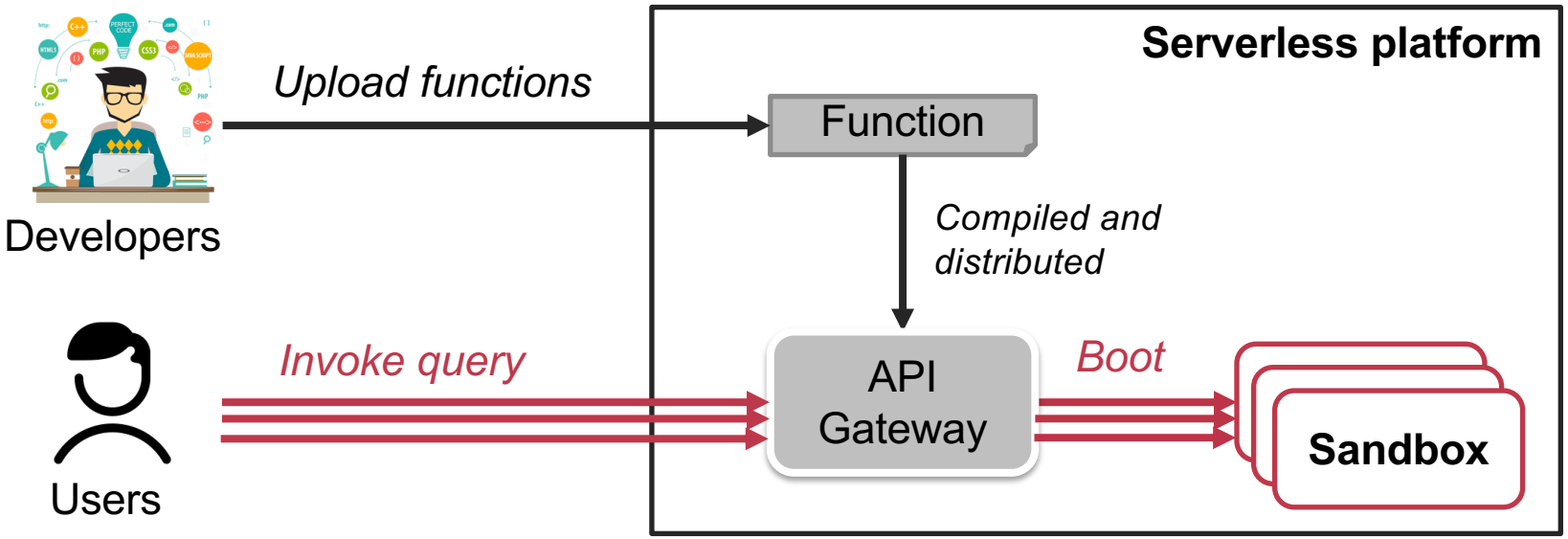
ServerlessBench: A benchmark Suite for Serverless Computing

Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang,
Ziqian Lu , Pingchao Yang, Chenggang Qin, Haibo Chen

IPADS, Shanghai Jiao Tong University
Ant Financial Services Group

Serverless Computing



AWS Lambda

Microsoft Azure

Google Functions

IBM OpenWhisk

Obstacles in Expanding of Serverless Computing

- **Serverless application developers**
  - How to architect a serverless application to be performant?
  - How to benefit in economy with serverless computing?
  - Which serverless platform to choose?
- **Serverless platform designers**
  - What metrics to emphasize in platform design?
  - Where are the performance bottlenecks of serverless platforms?

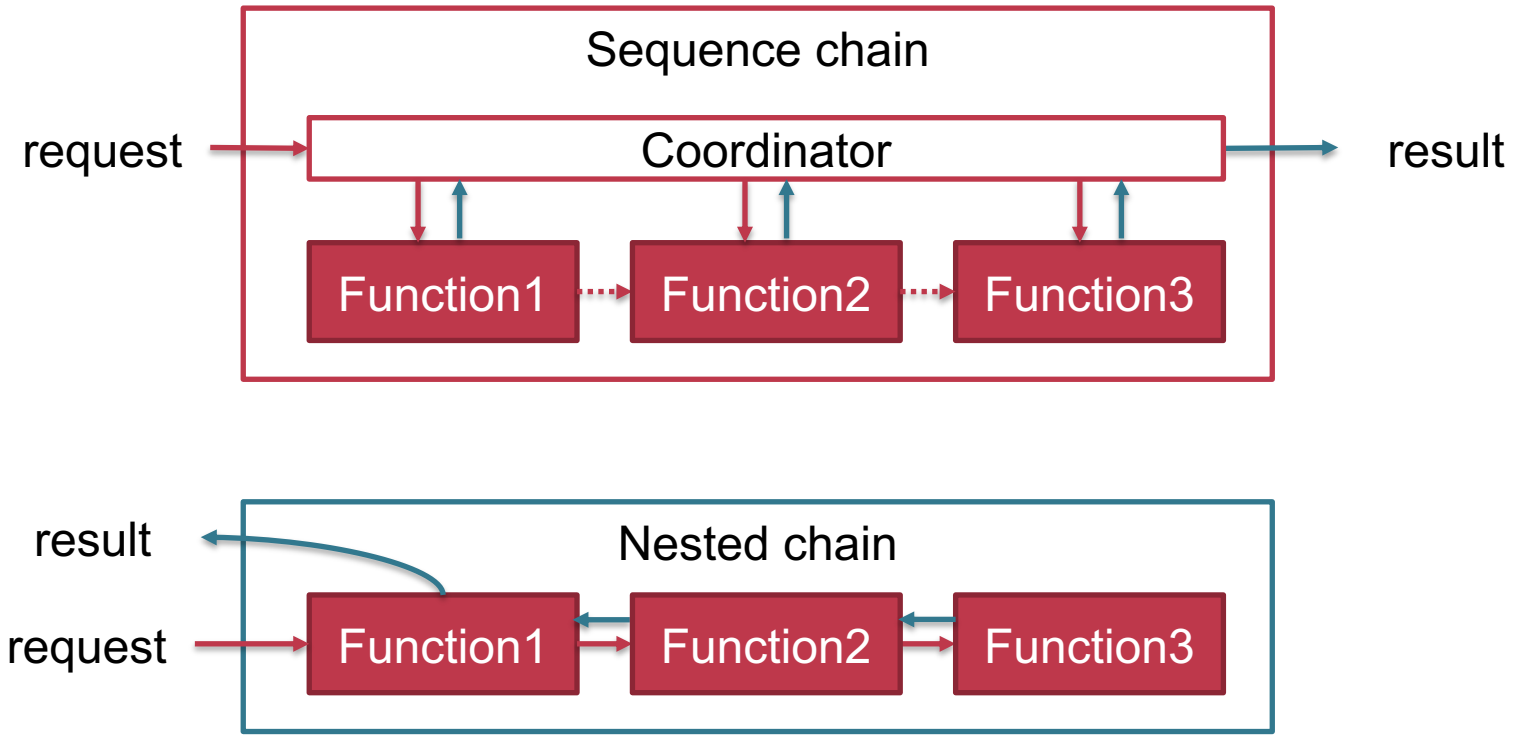
A benchmark suit that can reflect the critical metrics of serverless computing and characterize serverless platforms is necessary.

Related Works

- **AWS serverless application repository**
  - A catalog of serverless applications (589 when referenced in Sept. 2020)
  - Only simple and demonstrative functions
- **Evaluation of serverless platforms**
  - Comparing performance of certain workloads on different platforms
  - No insights for serverless platform and application design
- **Benchmark suites**
  - FuntionBench, DeathstarBench, CloudSuite, DCBench
  - Not design around serverless-specific metrics, e.g., startup performance

Serverless Metrics

- **1. Communication performance**
  - Implement complex serverless applications with function composition



Serverless Metrics

1. Communication performance

- Implement complex serverless applications with function composition
- Typical composition models: *sequence function chain* and *nested function chain*

2. Startup Latency

- Startup latency is added to each request processing time
- Short execution time of a serverless function (in seconds or milliseconds)
- Auto-scalability makes it harder to keep tail latency low
- Cold start with long latency, warm start with wasted resources

Serverless Metrics

3. Stateless overhead

- Explicit states (needed by function logic) passing using external storage services (e.g. AWS S3)
- Implicit states (e.g. JIT profile and session cache) lost that might hurt performance

4. Resource efficiency

- For users: how much resources to provision for best performance and economy benefits
- For platforms: ability to co-locate serverless functions with other workloads to improve utilization

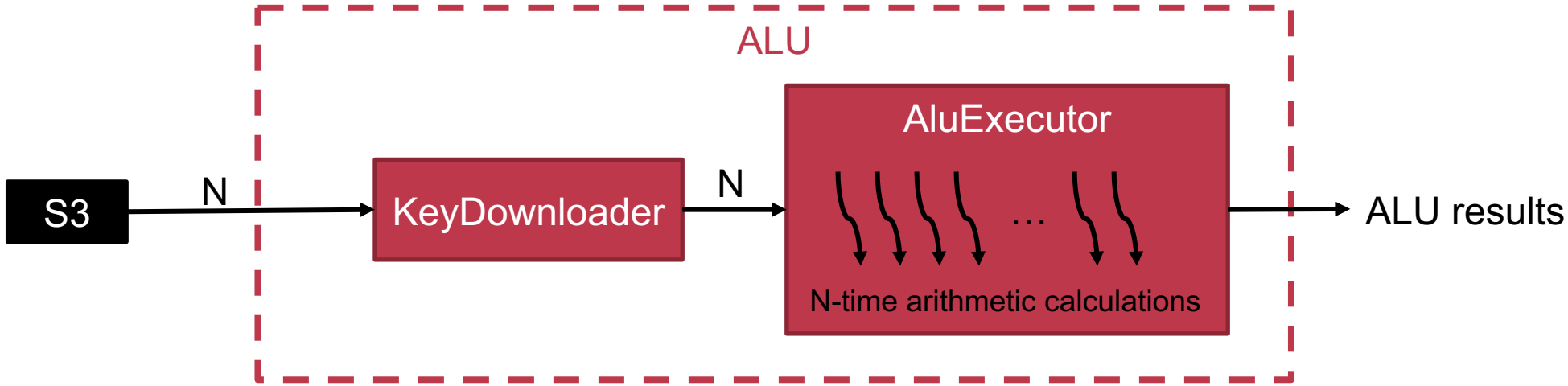
Overview

1. Serverless application with varied resource needs
2. Serverless application with parallelizable part
3. Real-world application breakdown
4. Concurrent startup
5. Stateless cost

Function Composition

1. Serverless application with varied resource needs

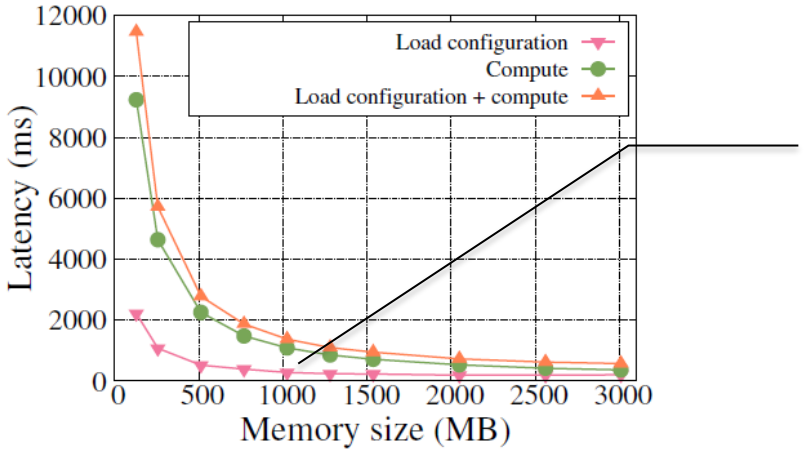
- Function execution bill proportional to provisioned resources
- Serverless platforms[1] allocate computing resources proportional to provisioned memory



[1] Including AWS Lambda, Google Cloud Functions and IBM Cloud Functions.

Function Composition

1. Serverless application with varied resource needs

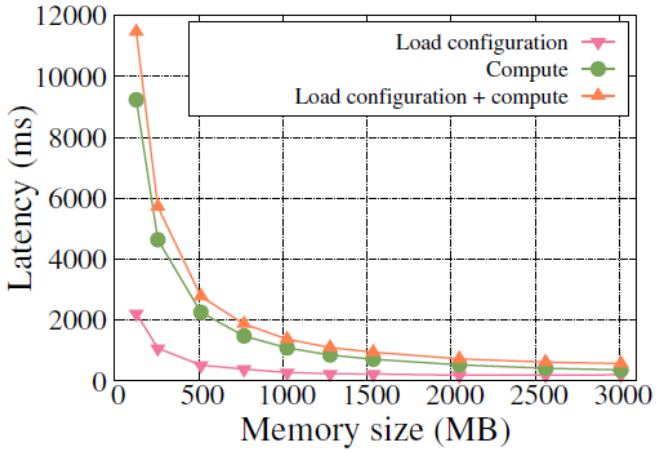


(a) Execution time.

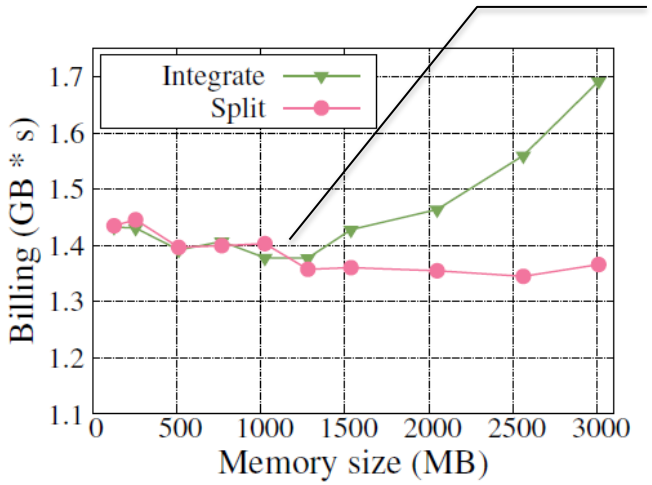
Performance for “Load configuration” phase not improve with larger memory

Function Composition

1. Serverless application with varied resource needs



(a) Execution time.



(b) Billing.

Memory allocation for “Load configuration” phase remains 1G after this point

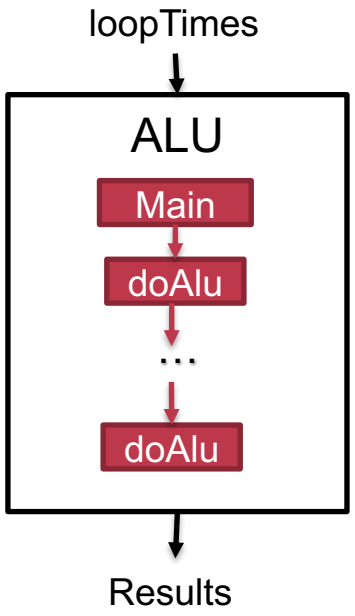
Implication: Decoupling a serverless application with varied memory needs across execution phases might save costs.

Function Composition

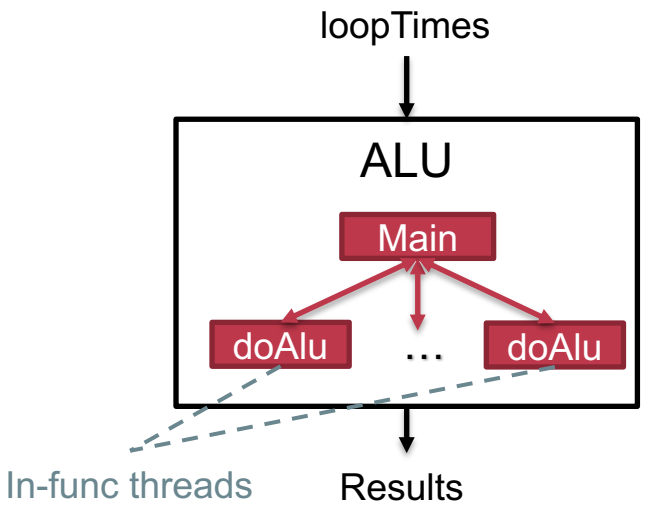
2. Serverless application with parallelizable part

Listing 1 Function code of Alu.

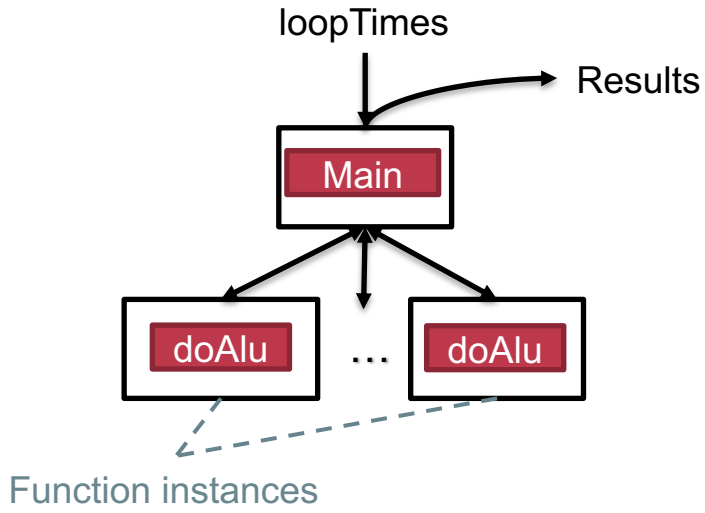
```
1: def Alu(loopTimes):
2:     result = 0
3:     for i in range(loopTimes):
4:         result += doAlu()
5:     return result
```



(a) Sequential



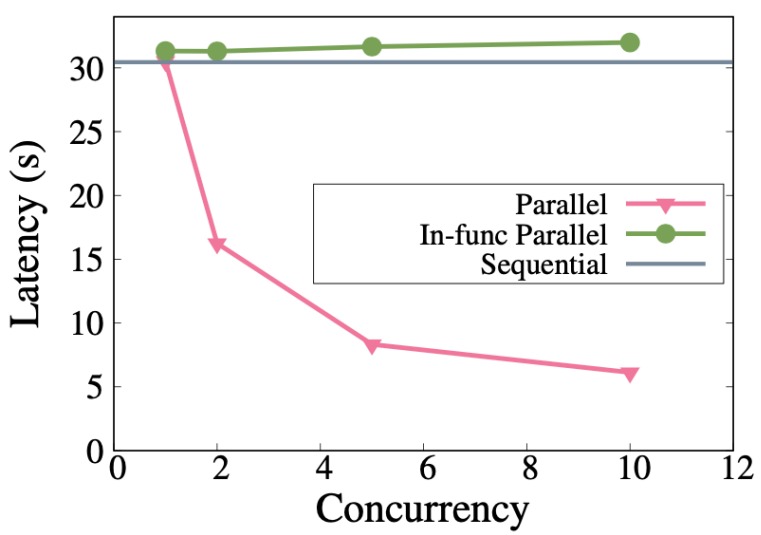
(b) In-function parallel



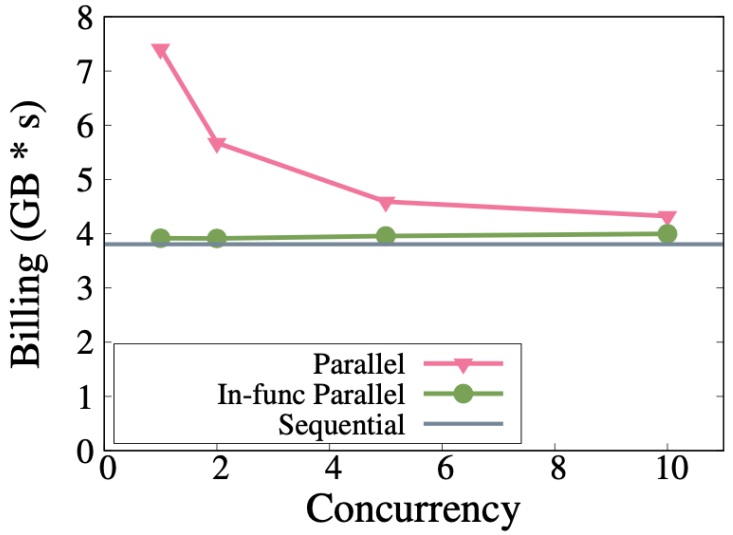
(c) Parallel

Function Composition

2. Serverless application with parallelizable part



(a) Latency.



(b) Billing.

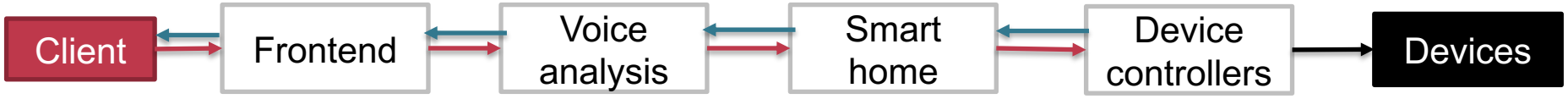
Implication: Decoupling the parallelizable part in an application might help boost the overall performance with parallel executed serverless functions.

Function Composition

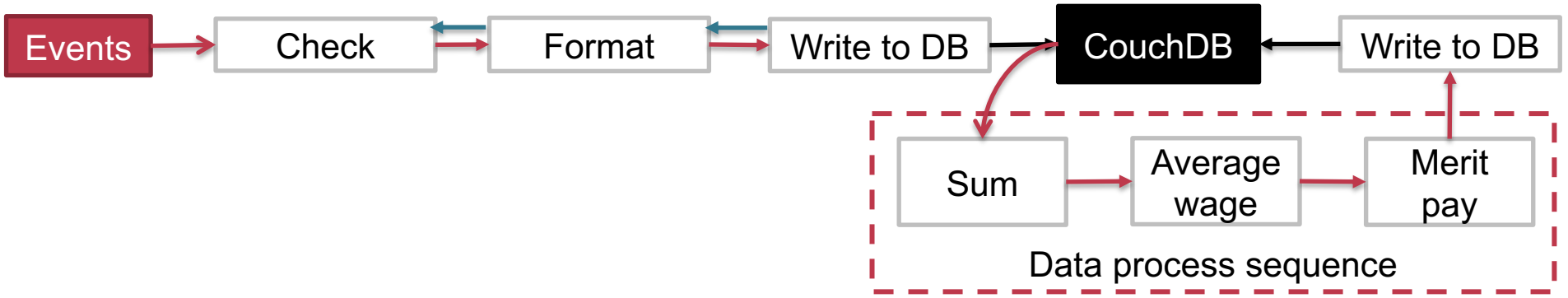
3. Real-world application breakdown

(a) Image processing (**sequence**)

(b) Alexa skills (**nested**)

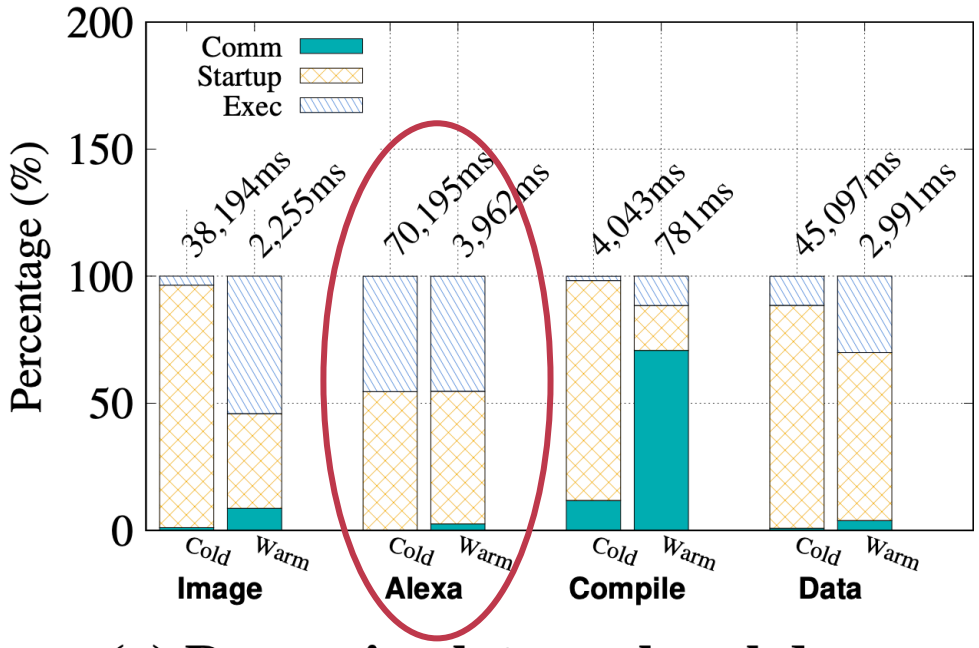


(c) Data analysis (**combined**)



Function Composition

3. Real-world application breakdown



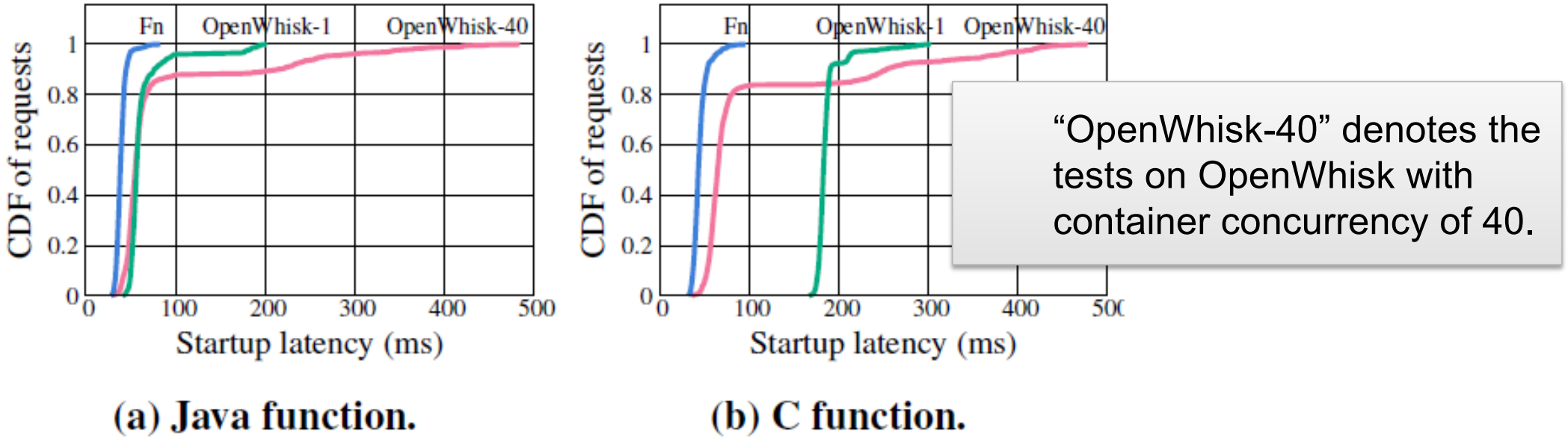
(a) Processing latency breakdown

Implication: Composition methods can significantly impact the billing in serverless computing as the execution time and overhead are charged more than once in a nested function chain.

Startup Latency

4. Concurrent startup

– 40 requests are issued concurrently



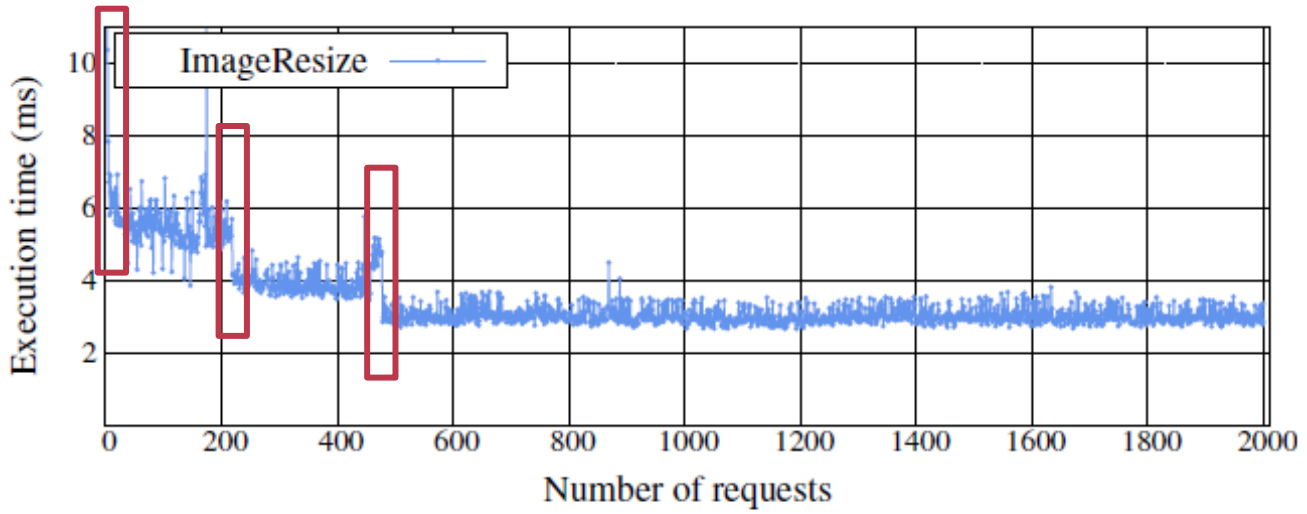
"OpenWhisk-40" denotes the tests on OpenWhisk with container concurrency of 40.

Implication: The design of supporting components for serverless platforms (such as load balancer and message queue) can affect the scalability of serverless computing.

Stateless Execution

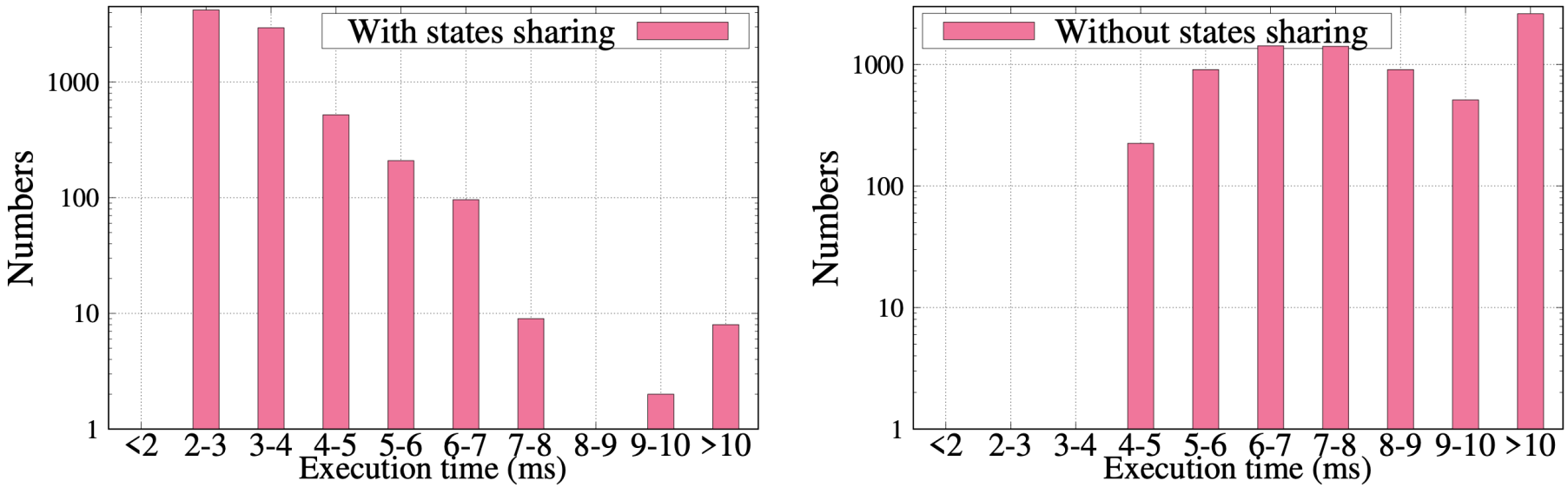
5. Stateless cost

- Explicit states: needed by application logic, handled by application developers
- Implicit states: not affect correctness, usually discarded, valuable for performance optimization



Stateless Execution

5. Stateless cost



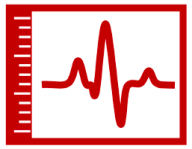
Execution time distribution on OpenWhisk.

Implication: Serverless platforms could share the implicit states, e.g., cache or JIT profile, among instances of a function to improve the execution performance.

More Details in the Paper

Test name	Metrics	Description
TC1: Varied resource needs	Communication	Resource-efficiency in function composition.
TC2: Parallel composition	Communication	Performance with parallel functions.
TC3: Long function chain	Communication	Performance of a long function chain.
TC4: Application breakdown	Communication	Latency breakdown of real-world applications.
TC5: Data transfer costs	Communication	Performance of cross-function data transfer.
TC6: Startup breakdown	Startup latency	Startup latency breakdown.
TC7: Sandbox comparison	Startup latency	Four serverless sandbox systems.
TC8: Function size	Startup latency	Different function sizes.
TC9: Concurrent startup	Startup latency	Startup latency with concurrent requests
TC10: Stateless costs	Stateless	Cost of stateless execution
TC11: Memory bandwidth	Perf isolation	Performance isolation on memory bandwidth.
TC12: CPU contention	Perf isolation	Performance isolation on CPU resource.

Conclusion



ServerlessBench

- **ServerlessBench: a benchmark suite for serverless computing**
  - Critical metrics in serverless computing is identified
  - Evaluation on existing serverless platforms
  - Serverless implications that can guide design of serverless platforms and applications
- **Open-source info:**
  - https://serverlessbench.systems/
  - https://github.com/SJTU-IPADS/ServerlessBench

IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

SHANGHAI JIAO TONG UNIVERSITY 1896

Thanks!