SpecFaaS: Accelerating Serverless Applications with Speculative Function Execution

HPCA 2023

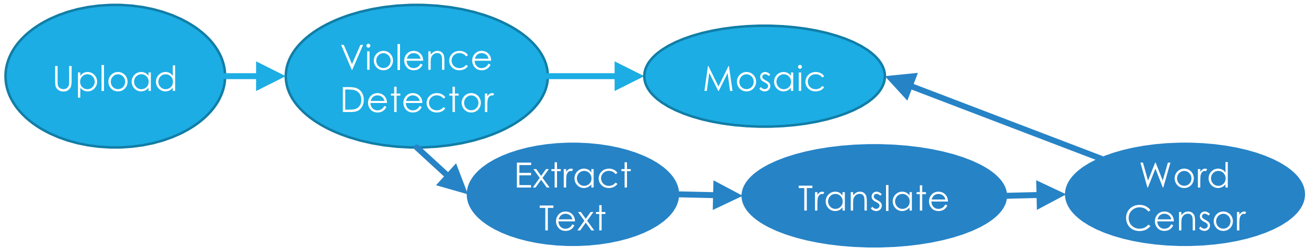
Jovan Stojkovic, Tianyin Xu, Hubertus Franke*, Josep Torrellas

University of Illinois at Urbana-Champaign

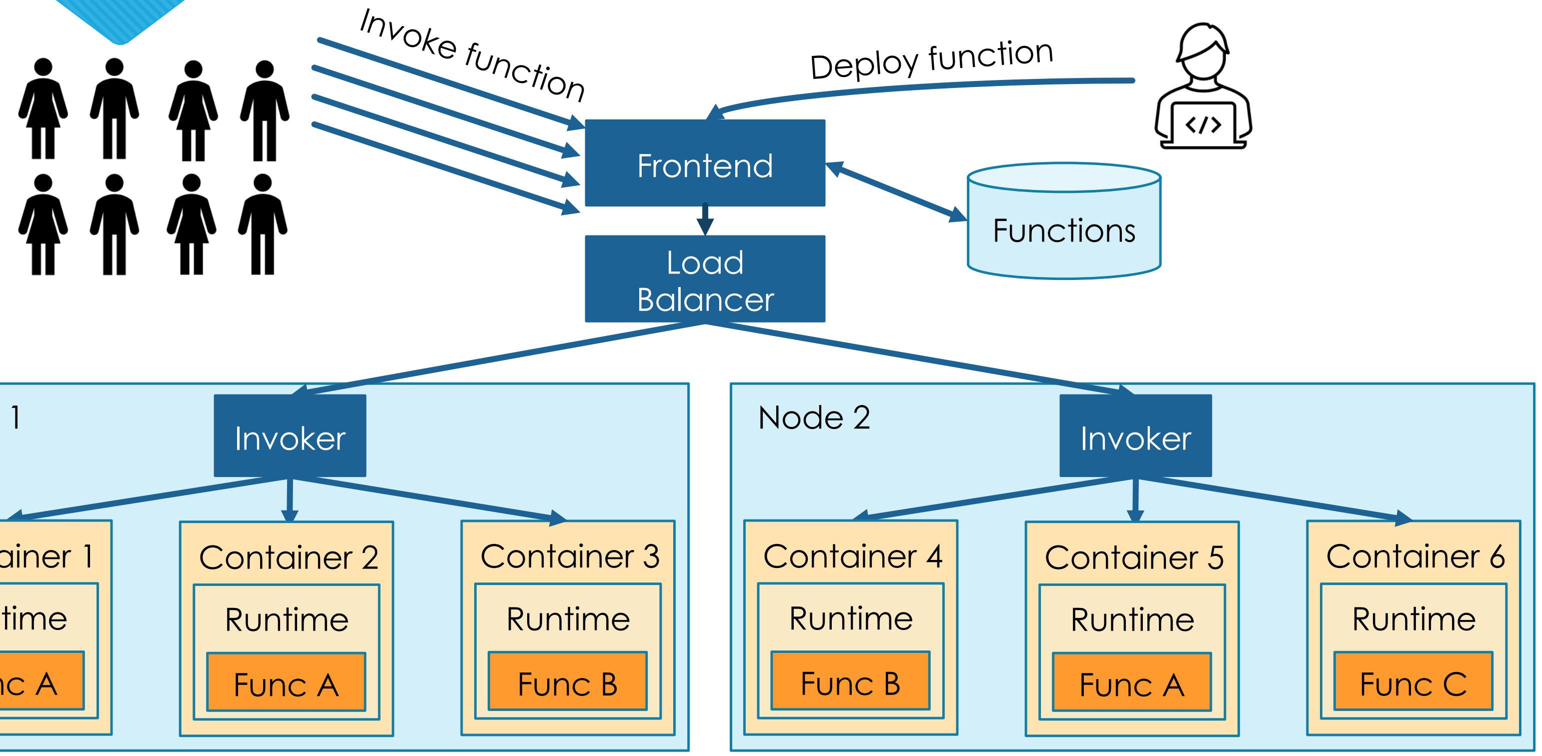
*IBM Research

Serverless Computing: Why do we want it?

- Breaking large monolithic applications into many small functions
  - Ease of programming
  - Elasticity
- Pay-as-you-go model
  - Opportunity for high resource utilization
  - Economic incentives
- AWS Lambda, Microsoft Azure, Google Cloud, IBM Cloud

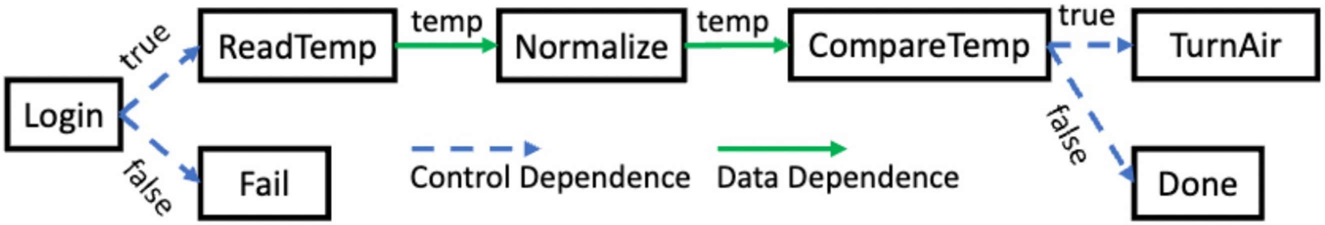


Serverless Computing: How does it work?



Real-world Applications

- Functions composed into applications with control and data dependences



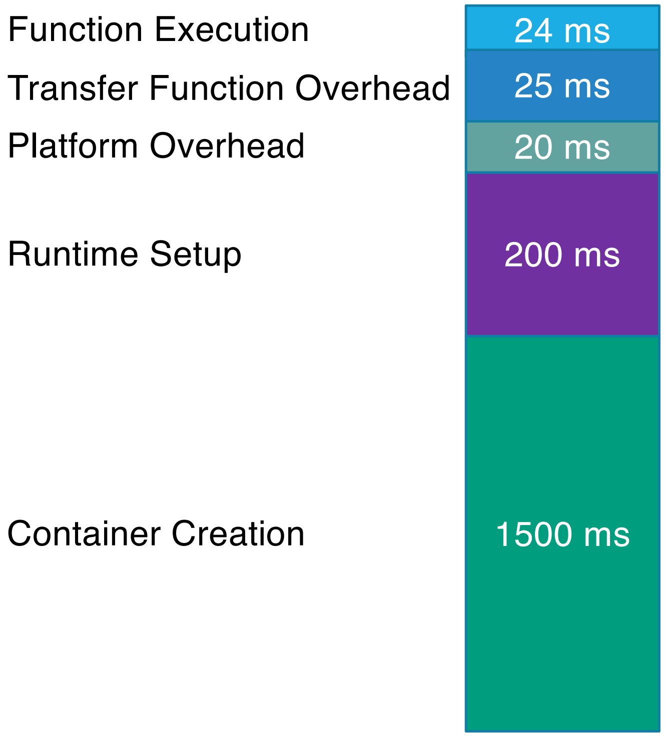
Contributions

- 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

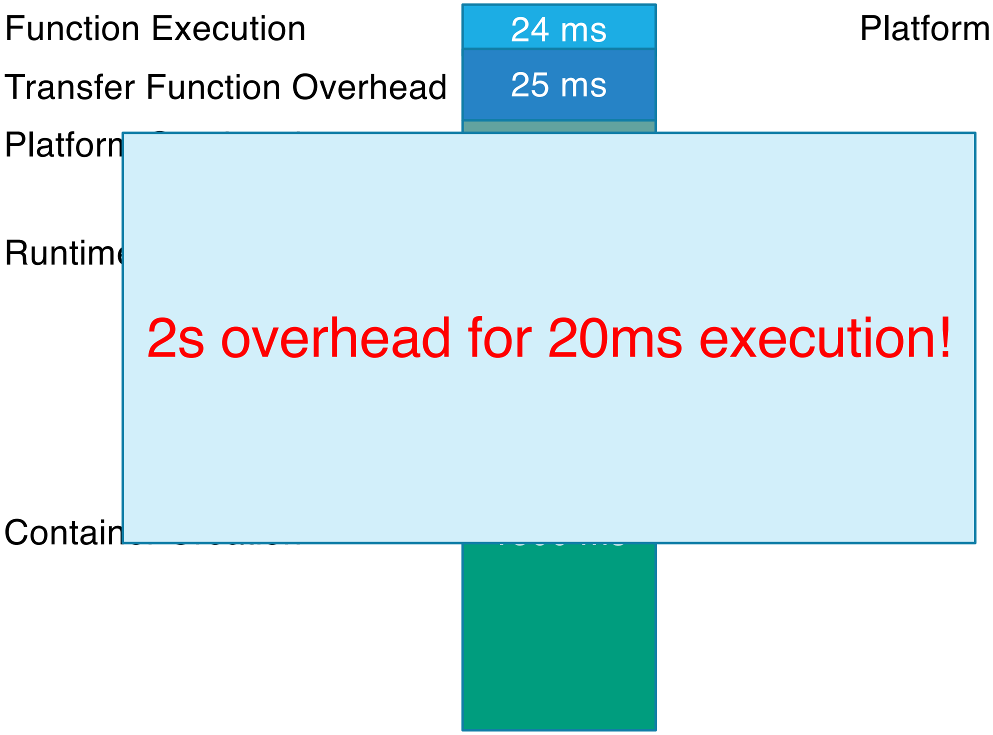
- **Characterization of Serverless Environments**
- SpecFaaS: Speculative Execution Engine of Serverless Applications
  - SpecFaaS Design and Implementation
  - SpecFaaS Key Results
- Conclusion

Short Functions, Huge Overheads



Platform: OpenWhisk

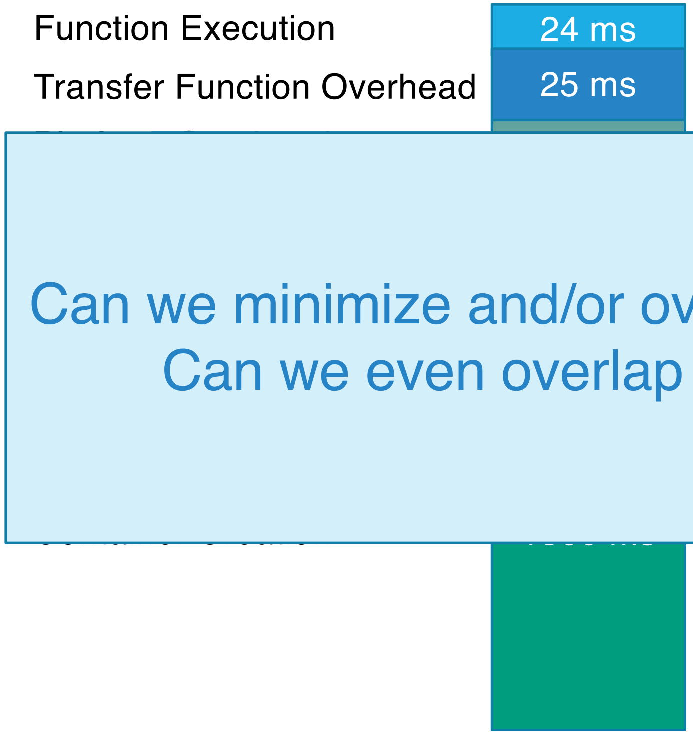
Short Functions, Huge Overheads



2s overhead for 20ms execution!

Platform: OpenWhisk

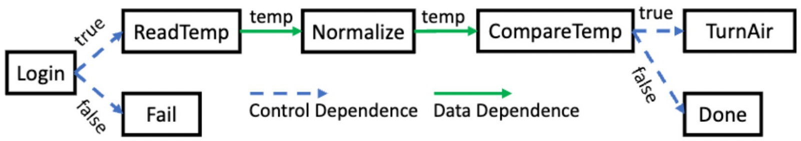
Short Functions, Huge Overheads

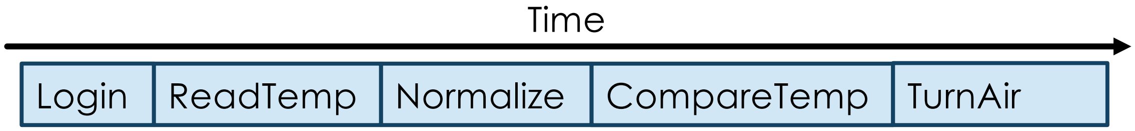


Platform: OpenWhisk

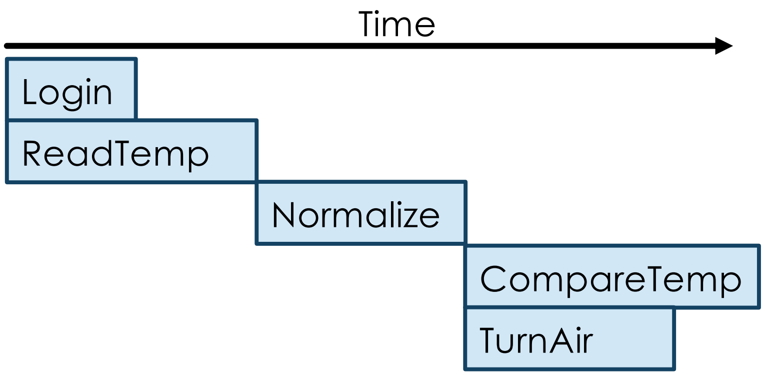
Can we minimize and/or overlap overheads?
Can we even overlap executions?

SpecFaaS Overview: Dependence Speculation

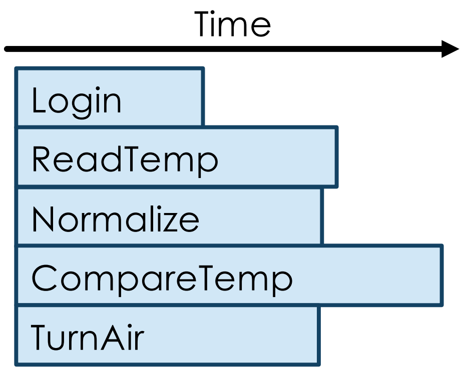




(a) Conventional Execution

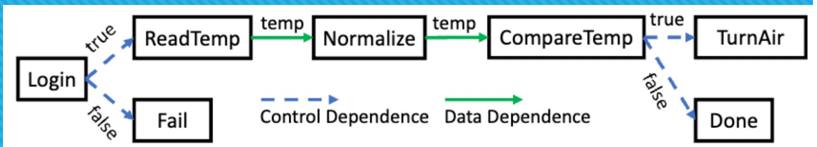


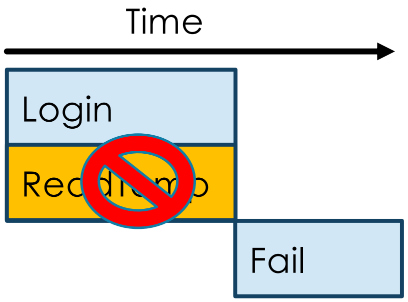
(b) Control-only Speculative Execution



(c) Data + Control Speculative Execution

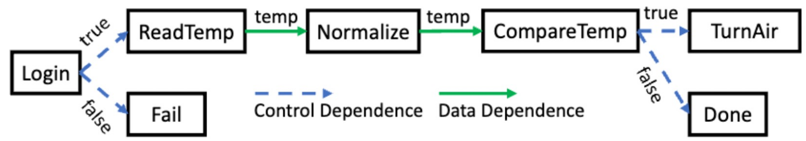
SpecFaaS Overview: Mis-speculation

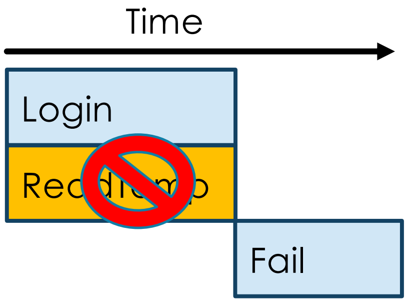




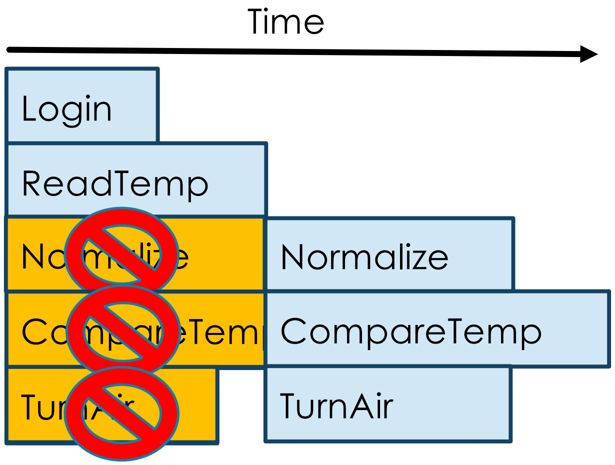
(a) Control mis-speculation

SpecFaaS Overview: Mis-speculation





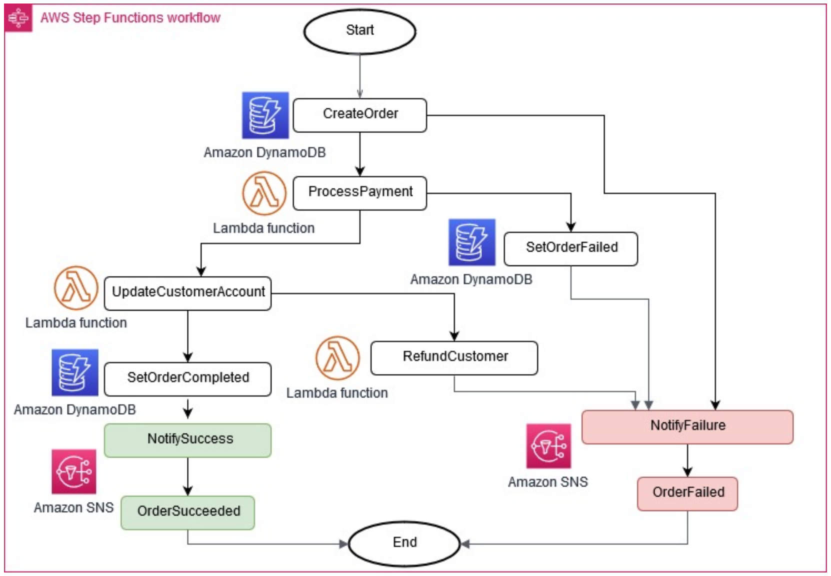
(a) Control mis-speculation



(b) Data mis-speculation

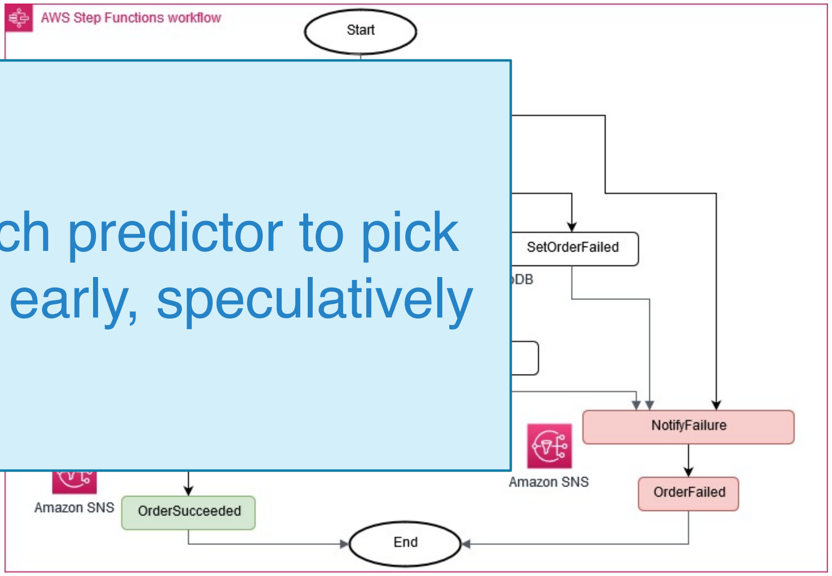
1. Control Dependences are Predictable

- 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



1. 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 po
  - 90% c
  - 98% c



We will develop a SW branch predictor to pick the next function to execute early, speculatively

2. Data Dependences are Predictable

- Most functions, given an input, generate the same output
  - They rarely depend on modifiable global state
  - 76% for TrainTicket, 85% for FaaSChain

2. Data Dependences are Predictable

- Most fu
  - The
  - 76%

We will memoize input/output value pairs for a given function and use it for speculative predictions

2. Data Dependences are Predictable

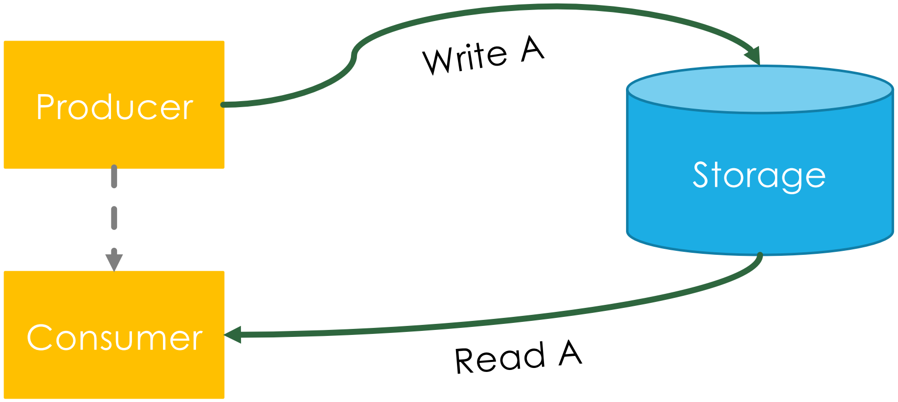
- Most fu
  - The
  - 76%

9

Many functions are pure: deterministic + no side-effects
We could completely skip execution of pure functions!

3. Communication via Global Storage is 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



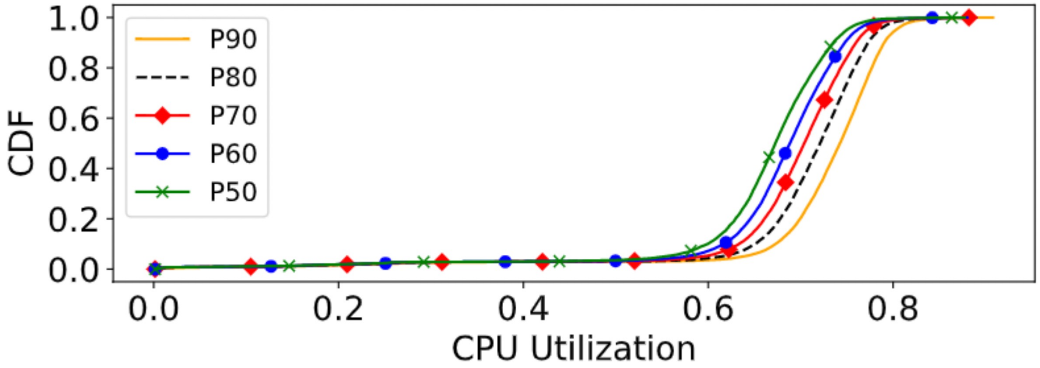
3. Communication via Global Storage is Rare

- Functions can communicate via remote storage
- Remote s
  - Azure

We will monitor implicit dependencies, but squashes will be rare

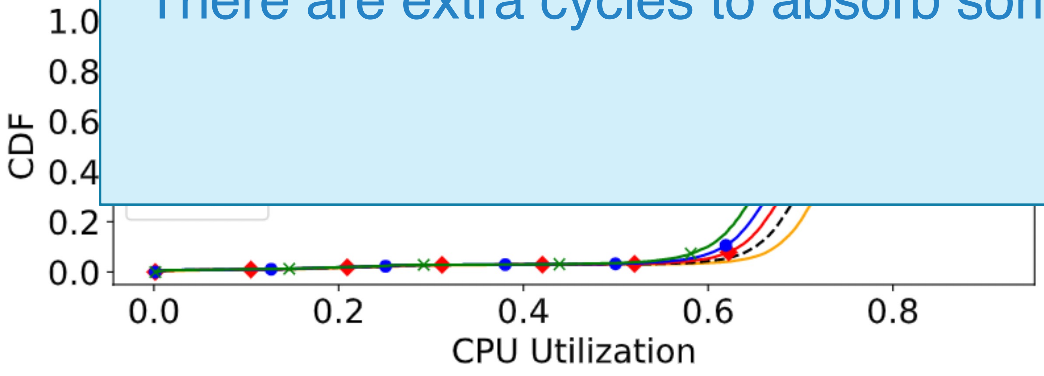
CPUs Not Fully Utilized

- 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%



CPUs Not Fully Utilized

- CPUs are not fully utilized in the cloud

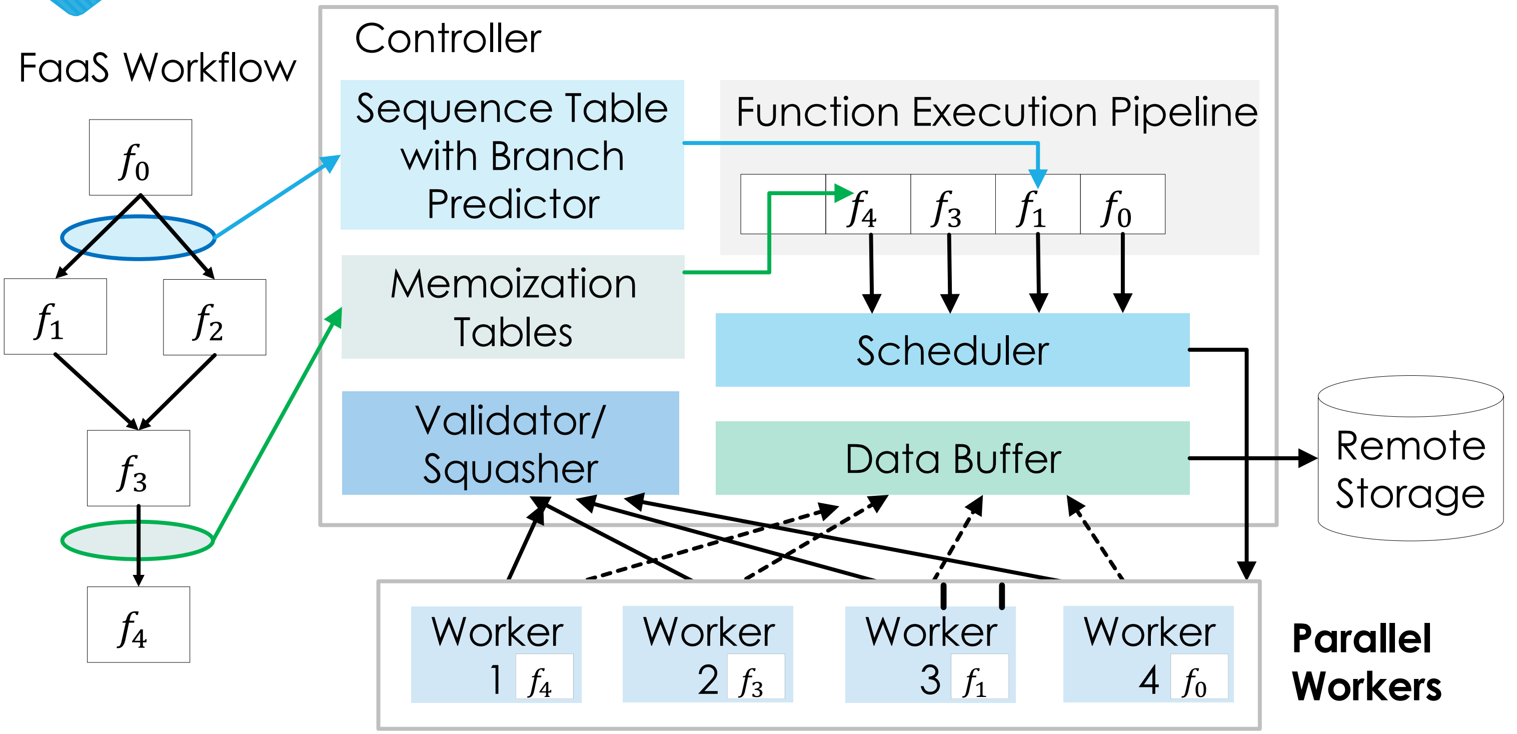


There are extra cycles to absorb some mis-speculation

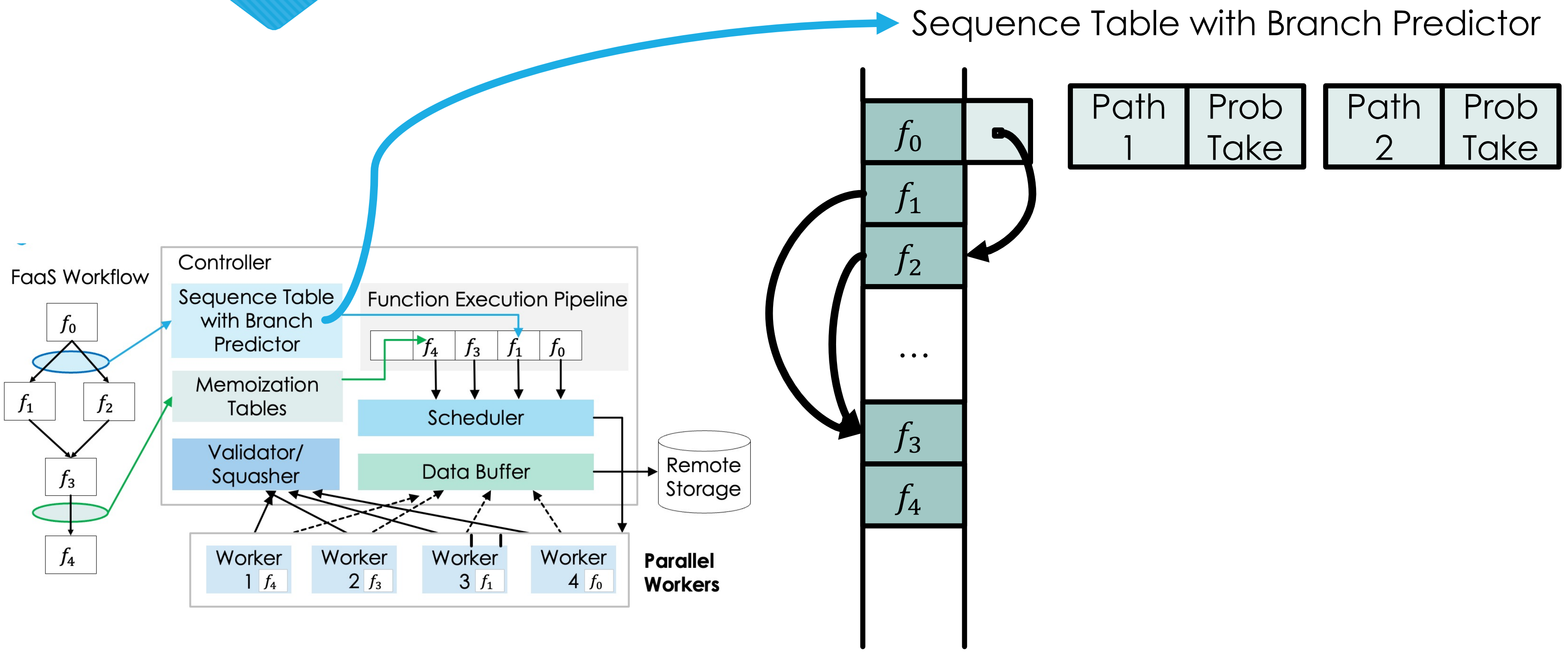
Outline of this talk

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

SpecFaaS Design: High-Level Overview

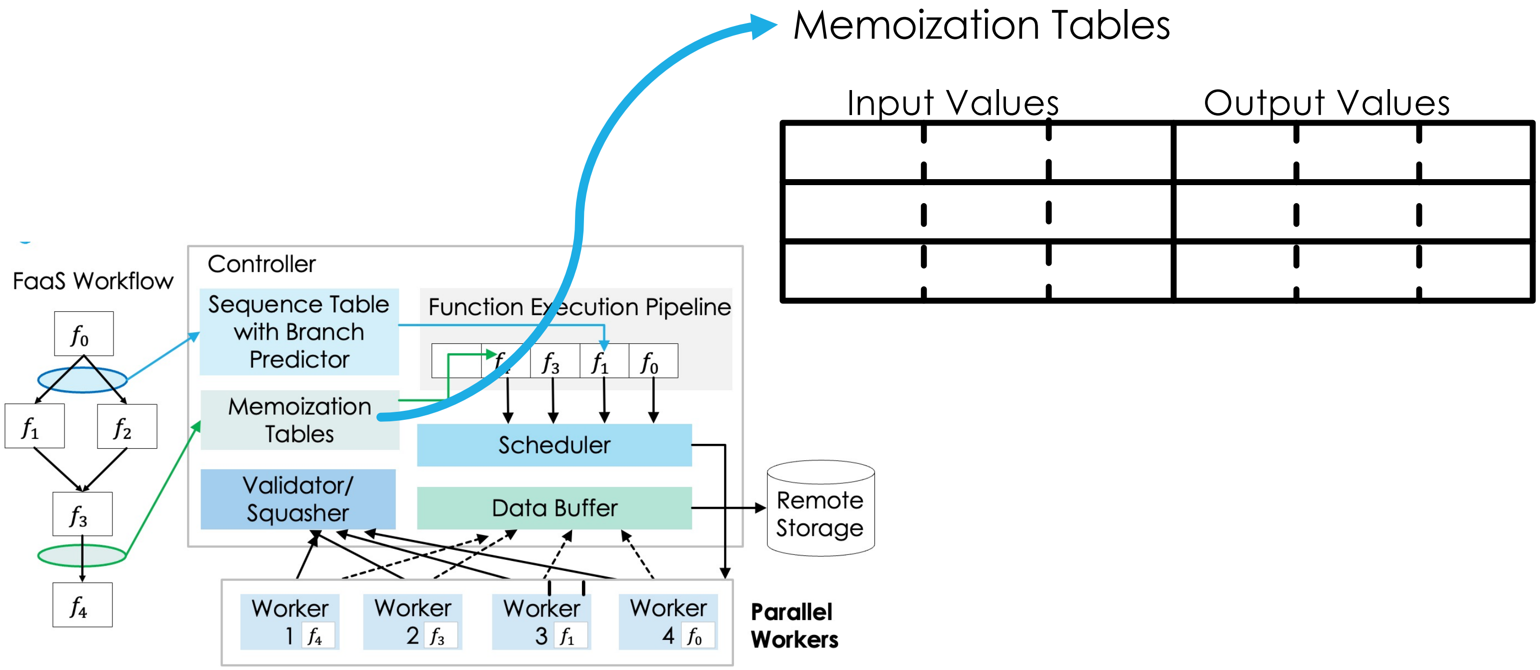


SpecFaaS Design: Sequence Table with Branch Predictor

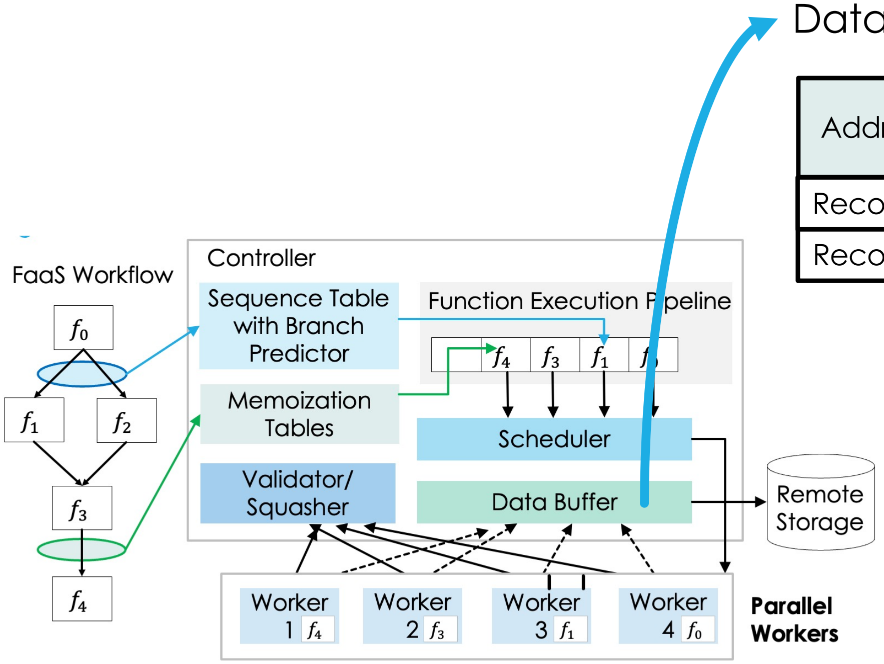


Path 1	Prob Take	Path 2	Prob Take

SpecFaaS Design: Memoization Tables



SpecFaaS Design: Data Buffer



Data Buffer

Address	*Function* $i-1$				*Function* i				*Function* $i+1$			
	V	R	W	Data	V	R	W	Data	V	R	W	Data
Record 1	1		1	Value 1								
Record 2									1	1		Value 2

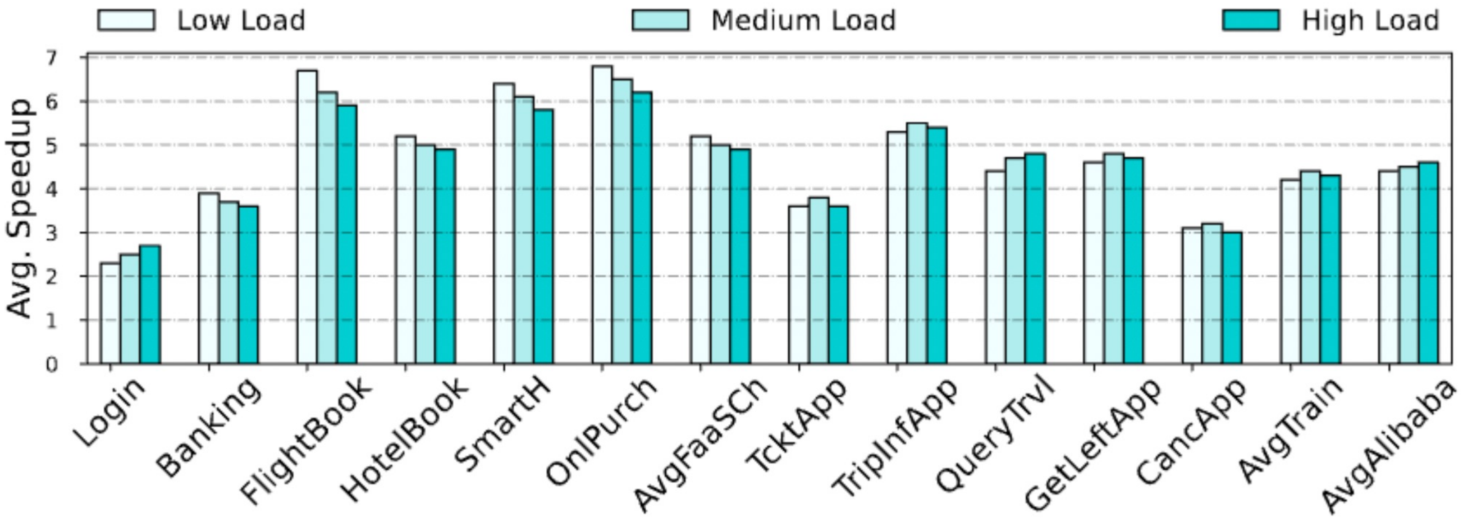
Outline of this talk

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

Experimental Setup

- 5 AMD Epyc servers, each 24 2-way SMT cores
- Platform: OpenWhisk
- Baseline: ideal sequential execution
  - All cold starts eliminated
- Various applications from three benchmark suites:
  - TrainTicket, FaaSChain and Alibaba
- 3 system loads: low, medium and high

SpecFaaS Delivers High Speedups!



Average speedup 4.6X over ideal sequential execution!

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 memoization
  - Data Buffer buffers speculative updates
- 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

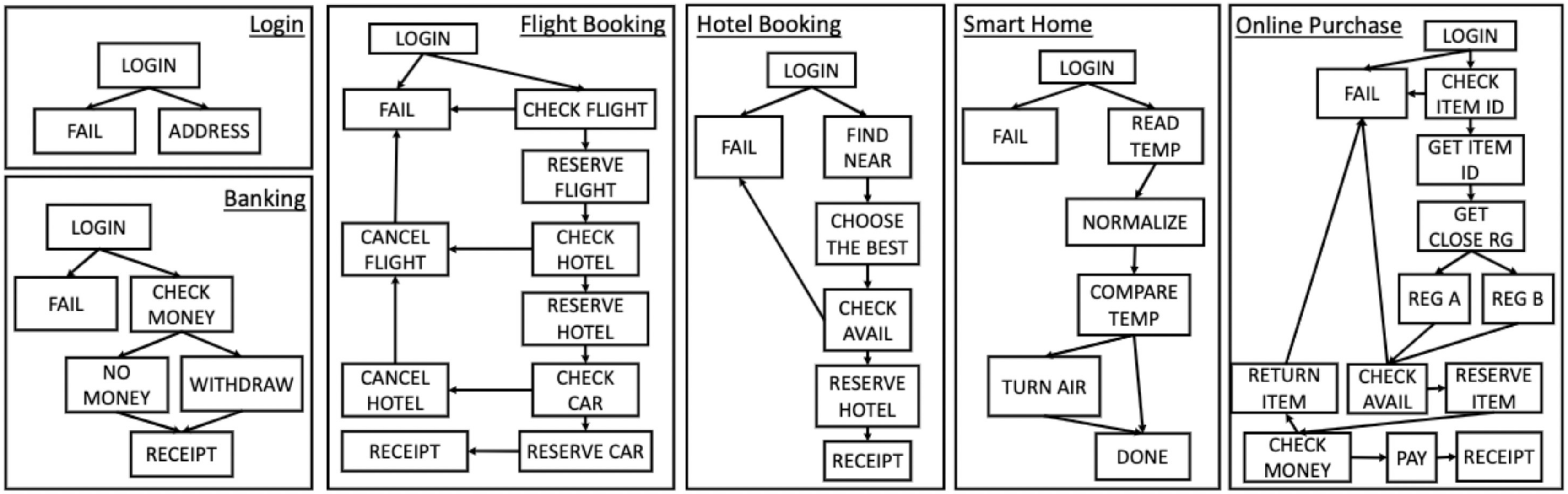
*IBM Research

Questions?

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



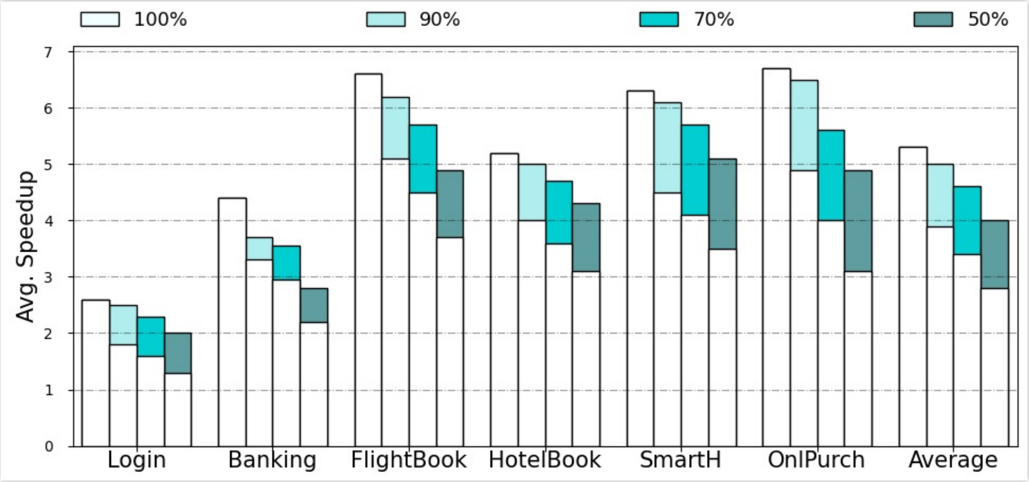
Backup Slides:
SpecFaaS Branch Predictor Sensitivity

Average Speedup (FaaSChain):

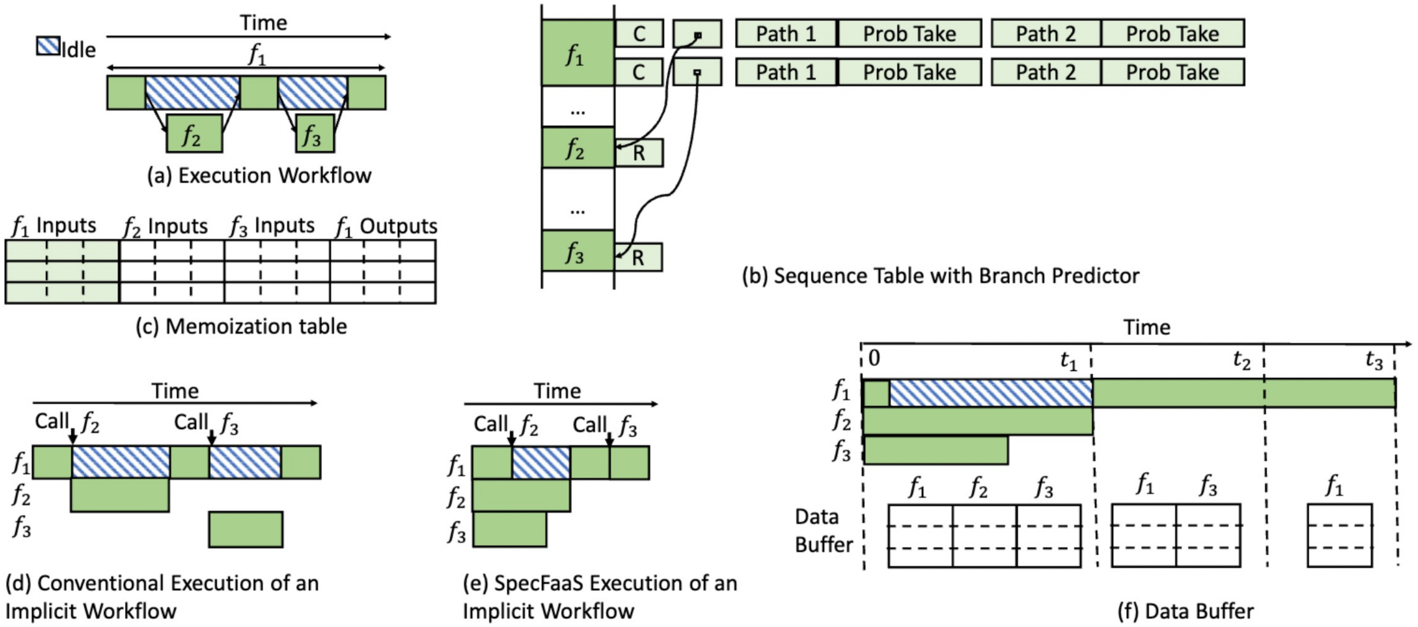
- 100% hit rate = 5.2X
- 90% hit rate = 5X
- 70% hit rate = 4.6X
- 50% hit rate = 4X

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

(b) Sequence Table with Branch Predictor

(c) Memoization table

(d) Conventional Execution of an Implicit Workflow

(e) SpecFaaS Execution of an Implicit Workflow

(f) Data Buffer

Backup Slides:
SpecFaaS Mis-Speculation Handling

- 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

Backup Slides:
SpecFaaS Side-Effects Handling

- 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

Backup Slides:
SpecFaaS Producer-Consumer Handling

- 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