Stateful Entities: Object-oriented Cloud Applications as Distributed Dataflows

Kyriakos Psarakis$^\delta$
Guido Salvanesci†
Wouter Zorgdrager$^\delta,\gamma$
Asterios Katsifodimos$^\delta$
Marios Fragkoulis$^\gamma$

$^\delta$Delft University of Technology
$^\gamma$University of St. Gallen
$^\gamma$Delivery Hero SE

Figure 1: Two stateful entities: User and Item. The content of imperative functions is split into multiple functions that access the common state of a given entity. Those functions are then encoded into a stateful dataflow graph that can be executed in a distributed streaming dataflow engine. As a result, i) imperative code is executed in an event-based manner without the need to block, and ii) the code retains exactly-once processing guarantees without the need for programmers to write failure-handling code such as state management, call retries or idempotency.

ABSTRACT

While there are multiple approaches for distributed application programming (e.g., Bloom [2], Hilda [14], Cloudburst [12], AWS Lambda, Azure Durable Functions, and Orleans [3, 4]), in practice developers mainly use libraries of popular general purpose languages such as Spring Boot in Java, and Flask in Python. None of these approaches offers message processing guarantees, failing to support exactly-once processing: the ability of a system to reflect the changes of a message to the state exactly one time. Instead, all of the above approaches offer at-most- or at-least-once processing semantics. Programmers then have to “pollute” their business logic with consistency checks, state rollbacks, timeouts, retries, and idempotency [8, 9].

We argue that no matter how we approach cloud programming, unless an execution engine offers exactly-once processing guarantees, we will never remove the burden of distributed systems aspects from programmers. In short, exactly-once processing should be assumed at the level of the programming model. To the best of our knowledge, the only systems able to guarantee exactly-once message processing [5, 11] at the time of writing, are batch [1, 7, 15] and streaming [6, 10, 13] dataflow systems. However, their programming model follows the paradigm of functional dataflow APIs which are cumbersome to use, and require training, and heavy rewrites of the typical imperative code that developers prefer to use for expressing application logic.

For these reasons, we believe that the dataflow model should be used as low-level IR for the modeling and execution of distributed applications, but not as a programmer-facing model. Technically, one of the main challenges in adopting a dataflow-based intermediate representation, is that the dataflow model is essentially functional, with immutable values being propagated across operators that typically do not share a global state. Hence, adopting a dataflow-based IR entails translating (arbitrary) imperative code into the functional style. Compiler research has systematically explored only the opposite direction: to compile code in functional programming languages into a representation that is executable on imperative architectures – like virtually all modern microprocessors. Yet, the translation from imperative to functional or dataflow programming remains largely unexplored.

To this end, we report on Stateful Entities a prototypical programming model (exemplified in Figure 1), compiler pipeline, and IR that compiles imperative, transactional object-oriented applications into distributed dataflow graphs and executes them on existing dataflow systems. The proposed system presented in this paper can be found at: https://github.com/delftdata/stateflow. Our preliminary experiments showed that the translation of imperative programs into dataflow graphs yields very promising performance results, of less than 50ms latency.

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution, provided that you attribute the original work to the authors and CIDR 2023. 13th Annual Conference on Innovative Data Systems Research (CIDR ’23). January 8-11, 2023, Amsterdam, The Netherlands. doi:
REFERENCES