support for the cloud have focused on. Instead of letting users (mis)use services to implement their applications, we can design abstractions that behave more like what users are already used to, and then implement these abstractions by composing different services. The implementations of these abstractions can be formally verified to satisfy their specification once, ridding developers from such reasoning. My recent work on Durable Functions [1, 2] is making steps towards that direction. Durable Functions is a programming model that composes a persistent log, a key-value store, and a serverless platform, and then hides them from the user by providing them with well-known abstractions (async-await and actors) to implement their stateful applications. The implementation is formally proven to provide exactly-once execution guarantees despite faults. We are also currently working on a programming model [3] for stateful service applications that supports programs written in standard python code extended with a transaction primitive and implements them on serverless while providing exactly-once execution guarantees. The user-written program is transformed and composed with a logging service and a database to correctly and efficiently execute in a serverless environment. The framework includes a formal proof of correctness, providing transactional guarantees and exactly-once execution in the context of faults.

However, hiding complexity from the users can only go that far, since application trends evolve in unpredictable ways, and abstractions are often leaky, requiring their users to eventually uncover them if they need better performance or more complex guarantees. The other path is to bite the bullet and expose the complexity of service APIs to the users, albeit in a principled and careful way, so that it is not only understandable by users, but also usable by automated reasoning tool like static analyzers and optimizing compilers. We can then develop tooling support that can either find errors in programs earlier, e.g., identifying that a service that provides an idempotent API needs to log the result of a call to another *non-idempotent* service, or enable additional generic optimizations having access to higher-level specifications of these APIs, e.g., requests to an eventually consistent service can be buffered, batched, and then sent all together to the service.

REFERENCES

[1] Sebastian Burckhardt, Badrish Chandramouli, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, Christopher S.

Meiklejohn, and Xiangfeng Zhu. 2022. Netherite: Efficient Execution of Serverless Workflows. *Proceedings of the VLDB Endowment* 15 (2022). https://arxiv.org/abs/2103.00033

[2] Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S Meiklejohn. 2021. Durable functions: semantics for stateful serverless. *Proceedings of the ACM on Programming Languages* OOPSLA (2021).

[3] Konstantinos Kallas, Haoran Zhang, Rajeev Alur, Sebastian Angel, and Vincent Liu. 2022. Implementing Microservice Applications on Serverless, Correctly. (2022).

SHORT BIO

I am a 4th year PhD student at the University of Pennsylvania, working in the intersection of programming languages and distributed systems. I am generally interested in applying techniques from the fields of programming languages and formal methods to improve system design. Recently, my focus has been on (i) the design and implementation of programming models for serverless computing, (ii) automatic parallelization and distribution of shell scripts, (iii) the design and implementation of programming models for stream processing.

My work has been published on both programming languages and systems venues including OSDI, EuroSys, VLDB, OOSPLA, ICFP, and PPoPP.

I am interested in participating at the HPTS workshop to bring a programming languages infused perspective to the discussion of how to design and build *correct* and *efficient* systems for the cloud era.

For more information you can check my website at https://angelhof.github.io.