Following the Data, Not the Function: Rethinking Function Orchestration in Serverless Computing

Minchen Yu[1], Tingjia Cao[2], Wei Wang[1], Ruichuan Chen[3]

[1]Hong Kong University of Science and Technology

[2]University of Wisconsin-Madison

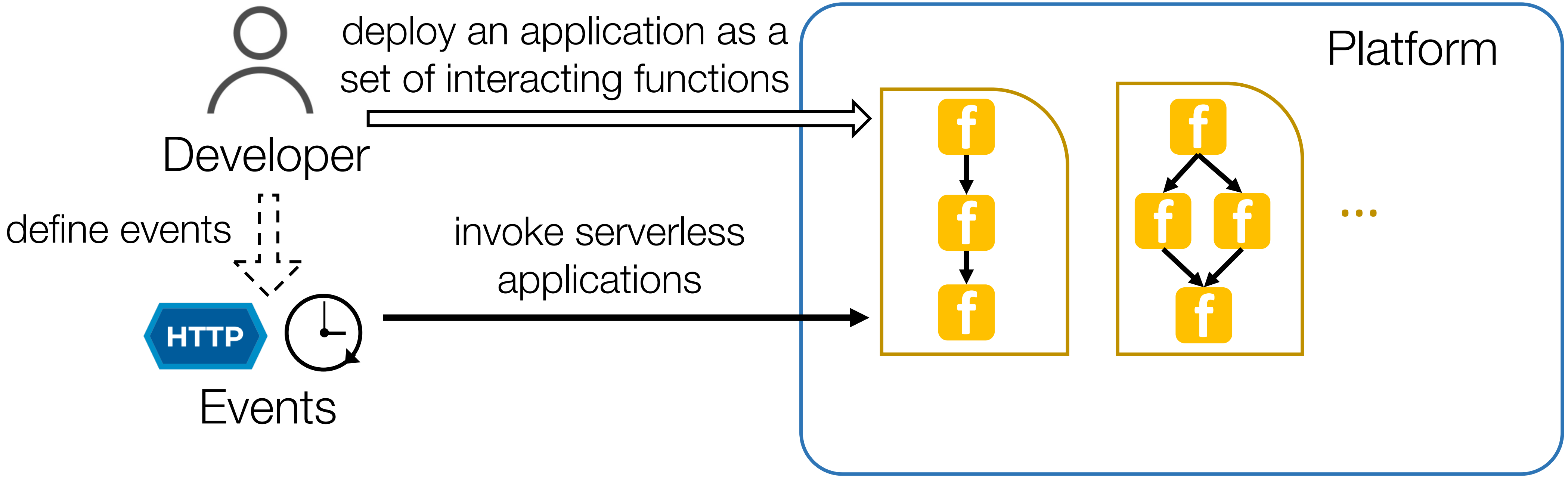
[3]Nokia Bell Labs

Full paper to appear at NSDI'23

Serverless computing (FaaS)

Relieving cloud users from managing servers

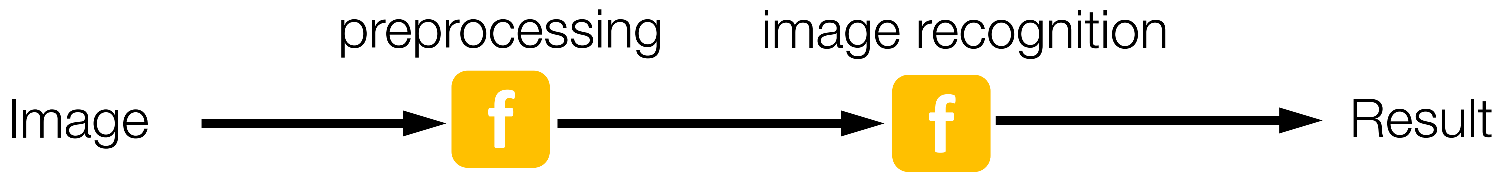
- Automatic scaling
- Pay per use



Orchestrating interacting functions today

Deploying and orchestrating functions as a workflow

- Express interactions between functions as a workflow DAG
- Function-oriented orchestration: driving the workflow execution following the function invocation order

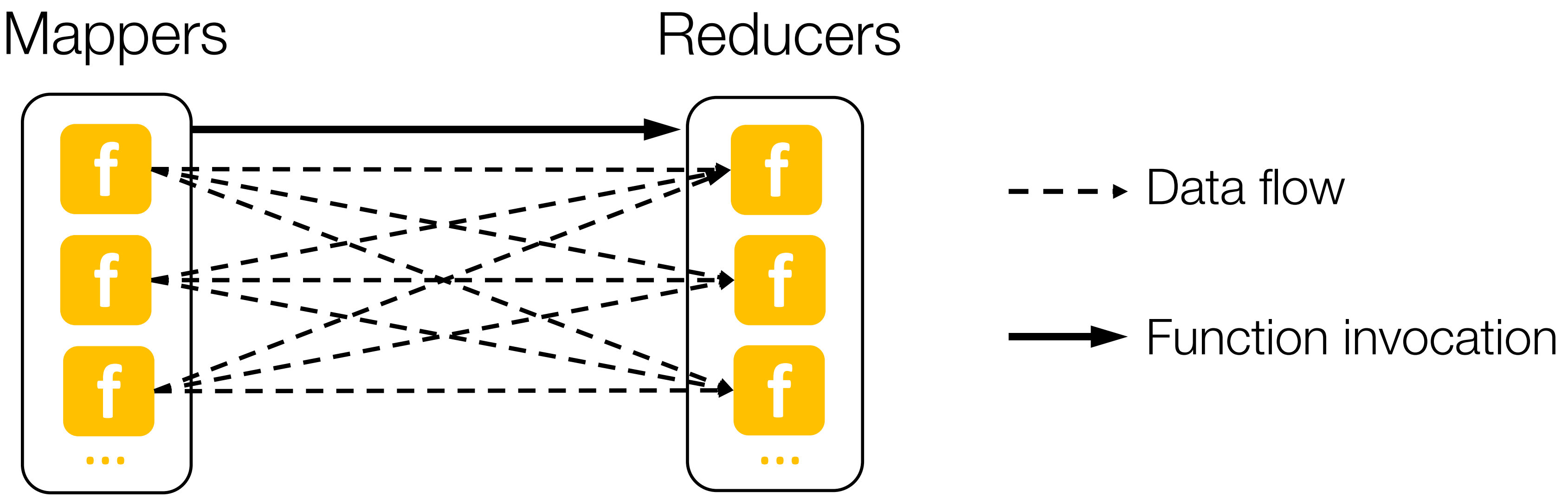


However, function-oriented orchestration has three limitations

- Limited expressiveness
- Limited usability
- Limited applicability

Limited expressiveness

Unable to express many complicated function invocation patterns once the data flow does not exactly follow function invocation

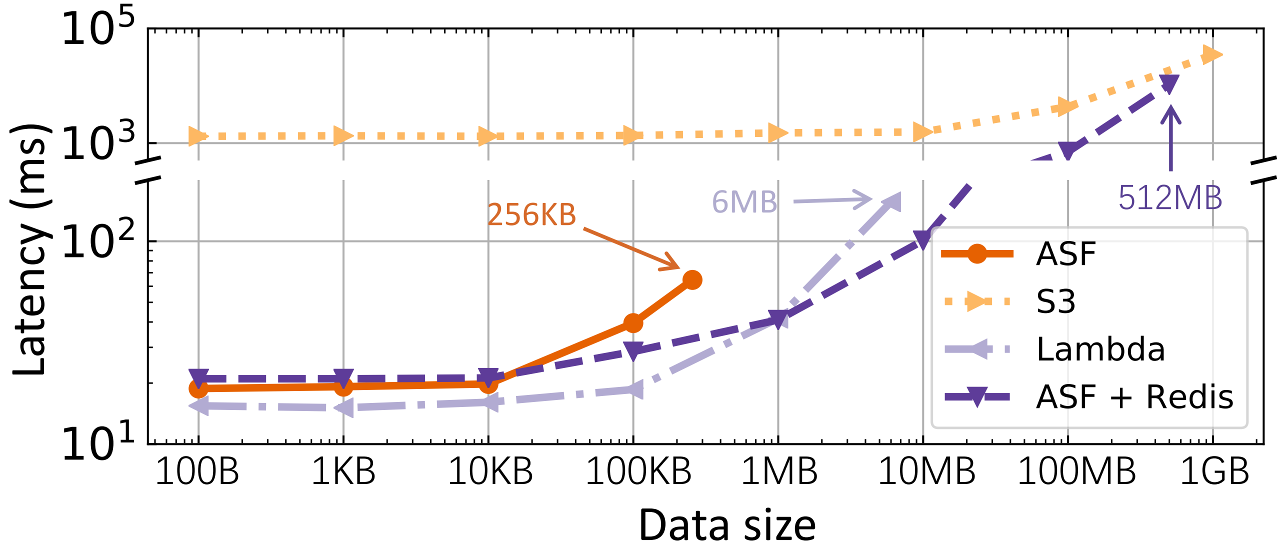


Developers must manually implement the data shuffle, e.g., via external storage such as PyWren [SoCC'17] and Locus [NSDI'19]

Limited usability

Not easy to use for exchanging data between functions

- Lack of direct communication forces users to explore many other options
- No single option has always best performance



Data volume exchanged can be dynamic and unpredictable, making it challenging to select optimal options, e.g., Pocket [OSDI'18] and Sonic [ATC'21]

Limited applicability

Long function invocation delay

- >10 ms to invoke a warm instance in AWS Step Functions, longer for a workflow
- Online services have stringent latency requirements, e.g., 10s of ms

Poor performance in data exchange

- Lack of data locality

Not applicable to latency-critical and data-intensive applications

Desired function orchestration

Existing limitations

- Limited expressiveness
- Limited usability
- Limited applicability

Desired properties of serverless function orchestration

- Rich expressiveness: easily express a rich set of workflow patterns
- High usability: no need to separately handle data exchange
- Wide applicability: applicable to latency-critical and data-intensive applications

Key insight

Why function-oriented orchestration is neither easy to use nor efficient?

Agnostic to intermediate data!

- Unaware of how and when data are consumed in a workflow
- Unaware of data locality, not designed for fast data sharing

Key insight

A desired orchestration approach should be data-centric

- Make data consumption explicit to allow fine-grained data exchange
- Enhance data locality for efficient workflow execution

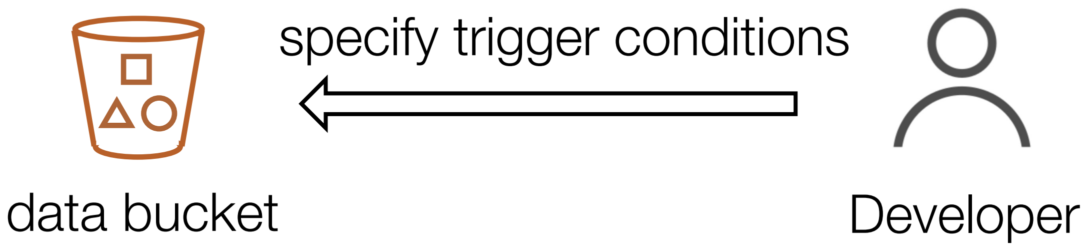
Following the insight, we propose data-centric orchestration

Data-centric orchestration

Let short-lived and immutable intermediate data trigger functions

Developers specify how and when the data bucket triggers functions

Data bucket stores and manages intermediate data



Data bucket triggers function once the condition is met, thus driving the workflow execution

No consistency issue

Data-centric: meeting all desired properties

Rich expressiveness

- Break the tight coupling between function invocation and data flow
- Allow fine-grained data exchange

High usability

- Unified interface for both function invocations and data exchange
- No need to separately implement data exchange

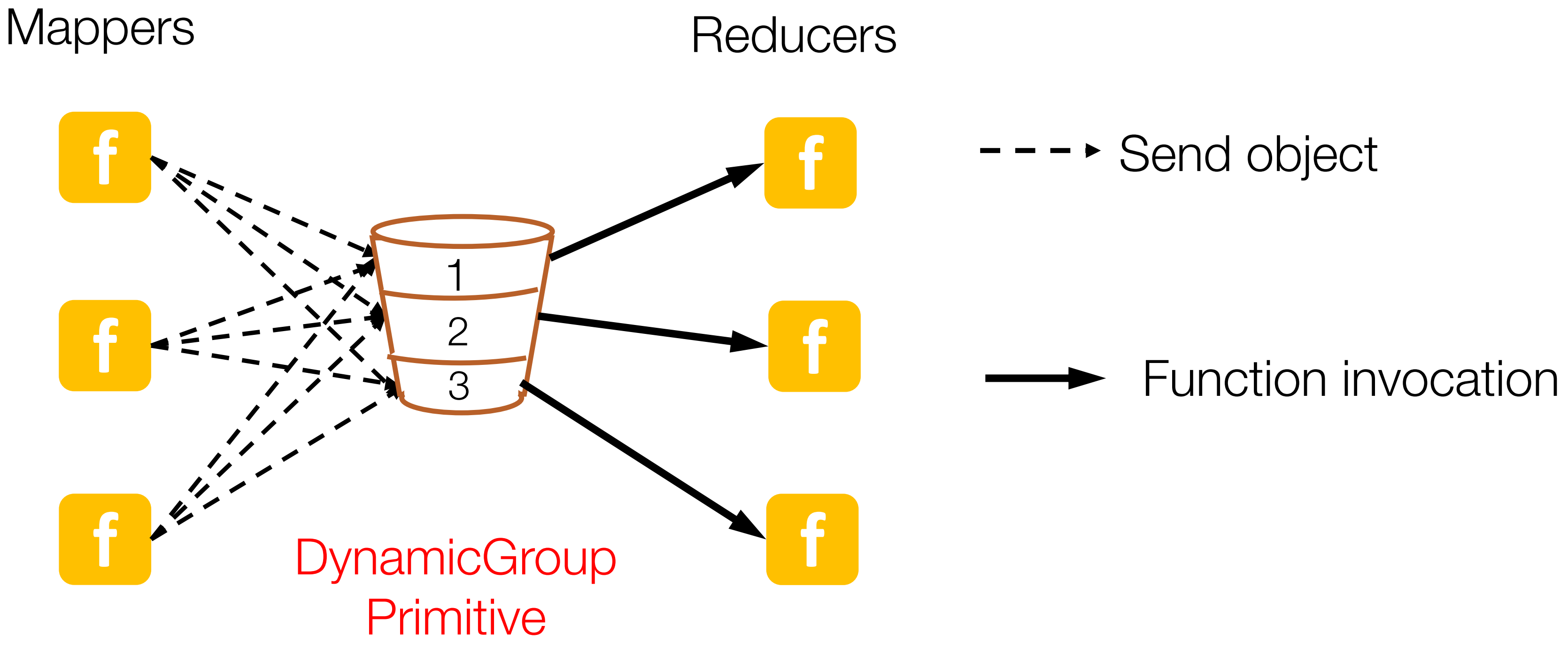
Wide applicability

- Fine-grained knowledge of data-function dependencies
- Enable more opportunities to optimize data locality and achieve high performance

Deploying real-world applications

Specify trigger conditions of data buckets with trigger primitives

Case study: MapReduce



Pheromone

A serverless platform with data-centric orchestration

Pheromone vs. AWS Step Functions (ASF)

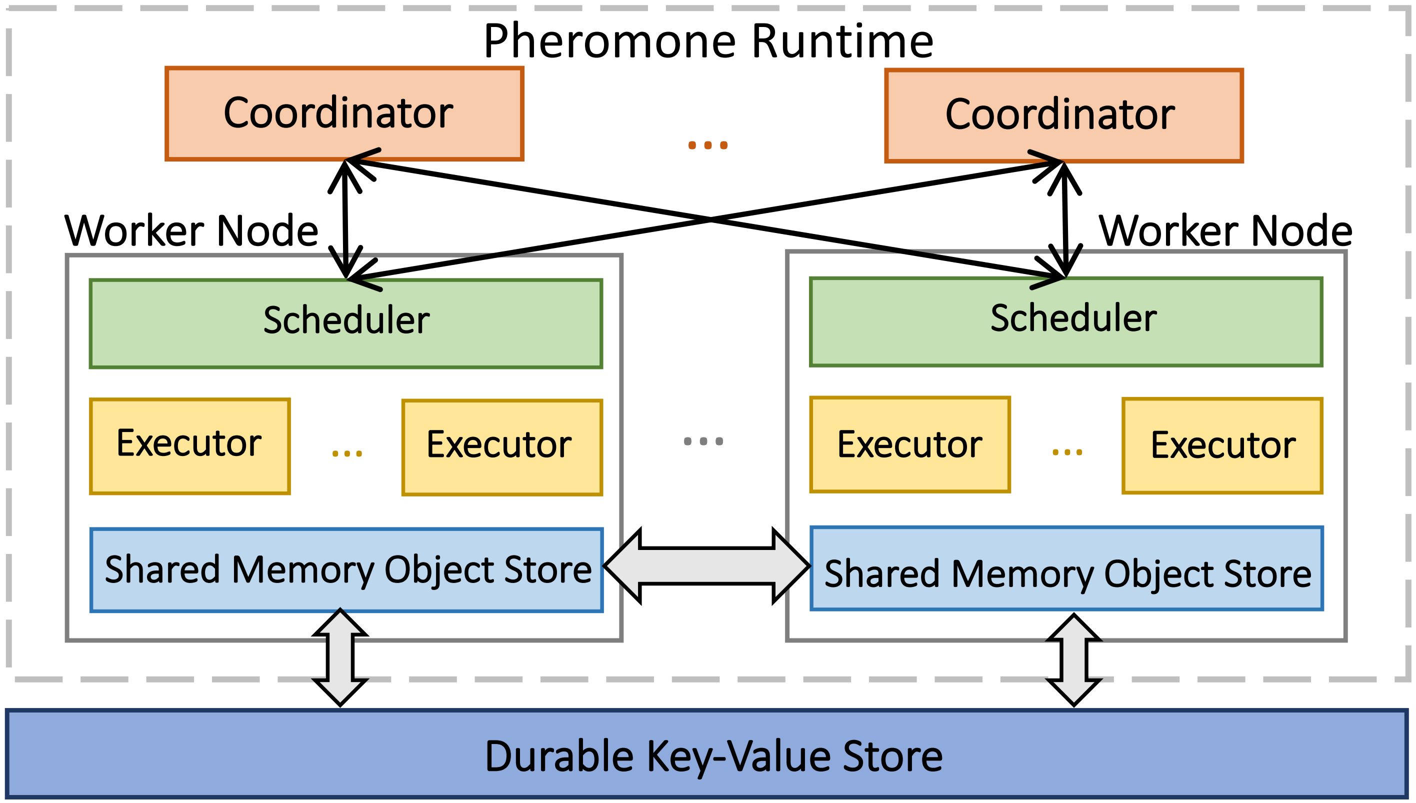
Invocation Patterns	**ASF**	**Pheromone**
Sequential Execution	Task	Immediate
Conditional Invocation	Choice	ByName
Assembling Invocation	Parallel	BySet
Dynamic Parallel	Map	DynamicJoin
Batched Data Processing	-	ByBatchSize ByTime
k-out-of-*n*	-	Redundant
MapReduce	-	DynamicGroup

```
app_name = 'event-stream-processing'
functions = ['preprocess','query_event_info','aggregate']
client.register_app(app_name, functions)

# configure the first bucket trigger.
bck_name = 'immed_bck'
trig_name = 'immediate_trigger'
prim_meta = {'function':'query_event_info'}
client.create_bucket(app_name, bck_name)
client.add_trigger(app_name, bck_name, trig_name, \
          IMMEDIATE, prim_meta)
```

Developers can implement customized primitives via an abstracted interface

Pheromone system design



Two-tier scheduling with schedulers and coordinators

- Invoke functions as locally as possible
- Sharded coordinators gather bucket status and enhance data locality in cross-node scheduling

Trade data durability for fast I/O

- Zero-copy data exchange via shared memory
- Direct data exchange between remote functions

Pheromone Evaluation

Experimental settings

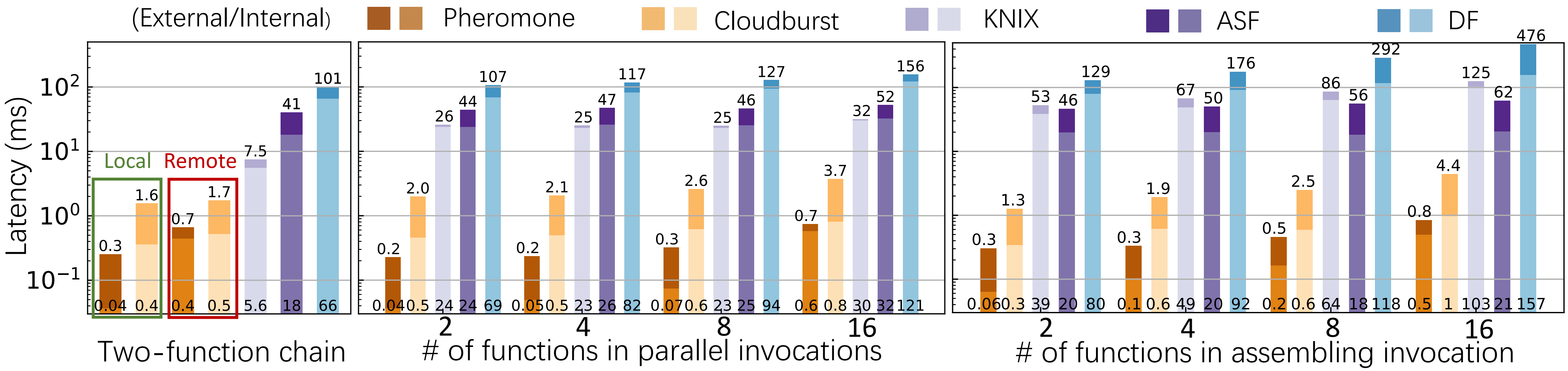
EC2 deployment

- c5.4xlarge (worker) and c5.xlarge (coordinator) on AWS EC2
- Up to 51 workers and 8 coordinators

Baselines

- Cloudburst [VLDB'20]
- KNIX [ATC'18]
- AWS Lambda and Step Functions (Express workflow)
- Azure Durable Functions

Function interaction latency



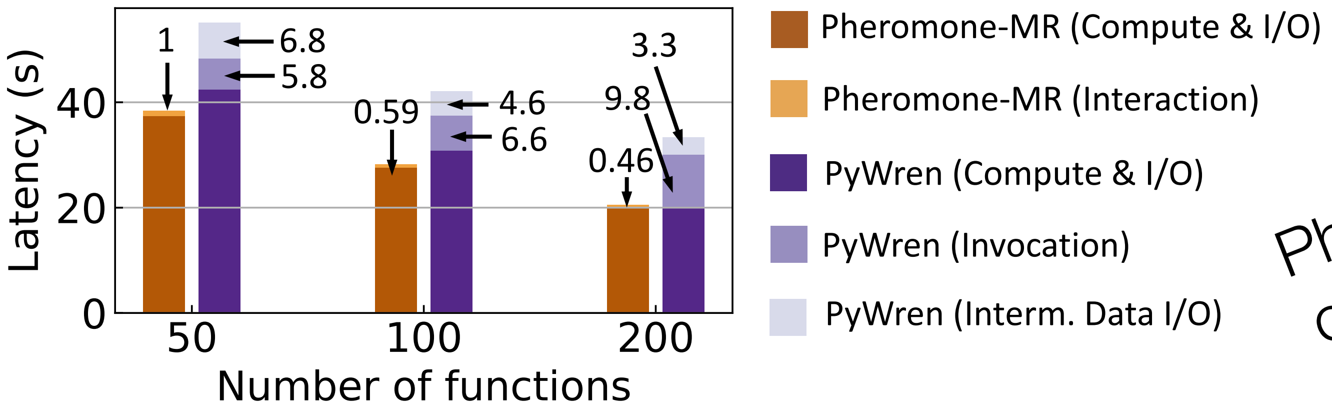
40 μs for local invocation: 10x improvement over Cloudburst

Case study: MapReduce sort

PyWren[1] atop Lambda vs. Pheromone-MR built atop Pheromone

Specifications	Pheromone-MR	PyWren
Supported operation	Map and reduce	Map-only
LOC for implementation	~500	~6k
Users need not handle data shuffle?	Yes	No

Shuffle 10 GB intermediate data



Pheromone-MR only incurs sub-second overhead, with up to 1.6x improvement

[1] E. Jonas et al. “Occupy the cloud: Distributed computing for the 99%” In Proc. SoCC, 2017

Summary

Pheromone: a serverless platform with data-centric orchestration

- Rich expressiveness
  - Easily express a rich set of workflow patterns with fine-grained data exchange
- High usability
  - No need to handle data exchange, unifying the interface with function invocation
- Wide applicability
  - High performance for both latency-critical and data-intensive applications

Thank you!

Pheromone code

Find me

I’m in the job market!