SpecFaaS: Accelerating Serverless Applications with Speculative Function Execution

HPCA 2023

Jovan Stojkovic, Tianyin Xu, Hubertus Franke*, Josep Torrellas

University of Illinois at Urbana-Champaign

*IBM Research
Serverless Computing: Why do we want it?

- Breaking large monolithic applications into many small microservices
  - Ease of programming
  - Elasticity
- Pay-as-you-go model
  - Opportunity for high resource utilization
  - Economic incentives
- AWS Lambda, Microsoft Azure, Google Cloud, IBM Cloud
Serverless Computing: How does it work?

- Frontend
- Load Balancer
- Functions
- Invoker
- Container 1: Runtime Func A
- Container 2: Runtime Func A
- Container 3: Runtime Func B
- Container 4: Runtime Func B
- Container 5: Runtime Func A
- Container 6: Runtime Func C
Real-world Applications

- Functions composed into applications with control and data dependences
- Two ways to compose application from functions
  - Explicit workflows
  - Implicit workflows
Real-world Applications

- Functions composed into applications with control and data dependences
- Two ways to compose application from functions
  - Explicit workflows
  - Implicit workflows

```python
import composer
def main():
    return composer.when('Login',
        composer.sequence(
            'ReadTemp',
            'Normalize',
            composer.when('CompareTemp',
                'TurnAir'),
            'Done'),
        'Fail')
```
Real-world Applications

- Functions composed into applications with control and data dependences
- Two ways to compose application from functions
  - Explicit workflows
  - Implicit workflows
Contributions

- Characterization of serverless environments
- Propose **SpecFaaS** – novel serverless execution model based on speculation
  - Functions execute before their control and data dependences are resolved
  - Control dependences are predicted with branch prediction
  - Data dependences are speculatively satisfied with memoization
- Average speedup 4.6X
Outline of this talk

- Characterization of Serverless Environments
  - SpecFaaS: Speculative Execution Engine of Serverless Applications
    - SpecFaaS Design and Implementation
    - SpecFaaS Key Results
  - Conclusion
Short Functions, Huge Overheads

Platform: OpenWhisk
Applications: TrainTicket

<table>
<thead>
<tr>
<th>Component</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Execution</td>
<td>24 ms</td>
</tr>
<tr>
<td>Transfer Function Overhead</td>
<td>25 ms</td>
</tr>
<tr>
<td>Platform Overhead</td>
<td>20 ms</td>
</tr>
<tr>
<td>Runtime Setup</td>
<td>200 ms</td>
</tr>
<tr>
<td>Container Creation</td>
<td>1500 ms</td>
</tr>
</tbody>
</table>
Short Functions, Huge Overheads

Function Execution
Transfer Function Overhead
Platform Overhead
Runtime Setup

2s overhead for 20ms execution!

Platform: OpenWhisk
Applications: TrainTicket

Can we minimize and/or overlap overheads?
Can we even overlap executions?
Control Dependences are Predictable

- Branches and conditional function calls create workflow divergence
- Sequence of functions highly predictable
  - Exception and error handling code rarely executed
  - Most popular sequence accounts for
    - 90% of invocations with Alibaba
    - 98% of invocations with TrainTicket
Control Dependences are Predictable

- Branches and conditional function calls create workflow divergence
- Sequence of functions highly predictable
  - Exception and error handling code rarely executed
  - Most popular sequence accounts for 90% of invocations with Alibaba
  - 98% of invocations with TrainTicket

Can we develop a SW branch predictor to pick the next function to execute early, speculatively?
Data Dependences are Rare

- Functions can communicate via remote storage
- Remote storage is not frequently updated
  - Azure Blob storage traces: only 23% writes, 66% of blobs never updated
- Reads and writes to the same location are well separated
  - Azure Blob storage traces: 96% more than 1s, 27% more than 10s
Data Dependences are Rare

- Functions can communicate via remote storage
- Remote storage is not frequently updated
  - Azure Blob storage traces: only 23% writes, 66% of blobs never updated
- Reads and writes to the same location are well separated
  - Azure Blob storage traces: 96% more than 1s, 27% more than 10s

Can we predict data dependences between functions without frequent squashing?
Data Dependences are often Predictable

- Most functions don’t read from writable storage, don’t write to storage
  - 76% for TrainTicket, 85% for FaaSChain
- Pure functions: stateless + deterministic
  - Guaranteed to produce the same outputs whenever invoked with the same inputs

```java
@Override
public mResponse queryForId(String stationName) {
    Station station = repository.findById(stationName);
    if (station != null) {
        return new mResponse<>(1, success, station.getId());
    } else {
        return new mResponse<>(0, "Not exists", stationName);
    }
}
```
Data Dependences are often Predictable

- Most functions don't read from writable storage, don't write to storage
  - 76% for TrainTicket, 85% for FaaSChain
- Pure functions: stateless + deterministic
  - Guaranteed to produce the same outputs whenever invoked with the same inputs

Can we memoize current input/output mapping and later use it for speculative predictions?
Only few types of side-effects

- Functions meant to be executed anywhere, should not carry/modify any local OS state
- 110 open-source functions: writes to remote storage, writes to local files, HTTP

CPUs are not fully utilized in the cloud

- Need to handle load spikes and be prepared for the worst-case scenario
- Alibaba Cloud: CPUs always in the range 60-80%
Side Effects not Diverse, CPUs Poorly Utilized

- Only few types of side-effects
  - Functions meant to be executed anywhere, should not carry/modify any local OS state
  - 110 open-source functions: writes to remote storage, writes to local files, HTTP
- CPUs are not fully utilized in the cloud
  - Need to handle load spikes and be prepared for the worst-case scenario
  - Alibaba Cloud: CPUs always in the range 60-80%
  - Can we waste some of the abundant idle CPU cycles in the cloud on mis-speculation?
Outline of this talk

- Characterization of Serverless Environments
- **SpecFaaS: Speculative Execution Engine of Serverless Applications**
  - SpecFaaS Design and Implementation
  - SpecFaaS Key Results
- Conclusion
SpecFaaS Overview: Executing Beyond Dependences

(a) Conventional Execution

(b) Control-only Speculative Execution

(c) Speculative Execution
SpecFaaS Overview: Squashing on Mis-speculation

(a) Control mis-speculation
SpecFaaS Overview: Squashing on Mis-speculation

(a) Control mis-speculation

- Login
- ReadTemp
- Fail

(b) Data mis-speculation

- Login
- ReadTemp
- Normalize
- CompareTemp
- TurnAir

- Normalize
- CompareTemp
- TurnAir
SpecFaaS Design: High-Level Overview
SpecFaaS Design: Sequence Table with Branch Predictor
SpecFaaS Design: Memoization Table and Data Buffer

[Diagram showing the structure of SpecFaaS design with components like Controller, Sequence Table with Branch Predictor, Memoization Tables, Function Execution Pipeline, Scheduler, Data Buffer, Remote Storage, and Parallel Workers.]

Memoization Tables

<table>
<thead>
<tr>
<th>Input Values</th>
<th>Output Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SpecFaaS Design: Memoization Table and Data Buffer

Data Buffer

<table>
<thead>
<tr>
<th>Address</th>
<th>Function $i - 1$</th>
<th>Function $i$</th>
<th>Function $i + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V</td>
<td>R</td>
<td>W</td>
</tr>
<tr>
<td>Record 1</td>
<td>1</td>
<td>1</td>
<td>Value 1</td>
</tr>
<tr>
<td>Record 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline of this talk

- Characterization of Serverless Environments
- **SpecFaaS: Speculative Execution Engine of Serverless Applications**
  - SpecFaaS Design and Implementation
  - SpecFaaS Key Results
- Conclusion
SpecFaaS: Key Results

- Average speedup 4.6X
- Tail latency reduced 2.4X
- Throughput increased 3.9X
Conclusion

- Serverless computing brings benefits but its execution is inefficient
- Propose **SpecFaaS** – novel serverless execution model based on speculation for performance
  - Functions execute before their control and data dependences are resolved
  - Control dependences are predicted with branch prediction
  - Data dependences are speculatively satisfied with memorization
  - Data Buffer buffers speculative updates and prevents them from being externalized before speculative function is committed
- Average speedup 4.6X
SpecFaaS: Accelerating Serverless Applications with Speculative Function Execution

HPCA 2023

Jovan Stojkovic, Tianyin Xu, Hubertus Franke*, Josep Torrellas

University of Illinois at Urbana-Champaign

*IBM Research
SpecFaaS: More in the Paper!

- Efficient support for implicit workflows
- Minimizing cost and frequency of mis-speculation
- Handling different side-effects
- ...

Backup Slides: FaaSChain Applications
Backup Slides:
SpecFaaS Branch Predictor Sensitivity

Average Speedup (FaaSChain):
- 100% hit rate = 5.2X
- 90% hit rate = 5X
- 70% hit rate = 4.6X
- 50% hit rate = 4X

Improvement due to squash optimization
- 90% hit rate = 1.28X
- 70% hit rate = 1.35X
- 50% hit rate = 1.43X
Backup Slides: SpecFaaS Support for Implicit Workflows

(a) Execution Workflow

(c) Memoization table

(d) Conventional Execution of an Implicit Workflow

(e) SpecFaaS Execution of an Implicit Workflow

(b) Sequence Table with Branch Predictor

(f) Data Buffer
Main challenge with SpecFaaS: it becomes expensive on mis-speculation

There are 3 options

**Option 1**: Let the mis-speculated function request (invocation) finish in the background and ignore all its global updates
- No squashing, uses precious CPU cycles

**Option 2**: Squash the function request by killing the container
- No waste of CPU cycles, expensive squash operation (stopping the container ~10s in the background + cannot reuse container for latter invocations)

**Option 3**: Squash the function request by killing the handler process
- No waste of CPU cycles, cheap squash operation (~1ms), can reuse container
Three main sources of side-effects
- Writing to global storage, writing to local files, sending HTTP requests
SpecFaaS able to deal with writes to the global storage via Data Buffer
Writing to local files → CoW for Files (intercept file syscalls)
  - For every request (invocation) we start with the initial shared files
  - As long as the request only reads from the files, it uses the original files
  - Once the request tries to write to the file, it gets its own temp copy of the file
  - When the request completes its execution discard all temporary files
Sending HTTP requests → Stall (intercept sendto syscall)
  - Once we detect a request tries to send data via socket, we stall the operation until the request becomes non-speculative
Functions can communicate over the storage when data is larger than the allowed input size defined by the FaaS platform
- FuncA producer writes to the storage, FuncB consumer reads from the storage
- If a consumer prematurely reads from the storage → need to squash it (used stale data)
- Controller can detect that a function is frequently squashed due to RAW dependence violation → introduce STALL operation
- Avoid squashing by stalling until data becomes available
  - Previous writer/producer wrote to the storage (data buffer)
  - Previous writer/producer completed its execution