SAND: Towards High-Performance Serverless Computing

Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, Volker Hilt
Serverless Computing -- Function-as-a-Service (FaaS)

- Define events
- Upload function code

Events -> Execution -> Result

Developer

Function Code
The Promise of Serverless Computing for Developers

- No server management
- Continuous scaling
- Increased productivity
Overheads of Existing Platforms
Running an image processing pipeline on AWS, IBM and OpenWhisk

Image

Extract Metadata

{ ... “name”: “cats.jpg”, “resolution”: “1280x1024”, “ISO”: 400 ... }

Process Metadata

{ ... “new_name”: “cats_resized.jpg”, “new_resolution”: “640x512” ... }

Recognize Objects

{ ... “objects”: [“cat”, “cup”] ... }

Resize Image

{ “new_name”: “cats_resized.jpg”, “new_resolution”: “640x512”, “objects”: [“cat”, “cup”] }

Resized image & metadata & found objects
Overheads of Existing Platforms
Running an image processing pipeline on AWS, IBM and OpenWhisk

Average of 10 runs with ‘warm’ starts

Overheads in existing solutions can limit the benefits of serverless computing.
SAND

A high-performance serverless computing platform

Goals:
– Reduce latency for applications
– Utilize resources efficiently for platform operators
Outline

• Motivation & Goal

• Background
  • Overview of existing platforms & common practices

• SAND Key Ideas

• Evaluation
Overview of Existing Platforms

• Functions are isolated with containers

• Containers are deployed where resources are available

• Containers handle events and stay deployed until a timeout

• Functions interact via a distributed message bus
Implications of Common Practices

**Function execution & concurrency:**

1. Start a new container for every function execution (i.e., cold start)
2. Keep and reuse idle containers (i.e., warm start)
3. Concurrency: cold starts or queuing

**Function interaction:**

- Go through the distributed message bus

- long invocation latency
- resource inefficiency
- long function interaction latency
Outline

• Motivation & Goal

• Background

• SAND Key Ideas
  • Application-level sandboxing
  • Hierarchical message queuing

• Evaluation
SAND Application-level Sandboxing

Insight: Different concepts should have different fault isolation

- Stronger isolation between applications
- Weaker isolation between functions of the same application
SAND Application-level Sandboxing Operation

1) Put applications in separate containers
2) Run functions as separate processes in the same container
3) Fork new processes to handle new events

**Advantages:**
1) Fast creation of function executions
2) Low execution footprint
3) Automatic de-allocation of resources
SAND Hierarchical Message Queuing

Insight: Exploit locality of the functions

- Shortcuts for interacting functions of an application
SAND Hierarchical Message Queuing Operation

1) Run a local message bus on each host
2) Functions interact with other functions via the local message bus
3) Coordinate local bus with the global bus

**Advantages:**
1) Low function interaction latency
2) Fault tolerance & parallelism if needed
Addressing Overheads in SAND

Application-level Sandboxing

➢ Fast startup
➢ Low execution footprint
➢ Automatic de-allocation

Hierarchical Message Queuing

➢ Shortcuts for interacting functions
Outline

• Motivation & Goal

• Background

• SAND Key Ideas

• Evaluation
  • Revisiting the image processing application
  • Local message bus and function interaction latencies
  • Trade-off between idle memory cost and latency
SAND Overhead Comparison

Image processing pipeline

Extract Metadata
Process Metadata
Recognize Objects
Resize Image

Resized image & metadata & found objects

Image

Overhead Comparison

- 43% reduction in total runtime
SAND Microbenchmarks
Message Bus Access & Function Interaction Latencies

- Access to local bus is 3-5x faster than global bus
- 8.3x as fast as OpenWhisk
- 3.6x as fast as Greengrass
Idle Memory Cost vs. Latency
Exploring container timeout with OpenWhisk

Setup:
- 5 synthetic workloads
- Different burst parameters
- Call a single function

Idle memory cost: product of assigned but unused memory and the duration of assignment.

With 1 sec timeout, 18 - 33% of calls have cold starts

3.3x to 2 orders of magnitude reduced idle memory cost with no sacrifice in latency

Longer timeouts lead to high idle memory cost
SAND
High-performance serverless computing platform

✓ Fast function invocation
✓ Increased resource efficiency
✓ Short function interaction latencies

➢ Application-level sandboxing
➢ Hierarchical message queuing

Invite-only beta coming soon!