Accelerating Serverless Computing by Harvesting Idle Resources

Hanfei Yu¹, Hao Wang¹, Jian Li², Xu Yuan³, Seung-Jong Park¹

¹Louisiana State University, ²SUNY-Binghamton University, ³University of Louisiana at Lafayette
Serverless Computing

User

Develop

Code

IntelliSys Lab

25-29 April 2022 | Lyon, France
Serverless Computing

User

Develop

Code

Package

Function

IntelliSys Lab

25-29 April 2022 | Lyon, France
Serverless Computing

User

Develop

Code

Package

Function

Deploy

Serverless Computing Providers

- AWS Lambda
- Azure Functions
- Google Cloud Functions
- Apache OpenWhisk
Serverless Computing

User

Develop

Code

Invoke

Package

Function

Deploy

Serverless Computing Providers

- AWS Lambda
- Azure Functions
- Google Cloud Functions
- Apache OpenWhisk

IntelliSys Lab

25-29 April 2022 | Lyon, France
Function Static Configuration

User \rightarrow \text{Static Config} \rightarrow \text{Function}: \begin{cases} f \{ \} & \text{2 CPU cores} \\ \end{cases}
Varying Input Data Size

User -> Static Config -> Function

```
f{}
```
2 CPU cores

User -> Invoke Large data -> Invocation #1

Response Latency: 5 sec

: Busy core

: Idle core
Varying Input Data Size

User → Static Config

Function

- Invocation #1: Busy core, 5 sec
- Invocation #2: Busy core, 1 sec

IntelliSys Lab

25-29 April 2022 | Lyon, France
Harvesting & Acceleration

User

Invoke

Large data

Invocation #1

Invocation #2

Small data

Response Latency

5 sec

1 sec
Harvesting & Acceleration

User invokes

Large data

Small data

Invocation #1: Busy core

Invocation #2: Idle core

Response Latency:

5 sec

1 sec

Harvest

Reassign
**Harvesting & Acceleration**

- **User**
  - Invoke
    - Large data
  - Small data

**Invocation #1**
- Function: $f()$
  - Busy core: 5 sec
  - Idle core: 1 sec

**Invocation #2**
- Function: $g()$
  - Busy core: 3 sec
  - Idle core: 1 sec

**Response Latency**
- Reduced!
Realistic Applications

EG: email generation
KNN: K nearest neighbors
Realistic Applications

Performance stops growing when supplying more resources!

EG-L  
EG-S  
Saturation

Latency (s)

CPU cores

Latency (s)

Memory (MB)

KNN-L  
KNN-S  
Saturation

Latency (s)

CPU cores

Latency (s)

Memory (MB)

EG: email generation
KNN: K nearest neighbors
Realistic Harvesting & Acceleration

(a) CPU allocation

(b) Function response latency

EG: email generation
IR: image recognition
ALU: arithmetic logic units
KNN: K nearest neighbors
The goal of the agent is to maximize the expected cumulative rewards, regarding what action to take at each timestep, reward for the agent to take action in (RL) algorithms to learn the optimal resource harvesting and response latency. Thus, we propose to utilize reinforcement learning functions' recent resource utilization, and the greedy resource harvesting non-trivial to accurately estimate the saturation points based on Due to the volatility and burstiness of serverless computing, it is 2.4 Deep Reinforcement Learning scenario.

![Image](image-url)

Latency can be reduced with supplying harvested resources!

(a) CPU allocation

(b) Function response latency

<table>
<thead>
<tr>
<th>CPU cores</th>
<th>Saturation</th>
<th>User-defined</th>
<th>Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>EG</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>IR</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ALU</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>KNN</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Latency (s)</th>
<th>EG</th>
<th>IR</th>
<th>ALU</th>
<th>KNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

EG: email generation
IR: image recognition
ALU: arithmetic logic units
KNN: K nearest neighbors
The goal of the agent is to re-assignment strategies. As a comparison, the controller makes decisions that make decisions.

Figure 0 shows the Greedy RM accelerates the ALU by harvesting the saturation points of each function invocation.

(a) CPU allocation

(b) Function response latency

Latency can be reduced with supplying harvested resources!

Saturation User-defined Greedy

0 2 4 6 8
CPU cores

EG IR ALU KNN
Function ID

0 2 4 6
Response Latency (s)

EG IR ALU KNN
Function ID

Careful harvesting does not degrade performance

EG: email generation
IR: image recognition
ALU: arithmetic logic units
KNN: K nearest neighbors
General Rebalance Cases

Case 1

Within the same function
Both donator and receiver

Serverless Cluster

- Idle core
- Idle memory

# Function index
* Invocation index
General Rebalance Cases

Case 1

\[ f^{*1}_{\#1} \rightleftharpoons f^{*2}_{\#1} \]

Within the same function
Both donator and receiver

Case 2

\[ f^{*1}_{\#1} \rightleftharpoons f^{*2}_{\#1} \]

Within the same function
One donator and one receiver

Serverless Cluster
General Rebalance Cases

Case 1

- **f(#1)**
- **f(#1)**

Within the same function
Both donator and receiver

Case 2

- **f(#1)**
- **f(#1)**

Within the same function
One donator and one receiver

Case 3

- **f(#1)**
- **f(#2)**

Between two functions
Both donator and receiver

Serverless Cluster

- Idle core
- Idle memory

# Function index
* Invocation index
General Rebalance Cases

Case 1
Within the same function
Both donator and receiver

Case 2
Within the same function
One donator and one receiver

Case 3
Between two functions
Both donator and receiver

Case 4
Between two functions
One donator and one receiver

Serverless Cluster
Dynamic Decisions

Perspective of a serverless platform:
- Varying **functions**
Dynamic Decisions

Perspective of a serverless platform:
- Varying **functions**
- Varying **invocations** per functions
Dynamic Decisions

Perspective of a serverless platform:
- Varying functions
- Varying invocations per functions
- Varying input data per invocation
Dynamic Decisions

Perspective of a serverless platform:
- Varying functions
- Varying invocations per functions
- Varying input data per invocation
- Every invocation requires an allocation decision
Dynamic Decisions

Perspective of a serverless platform:
- Varying **functions**
- Varying **invocations** per functions
- Varying **input data** per invocation
- Every invocation requires an **allocation decision**

A series of sequential allocation decisions
Dynamic Decisions

Perspective of a serverless platform:
- Varying functions
- Varying invocations per functions
- Varying input data per invocation
- Every invocation requires an allocation decision

A series of sequential allocation decisions

Markov Decision Process (MDP)
Deep Reinforcement Learning

Perspective of a serverless platform:
- Varying functions
- Varying invocations per functions
- Varying input data per invocation
- Every invocation requires an allocation decision

A series of sequential allocation decisions

Markov Decision Process (MDP)

Deep Reinforcement Learning
Deep Reinforcement Learning
Freyr Workflow

![Diagram of Freyr Workflow]

- **Embedding**
  - Platform
  - Function
  - Alloc option 1
  - Alloc option N
  - Inflight invocations
  - Available CPU
  - Available Memory
  - Avg CPU peak
  - Avg memory peak
  - Baseline exec time
- **Scoring**
  - Score network
    - State vector $s^1$
    - State vector $s^N$
  - Softmax
    - $q^1$
    - $q^N$
  - Mean
    - $b^1$
    - $b^N$
- **Selection**
  - P(option)
  - Best Alloc Option

**Platform info...**
**Function info...**
Freyr Workflow

State information from the platform and the function
Freyr Workflow

Proximal Policy Optimization (PPO)
Freýr Workflow

Safeguard
- Filter invalid allocation options
- Return resources when detecting a potential full usage
Freyr Architecture

Controller

Database

Frontend

Func req 1

Func req N

Load Balancer

Safeguard

DRL Agent

KV Storage

Invoker

Waiting Queue

Container

Distributed Message Queue

(Pub/Sub)

(results, usage)

(CPU, mem) allocation

(state)

allocation
Frontend receives function invocations from users

Freyr Architecture

- Frontend
- Controller
  - Load Balancer
  - Database
  - Distributed Message Queue (Pub/Sub)
- Freyr
  - Safeguard
  - DRL Agent
- KV Storage
- Invoker
  - Waiting Queue
  - Container
  - CPU
  - Mem

Func req 1, Func req N

(results, usage)

(CPU, mem) allocation

(state)

allegation
Freyr Architecture

Controller collects and sends states to the DRL agent.
Freyr Architecture

Agent predicts an allocation and sends it back to controller.
Controller then forwards the function invocation with its decision to an Invoker
Freyr Architecture

Invoker executes the function
Freyr Architecture

Invoker submits the results and usage to database for further predictions.
Experiment

Setup
- 13 VMs, each with 8 CPUs and 32 GB memory
- One user client, one frontend, one controller
- 10 Worker nodes

Baselines
- Fixed RM
- Greedy RM
- ENSURE

Fixed RM: default OpenWhisk as well as in existing serverless platforms
Greedy RM: heuristic
ENSURE: Suresh, Amoghavarsha, et al.
"Ensure: Efficient scheduling and autonomous resource management in serverless environments."
(ACSOS 2020)
**Function Execution Speedup**

**Response latency**: function invocation end-to-end latency

**Slowdown**: relative performance compared to user-defined resources.

Larger than 1.0 means degradation, less than 1.0 means speedup
Response latency: function invocation end-to-end latency
Slowdown: relative performance compared to user-defined resources.
Larger than 1.0 means degradation, less than 1.0 means speedup
**Function Execution Speedup**

![Graphs showing response latency and slowdown distributions for different algorithms.]

**Response latency**: function invocation end-to-end latency

**Slowdown**: relative performance compared to user-defined resources. Larger than 1.0 means degradation, less than 1.0 means speedup.
Resource Allocation

**Slowdown**: relative performance compared to user-defined resources. Larger than 1.0 means degradation, less than 1.0 means speedup.
**Resource Allocation**

**Slowdown**: relative performance compared to user-defined resources. Larger than 1.0 means degradation, less than 1.0 means speedup.
**Resource Allocation**

**Slowdown**: relative performance compared to user-defined resources. Larger than 1.0 means degradation, less than 1.0 means speedup.
Slowdown: relative performance compared to user-defined resources. Larger than 1.0 means degradation, less than 1.0 means speedup.
Resource Allocation

Safeguard guarantees SLOs of harvested function invocations!

**Slowdown**: relative performance compared to user-defined resources. Larger than 1.0 means degradation, less than 1.0 means speedup.
Thank You

Hanfei Yu: hyu25@lsu.edu

IntelliSys Lab: https://intellisys.haow.ca