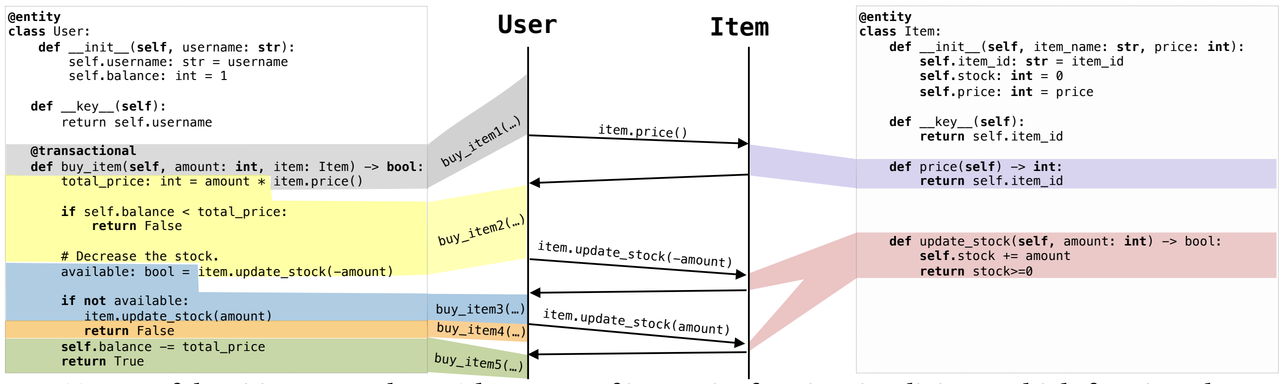
Stateful Entities: Object-oriented Cloud Applications as Distributed Dataflows

Kyriakos Psarakis[δ] Wouter Zorgdrager[η,δ] Marios Fragkoulis[η]
Guido Salvaneschi[†] Asterios Katsifodimos[δ]
[δ]Delft University of Technology [†]University of St. Gallen [η]Delivery Hero SE



```
@entity
class User:
    def __init__(self, username: str):
        self.username: str = username
        self.balance: int = 1

   def __key__(self):
       return self.username

   @transactional
   def buy_item(self, amount: int, item: Item) -> bool:
       total_price: int = amount * item.price()

       if self.balance < total_price:
           return False

       # Decrease the stock.
       available: bool = item.update_stock(-amount)

       if not available:
          item.update_stock(amount)
          return False
       self.balance -= total_price
       return True
```

```
@entity
class Item:
    def __init__(self, item_name: str, price: int):
        self.item_id: str = item_id
        self.stock: int = 0
        self.price: int = price

    def __key__(self):
        return self.item_id

    def price(self) -> int:
        return self.item_id


    def update_stock(self, amount: int) -> bool:
        self.stock += amount
        return stock>=0
```

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.

REFERENCES

[1] [n.d.]. Apache Spark project. http://spark.apache.org/.

[2] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M Hellerstein, and Russell Sears. 2010. Boom analytics: exploring data-centric, declarative programming for the cloud. In *EuroSys*.

[3] Phil Bernstein, Sergey Bykov, Alan Geller, Gabriel Kliot, and Jorgen Thelin. 2014. Orleans: Distributed virtual actors for programmability and scalability. *MSR-TR*.

[4] Sergey Bykov, Alan Geller, Gabriel Kliot, James R Larus, Ravi Pandya, and Jorgen Thelin. 2011. Orleans: cloud computing for everyone. In *SoCC*.

[5] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State Management in Apache Flink&Reg;: Consistent Stateful Distributed Stream Processing. In *VLDB*.

[6] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink TM : Stream and Batch Processing in a Single Engine. In *IEEE Data Eng. Bull.*

[7] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. In *Communications of the ACM*.

[8] Tom Killalea. 2016. The Hidden Dividends of Microservices. *ACM Queue* (2016).

[9] Rodrigo Laigner, Yongluan Zhou, Marcos Antonio Vaz Salles, Yijian Liu, and Marcos Kalinowski. 2021. Data Management in Microservices: State of the Practice, Challenges, and Research Directions. *PVLDB* 14, 13 (2021).

[10] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: a timely dataflow system. In *ACM SOSP*.

[11] Pedro Silvestre, Marios Fragkoulis, Diomidis Spinellis, and Asterios Katsifodimos. 2021. Clonos: Consistent Causal Recovery for Highly-Available Streaming Dataflows. In *SIGMOD*.

[12] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. 2020. Cloudburst: Stateful Functions-as-a-Service. In *VLDB*.

[13] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. 2014. Storm@ twitter. In *SIGMOD*.

[14] Fan Yang, Jayavel Shanmugasundaram, Mirek Riedewald, and Johannes Gehrke. 2006. Hilda: A high-level language for data-drivenweb applications. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 32–32.

[15] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Gunda, and Jon Currey. 2008. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *OSDI*.