fpgascheduling

Cloud-Native Scheduling for Serverless FPGAs

Bruno Scheufler
Advisors: Charalampos Mainas, Dr. Atsushi Koshiba
Chair of Distributed Systems and Operating Systems
https://dse.in.tum.de/

April 15, 2023 – July 15, 2023

Serverless computing and FPGAs

Field-programmable gate arrays (FPGAs)

Increase application performance

Serverless Platforms (Functions-as-a-Service/FaaS)

Simple deployment model

Serverless computing and FPGAs

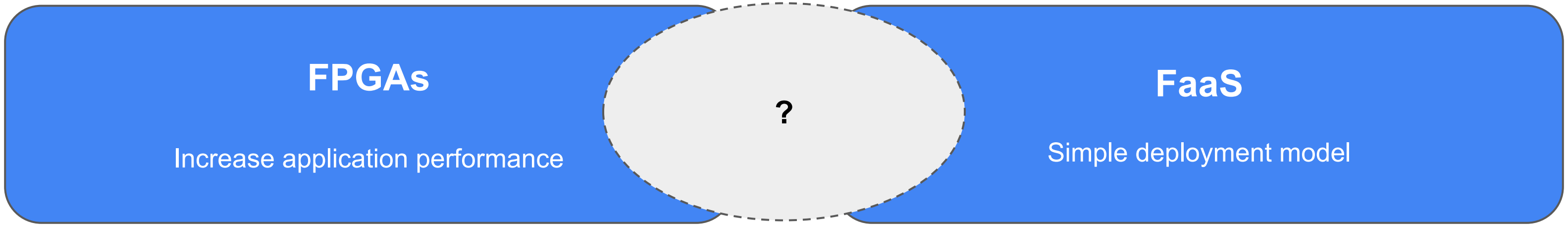
Field-programmable gate arrays (FPGAs)

Increase application performance

Serverless Platforms (Functions-as-a-Service/FaaS)

Simple deployment model

Serverless computing and FPGAs



- Existing systems allocate FPGAs statically
- Applications are heterogeneous by nature, flexible allocation required in large-scale distributed systems

No system supports FPGA acceleration for serverless workloads today

Serverless computing and FPGAs

FPGAs

Increase application performance

Orchestration

FaaS

Simple deployment model

- **Low resource utilization, inefficient workload distribution in large systems**
    - Unused compute resources lead to wasted energy and higher costs for cloud providers

Orchestration is core challenge to support FPGAs in serverless platforms

fpgascheduling

An extended Kubernetes scheduler with FPGA awareness

System Design Goals

- Utilization-based function placement ensuring application requirements
- Extensible metrics collection
- Low scheduling overhead
- Compatibility with existing systems

Outline

- ~~Motivation~~
- Design
- Implementation
- Evaluation

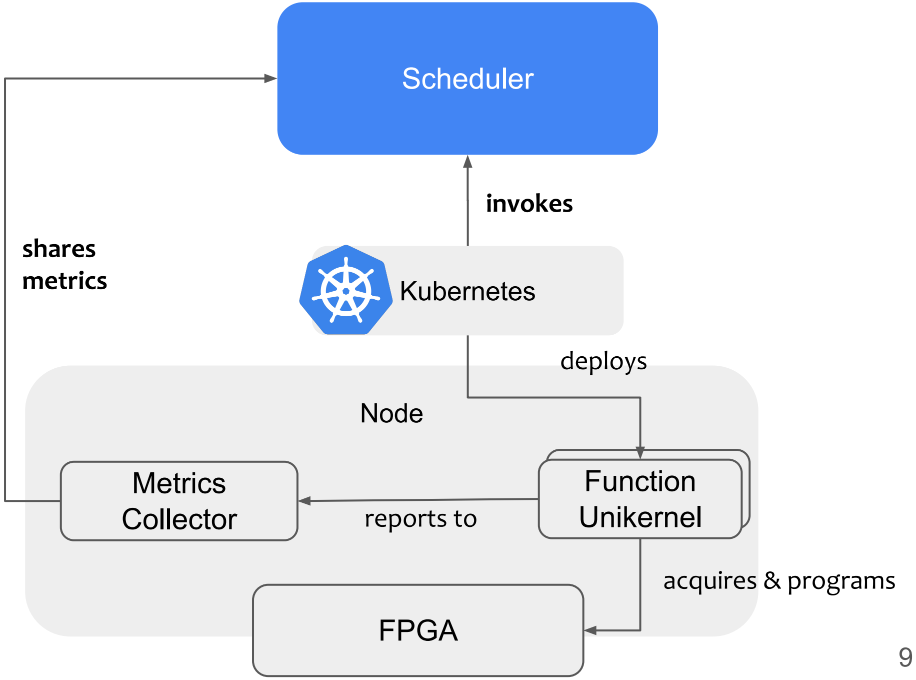
Design overview

1. Developer deploys function ahead of time
2. Kubernetes places function on node
3. Function acquires FPGA slot at runtime

Design overview

fpgascheduling introduces FPGA awareness to Kubernetes scheduling

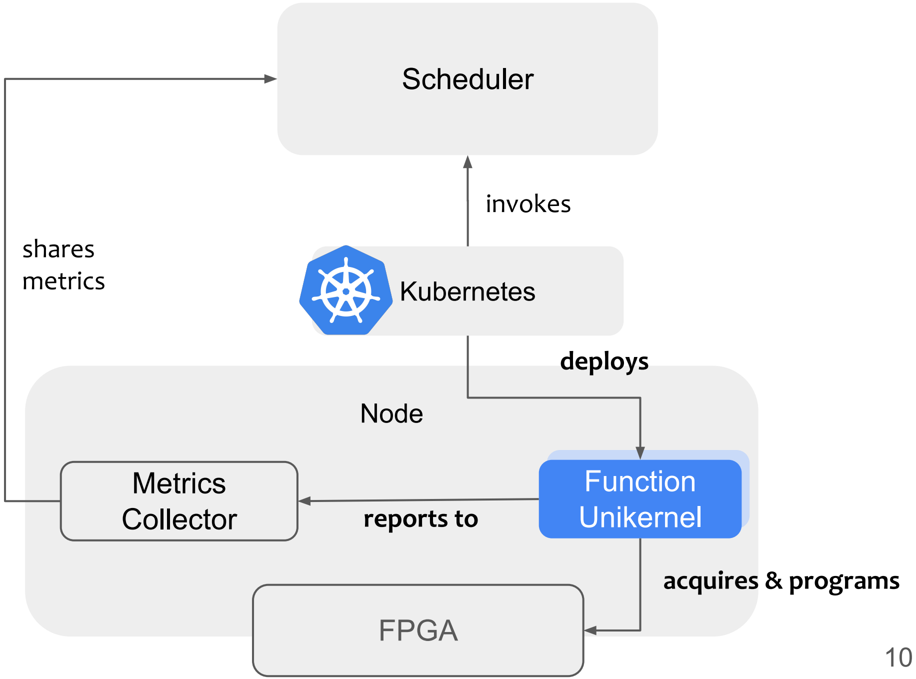
- extended Kubernetes scheduler
- Function unikernels in Funky Monitor
- Metrics collector service



Design overview

fpgascheduling introduces FPGA awareness to Kubernetes scheduling

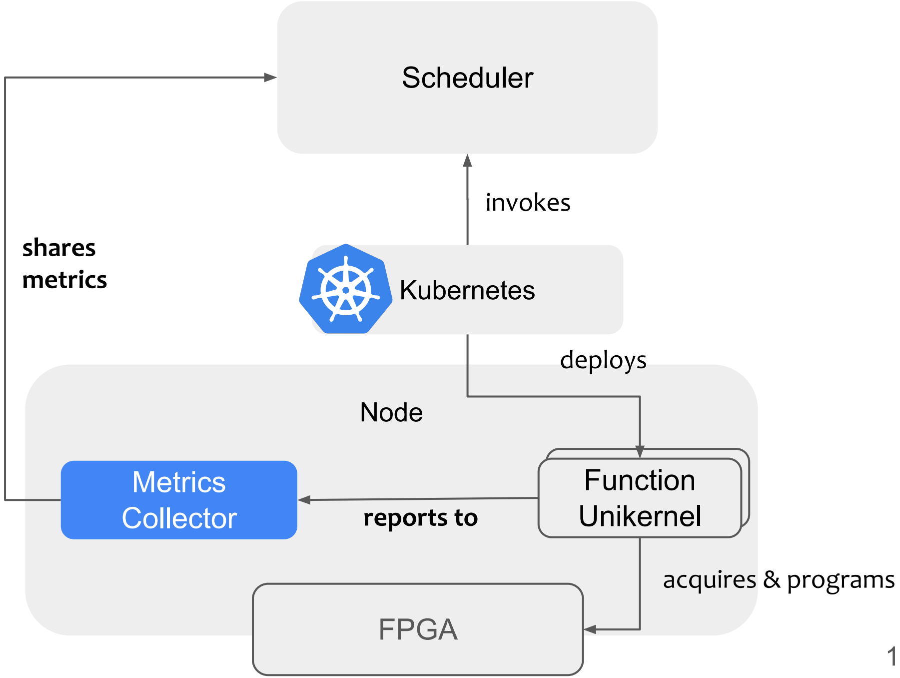
- extended Kubernetes scheduler
- Function unikernels in Funky Monitor
- Metrics collector service



Design overview

fpgascheduling introduces FPGA awareness to Kubernetes scheduling

- extended Kubernetes scheduler
- Function unikernels in Funky Monitor
- Metrics collector service



Design overview

Metrics are recorded on hypervisor level

- Measure FPGA usage and reconfiguration time in hypervisor

- Assign unique bitstream identifier to application

Design overview

Collected metrics are aggregated and sent to Kubernetes

- Metrics collector keeps state of recent requests
- Metrics are annotated to Kubernetes, out of the critical path
    - Scheduler has no overhead

Design overview

Scheduler uses FPGA metrics for function placement

- Scheduler is invoked for new pods
- Calculates weighted score for FPGA metrics for every feasible node
- Filters out infeasible nodes (different vendor preferences, etc.)

Outline

- ~~Motivation~~
- ~~Design~~
- Implementation
- Evaluation

Implementation

Metrics collector aggregates and syncs metrics

- Simple Go service
- Receives metrics from Funky Monitor (hypervisor)
- Aggregates metrics
- Stores aggregate values in Kubernetes key-value store (annotations)

Implementation

Extended Scheduler uses metrics for determining best node

- Default Kubernetes scheduler extended with new “plugin”
- Every time pod (container) must be assigned to node, plugin is invoked
- Calculate weighted average score based on metrics and scheduler weights
  - Recent usage time
  - Recent reconfiguration time
  - Bitstream locality (is bitstream configured on FPGA)

Outline

- ~~Motivation~~
- ~~Design~~
- ~~Implementation~~
- Evaluation

Evaluation

Does FPGA-aware scheduling improve overall resource utilization?

- Key questions
  - What is the impact on FPGA utilization?
  - Will there be fewer cold starts (FPGA reconfigurations)?
  - Is the scheduler fair? Does it distribute FPGA workloads uniformly?

Evaluation

Measure impact on utilization and fairness

- Key metrics
    - Median FPGA usage time
    - FPGA reconfigurations per node
    - FPGA utilization dispersion between nodes
    - Cold start percentage

Evaluation

Production traces from Microsoft Azure

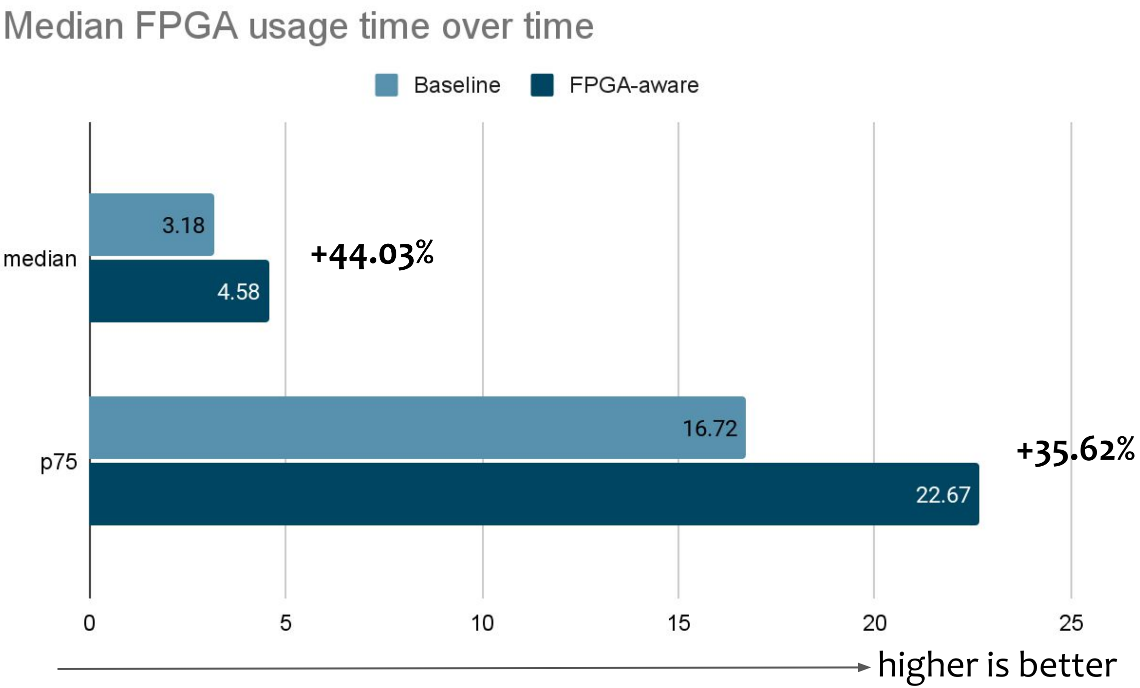
- Trace-based simulation using discrete-event simulation
- Real-world function invocation data set
    - 1.9m requests over span of two weeks
    - 119 applications, 424 functions

Evaluation

- Assumptions
  - 10 nodes, 1 FPGA per node, 4 slots per FPGA, 10ms reconfiguration time
  - Prefer recent FPGA usage in scheduler weights
  - Assume **50%** of all invocations **longer than 10ms** spend **75%** of their duration on FPGAs

Evaluation

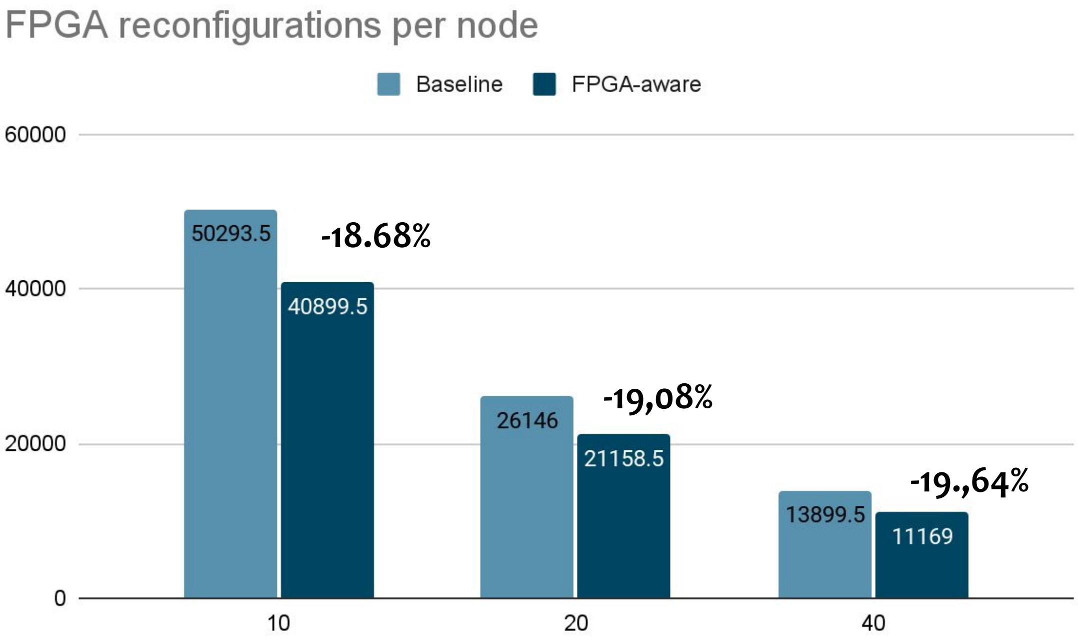
Median FPGA usage time increased up to 44.03%



Evaluation

FPGA reconfigurations per node decreased up to 18.68%

lower is better



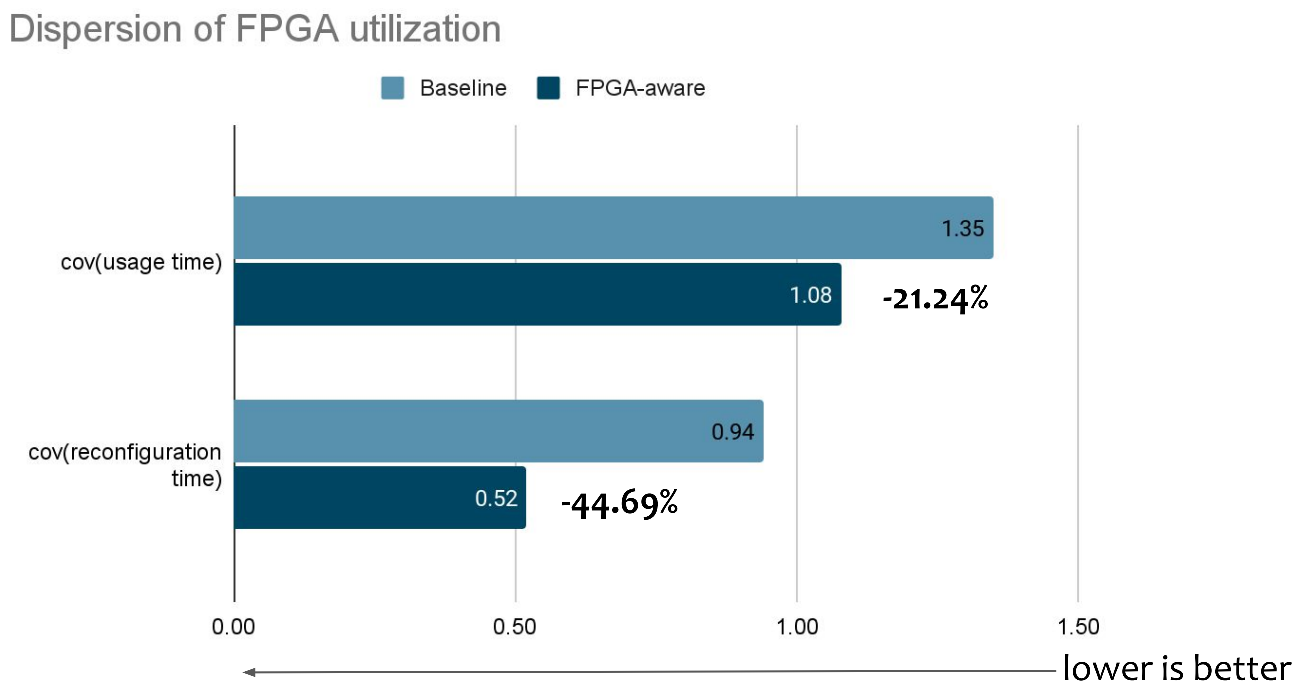
Evaluation

Cold starts reduced by up to 18.89%

	Cold start %		
Nodes	Baseline	FPGA-aware	Difference in %
10	25.51%	20.69%	**-18.89%**
20	26.71%	21.39%	**-19.92%**
40	27.57%	22.60%	**-18.03%**

Evaluation

Dispersion in usage time reduced by up to 21.24%



Summary

- **Current serverless platforms do not support FPGAs for acceleration**
    - Utilization-based scheduling is important for uniform load distribution
- **fpgascheduling extends Kubernetes orchestrator with FPGA awareness**
    - Metrics collected and reported per node using new Metrics Collector service
    - Scheduler considers recent usage and other factors
- **Simulator benchmarks yield potential improvements**
    - Higher median FPGA utilization, lower reconfigurations per node, fewer cold starts, more uniform workload distribution/higher fairness