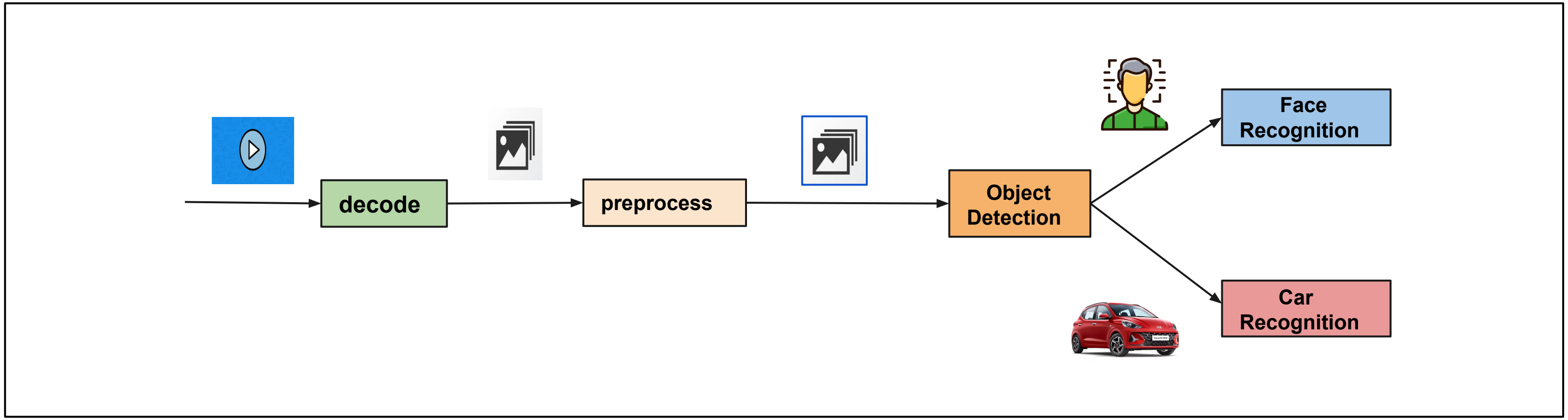
Resource Management, Data Handling and GPU Support for Serverless Workflows

By Anubhav Jana (22M2109)

Guide : Prof Purushottam Kulkarni

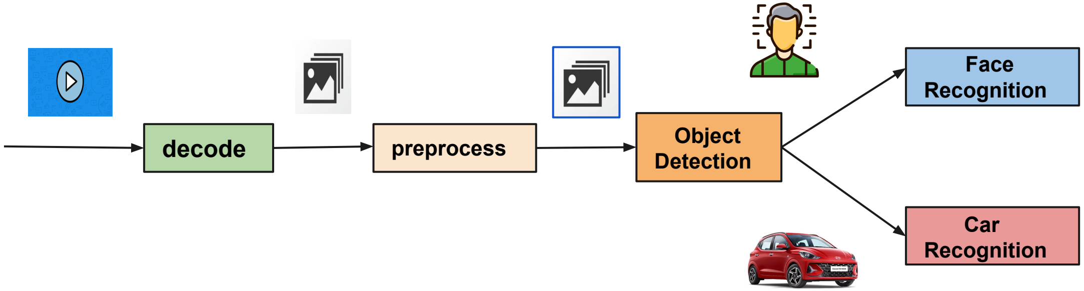
Context

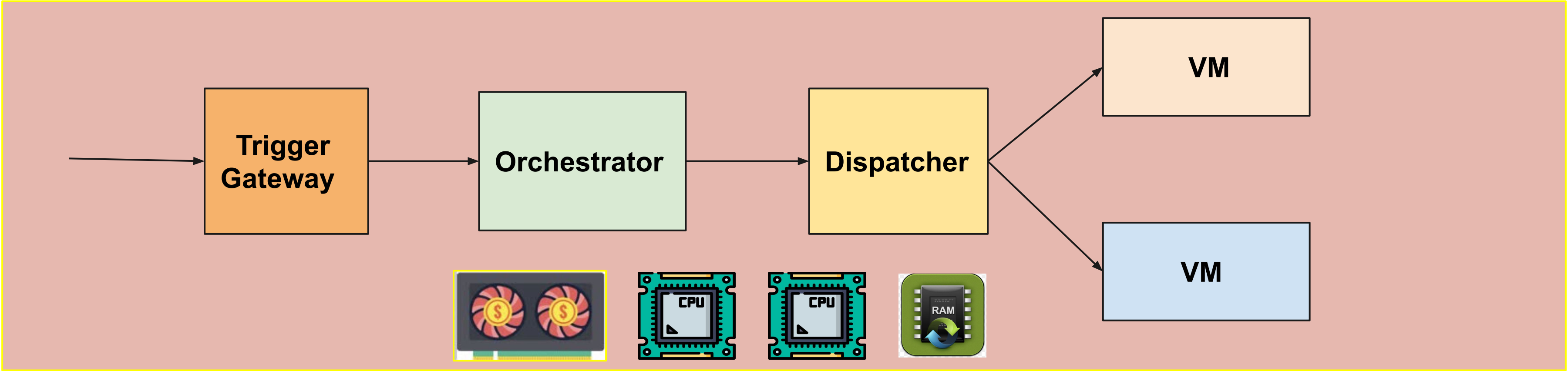
Serverless
Dynamic agile composition
Autoscaling
Pay-as-you-go



Context

Serverless
Dynamic and agile composition
Autoscaling
Pay-as-you-go





Seminar Scope

Resource Management

Data Handling

GPU Management & Provisioning

Resource Management

FaaS platform decouple the functionality and the setup and orchestration needed to execute functions

Platforms abstract away the operational complexities of managing servers, scaling, and resource allocation

Resource Management is an critical aspect of serverless platform as it provisions and manages resources without which functions wont run

Problems: Resource Management

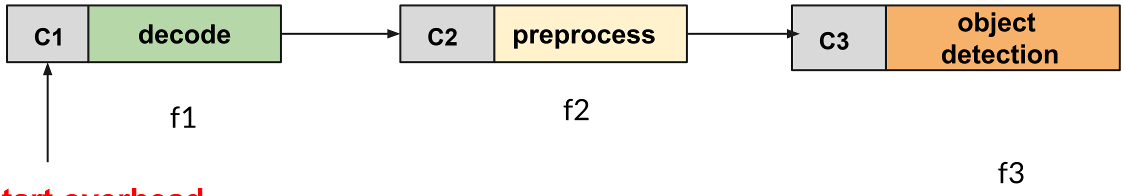
Cascading cold starts in serverless chain

Communication Latency between serial functions -> Remote Storage

Computation Skew among parallel invocations within the same stage

Large Resource Configurational Search Space

Cascading Cold Start in Serverless Workflows (CCS)



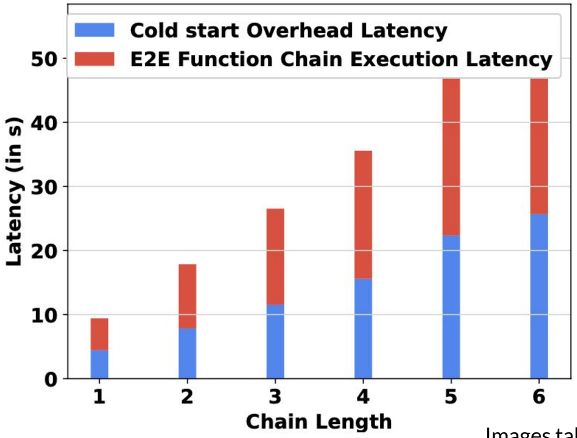
Coldstart overhead

Each function suffers from cold start

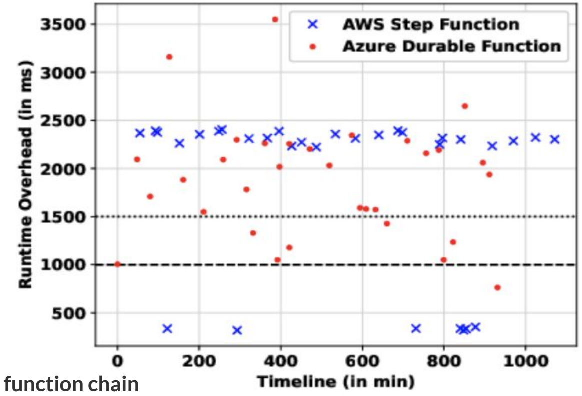
c1+c2+c3	f1+f2+f3

ccs

Function execution time



- **CCS overhead increases linearly with the length of the chain**
- **62.5% requests to ADF & 78.1% to AWS suffer from CCS**



Images taken from "Xanadu: Mitigating cascading cold starts in serverless function chain deployments"

CCS : Issues & Solution Approaches

Issue: CCS cause performance degradation

Solution: Speculatively pre-deploy resources

Issue: Naive pre-deployment is resource expensive

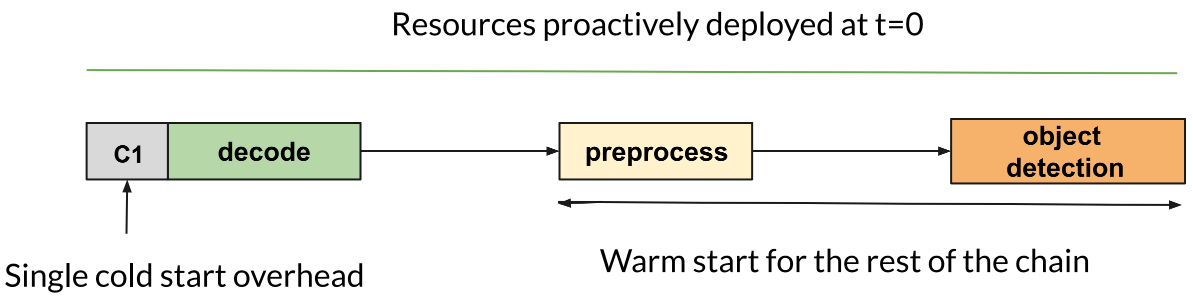
Solution: Detect the Most Likely Path (MLP) for speculative deployment

Issue: Cost of speculative deployment still significant

Solution: Deploy resources just in time (JIT) to reduce resource costs

Speculative deployment of resources

Speculatively pre-deploy all functions of the workflow on the onset of the chain execution



Advantages:

CCS reduces to a single cold start for the first function in the chain

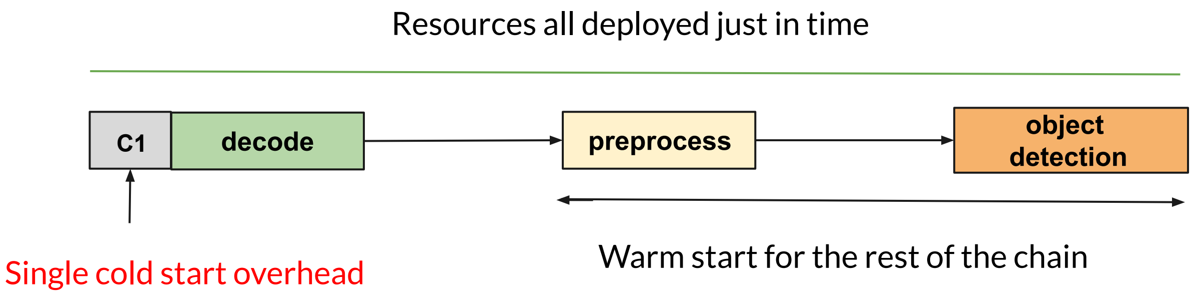
Disadvantages:

Resources locked ahead of time

Resource wastage in case of conditional chains

Just In Time Deployment along MLP

Delay the speculative deployment to reduce resource lock in time

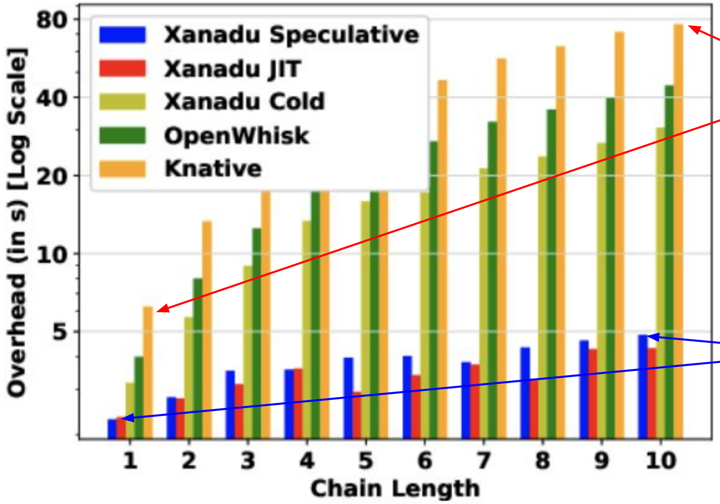


Profile the runtime characteristics of the function [cold start time, execution time]

Calculate the Most Likely Path (MLP)

Speculatively deploy resources along the MLP JIT before expected invocation

JIT Speculation Benefits



Knative & OpenWhisk : Latency increases linearly with chain length (~10x)

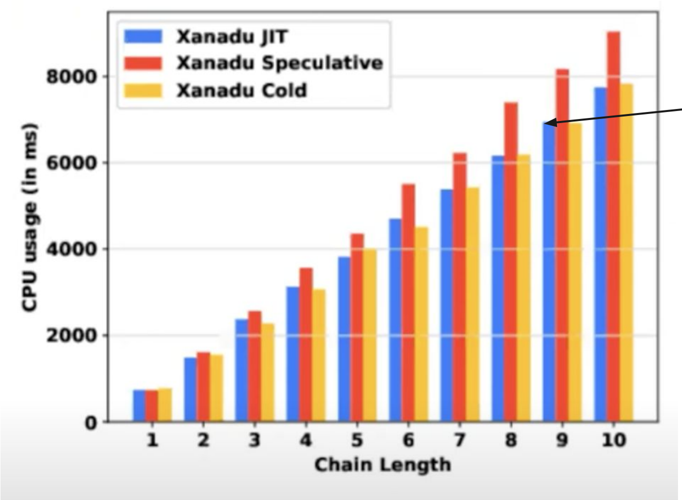
Xanadu : Latency increases only by ~ 1.11x

Baseline: OpenWhisk & Knative

JIT speculative deployment reduces startup latency to nearly 15-20 %

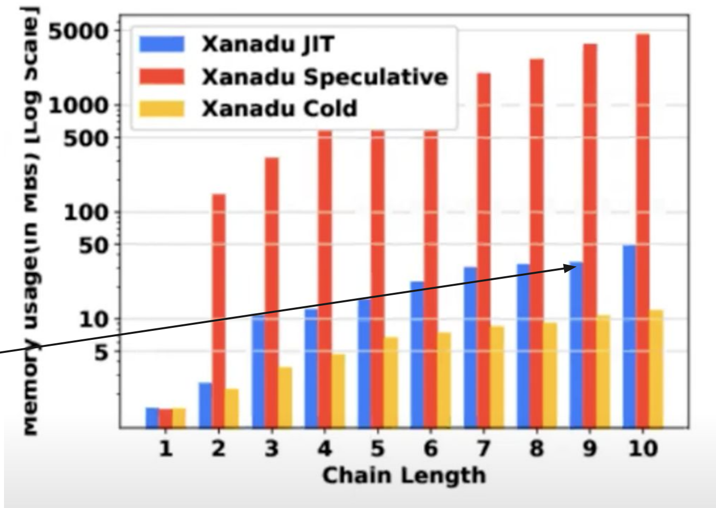
Images taken from "Xanadu: Mitigating cascading cold starts in serverless function chain deployments"

JIT Resource Advantage



Xanadu JIT just 0.9% more expensive than Xanadu cold

Xanadu JIT 2.18x more expensive than Xanadu cold



Xanadu Speculative more costly than Xanadu Cold due to **large resource lock-in period**

Xanadu JIT reduces both CPU cost and memory cost with negligible increase in resource provisioning costs [Images taken from “Xanadu: Mitigating cascading cold starts in serverless function chain deployments“]

Takeaways

A serverless platform that reduces cascading cold start to a single instance, reducing overhead by 10x compared to OpenWhisk

Dynamically detects MLP using probabilistic modelling
Speculatively deploy resources on the MLP

Performs JIT deployment to reduce resource overhead costs, limiting overhead to 0.9x for CPU and 2x for memory

Possible scope of extensions

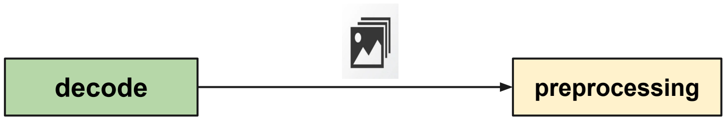
Prediction miss penalty mitigation is limited to stopping the speculation process itself

Can be extended to the MLP path to be re-evaluated and functions on the new path to be deployed speculatively based on the updated MLP.

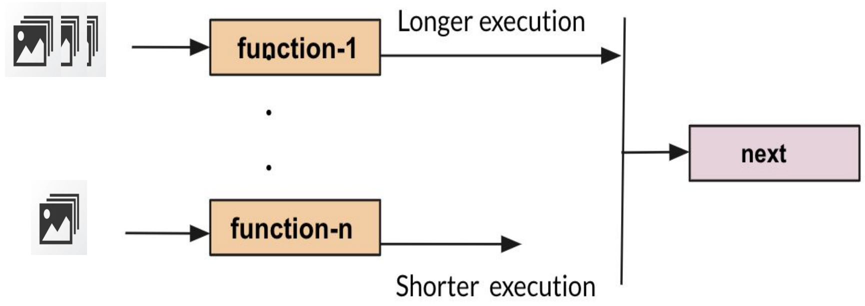
Can reduce the keepalive time of worker nodes for significant resource savings on account of speculative resource deployment

Serverless Workflows: Performance Bottlenecks

Communication Latency between serial functions



Computational Skew among in-parallel invocations within the same stage



Focus on FaaS Platforms that ***executes each function in a separate VM / microVM***

AWS Lambda - microVM

Azure Functions - VM

Serverless DAG Workload Characterization

General DAG structure : **wide** & **shallow:**
Max width : 10.9 K Max depth: 47
(Inferred from real workload traces from Azure Durable Functions)

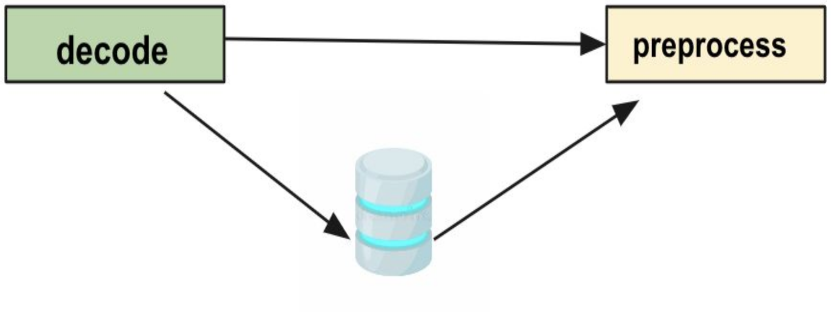
Top 5% most frequent DAG invocations:
- constitute **95%** of all DAG invocations
- invocation rate **1.6K/day**

Workload Characterization: Intermediate Data & Skew

DAGs with intermediate data size >= 1 MB have a ***9.5x higher median latency*** than DAGs with size < 1 MB

DAGs with skew >= 100 have ***17x times higher latency*** than DAGs with skew < 100

Communication Latency between serial functions



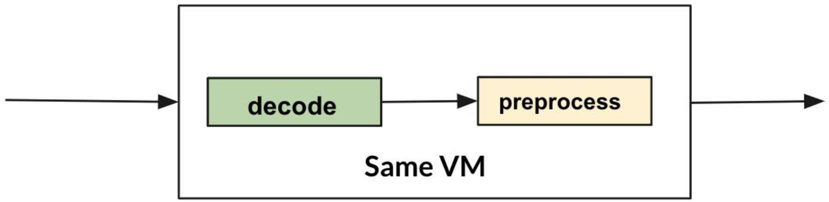
Remote storage
(S3)

Direct communication between serverless functions is infeasible

Asynchronous communication through remote storage

Adds to overall DAG latency

Fusion : Solution Approach



Fusion

Execute both functions in the same VM

Leverage local data passing

No data copy over the network

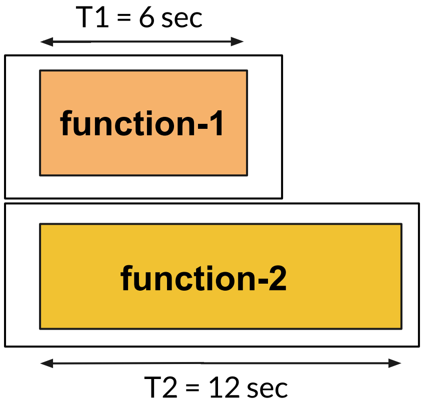
Reduce overall DAG latency

Challenges

Which functions to fuse?

Additional fusion cost if functions have different resource requirements?

Computation skew among parallel invocations

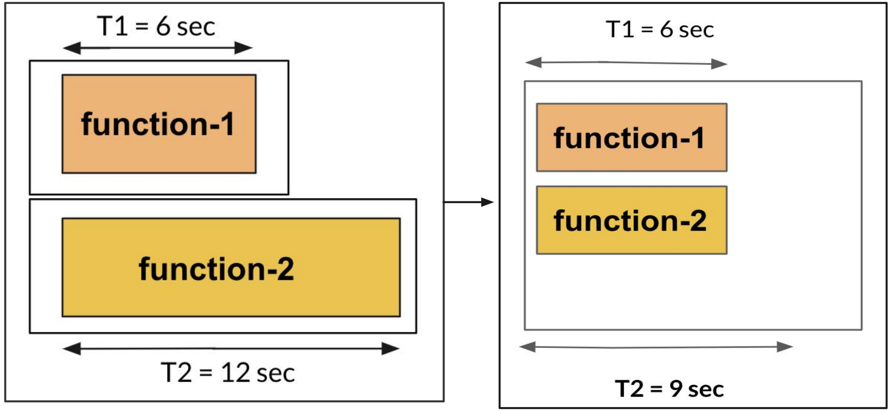


Separate VMs

Each invocation in different VMs

Straggler dominates the end-to-end latency

Bundling : Solution Approach



Execution on diff VMs

Single VM but double the size

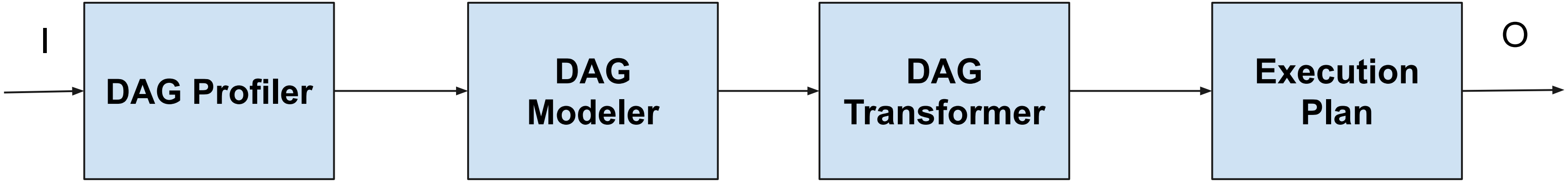
Execute both invocations in one VM

Straggler gets additional resources after fast invocation finishes execution

Challenge : Selecting a bundle size

DAG Transformation : Logical Flow

I : Input DAG
O : Optimized DAG



Captures execution time of each function and the entire DAG

Builds performance model for each function & DAG & estimated e2e latency

Identifies serial functions to fuse & Parallel stages that can benefit from bundling

Describes stages for fusion, bundle sizes and VM sizes

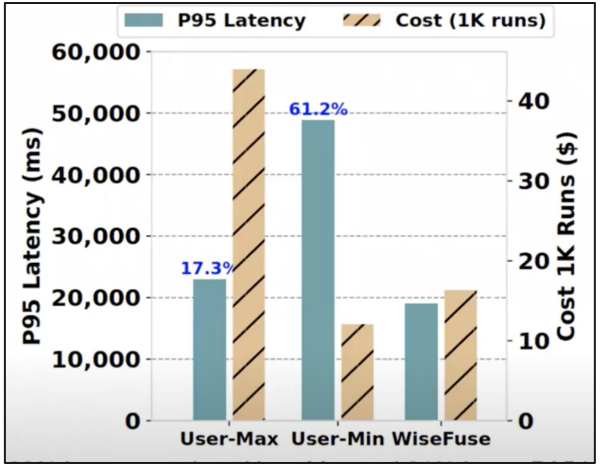
Experimental Evaluation : Video Analytics Application

Question: What is the tradeoff between latency and cost?

Test Platform : AWS Lambda

Baseline :
- User-Max : user-provided DAG using max VM sizes (**lowest latency**)
- User-Min : user-provided DAG using min VM sizes (**lowest cost**)

Related Works: Photons, Sonic , FaastLane



WiseFuse achieves **63% lower cost** than User-Max and **65% lower P95 latency** than User-Min

Comparison with related works

Observation	Reason
WiseFuse achieves 90% lower latency than Sonic	**Sonic :** Uses **ONLY** fusion Neither use bundling nor consider latency distributions
WiseFuse achieves 62% lower latency than Photons	**Photons : Does not** adjust bundle size to meet various latency targets **Does not** use fusion to reduce latency
WiseFuse achieves 39% lower latency than FaastLane	**FaastLane**: Does NOT implement fusion to reduce latency Uses fixed bundle size of 6 workers (to match AWS Lambda max VM size of 6 vcpus)

Takeaways

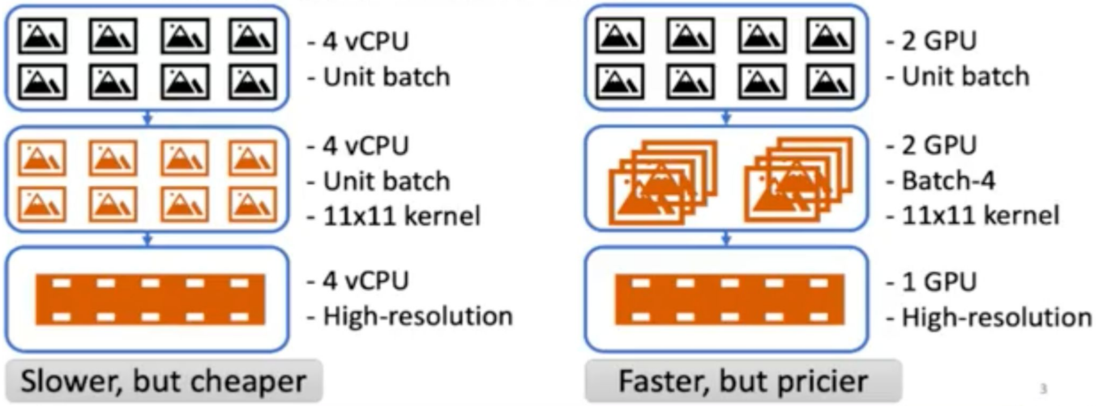
WiseFuse uses **Fusion** and **Bundling** operations to derive an optimized execution plan that meets a user-defined latency SLA with low cost

WiseFuse achieves a P95 latency that is 67% lower than Photons, 39% lower than Faastlane, and 90% lower than Sonic, without increasing the $ cost

Large Configuration Search Space

User configure operation knobs to best meet their targets

<Hardware resources, batch size, resolution…>

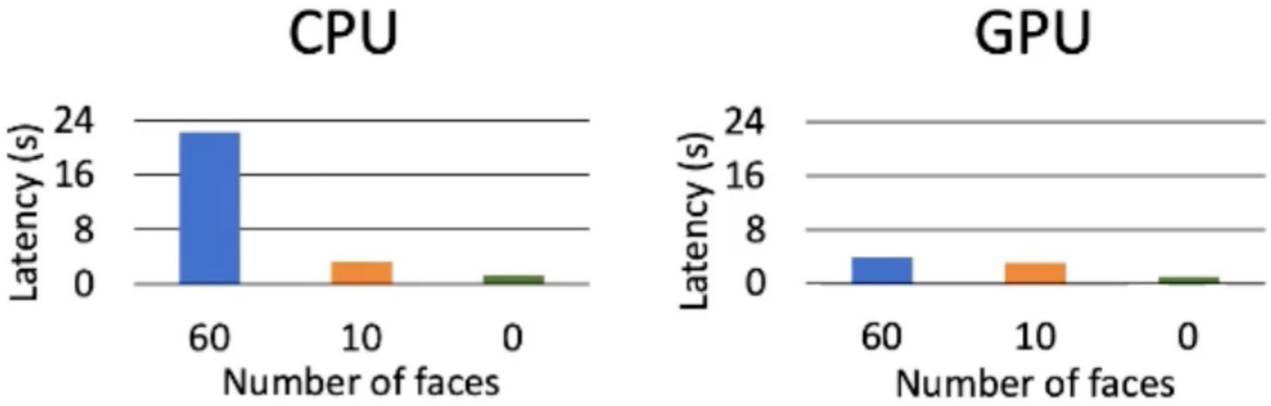


Maintain balance between latency and cost

Can have 100 different combinations of resources that can take hours and days to testing

Input-dependant execution flow

Example : Latency Dependency on Hardware



Challenges

Searching a large configuration space

If profiling does not exist, we need to generate a profile for each function

If there is a success in finding a configuration, then what if the input video itself changes?

What if the pipeline itself changes?
E.g. adding a new path to the workflow or adding a new function to the workflow

Problem: hours and hours of profiling !

Challenges

Existing frameworks struggle to handle dynamic workloads that change over time, leading to inefficiencies

Users may not have visibility into all components of a system and their interactions, making it difficult to determine the best configuration

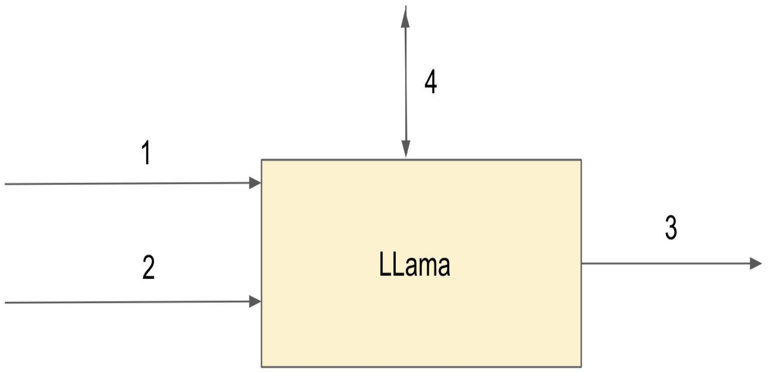
Exhaustive profiling per video is expensive and time consuming

Solution Approach

Perform one-time profiling of each function to collect performance and resource statistics (**Configuration decision based on profile**)

Configuration decisions are taken on a per-function invocation rather than a per-pipeline (**Reduces search space**)

Decompose the DAG into all possible sequential paths from start to end to compute per function slack (**Pipeline Decomposing**)



1, Registration (Pipeline DAG)
2. Invocation (DAG, input data, latency)
3. Invocation Result with configured system-level and user-level knobs/parameters
4. Knobs - CPU core, GPU memory, Batch Size etc.

Takeaways

Llama's ability to dynamically reconfigure operation invocations enables it to outperform existing systems, both in terms of latency and cost

Llama's slack allotment and configuration selection algorithms are effective in meeting pipeline latency targets while minimizing cost

Llama achieves an average improvement of 7.8x for latency and 16x for cost compared to state-of-the-art systems

Possible extensions

Limited Scope : experiments has been done on the domain of video analytics pipeline only.

No backup or recovery mechanism discussed for Llama components : what if any component fails?

Integration with edge devices : could be extended to support edge devices, such as cameras or gateways, which can improve the privacy and security of the video data.

Support for additional hardware resources : currently supports CPUs / GPUs but can be extended to use TPUs or specialized video processing chips

Challenges : Data Handling

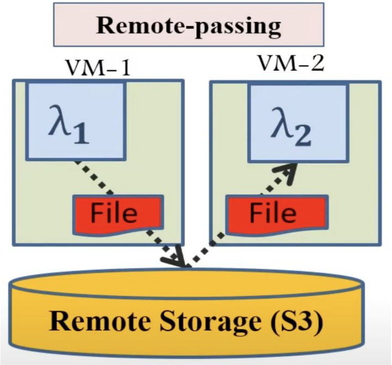
Serverless platform used by various workflow applications - video analytics, ML , BigData etc.

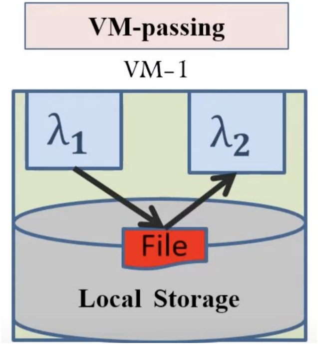
Serverless functions are ephemeral and stateless

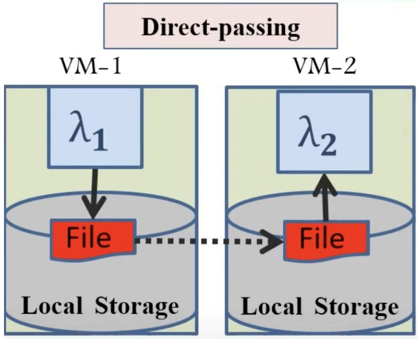
State-of-the art using remote storage as intermediate store (AWS S3) which adds to overall latency

AWS Step Functions supports passing direct JSON payloads of very small sizes (<= 256 KB)

Solutions - Data Passing Challenges







Data copied twice
Latency Overhead

Zero Data Copy
Reduced Latency

Single Data Copy
VM1's network bandwidth is the bottleneck

Images in this slide are cited from the paper "**SONIC: Application-aware Data Passing for Chained Serverless Applications**"

Data Passing Performance Tradeoff

LightGBM : This application trains decision trees, combining them to form a random forest predictor

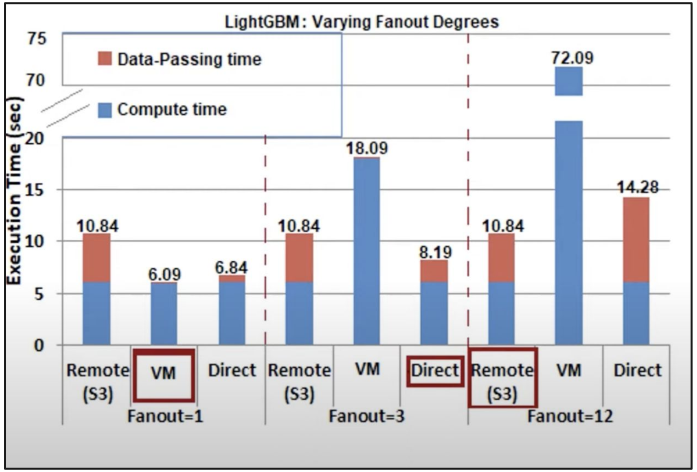
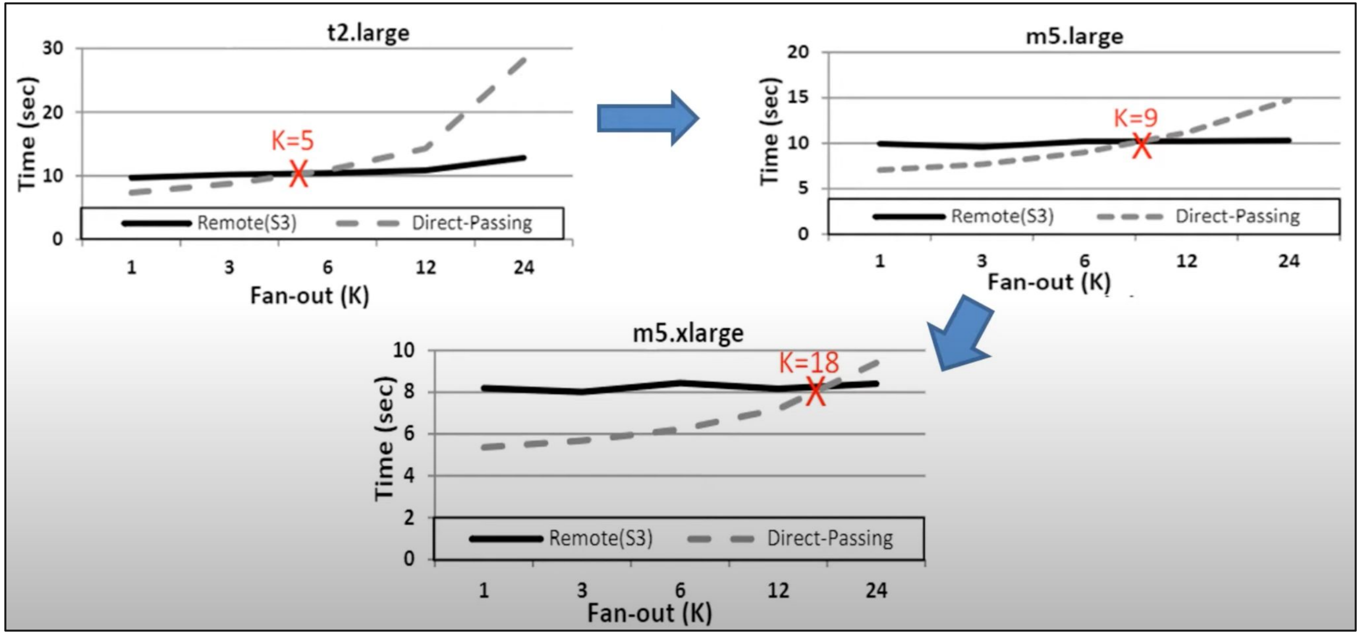


Image in this slide are cited from the paper “**SONIC: Application-aware Data Passing for Chained Serverless Applications**”

Direct Passing vs Remote Storage

With higher network bandwidth, the crossover point from direct passing to remote storage shifts to higher fanout values



Takeaways

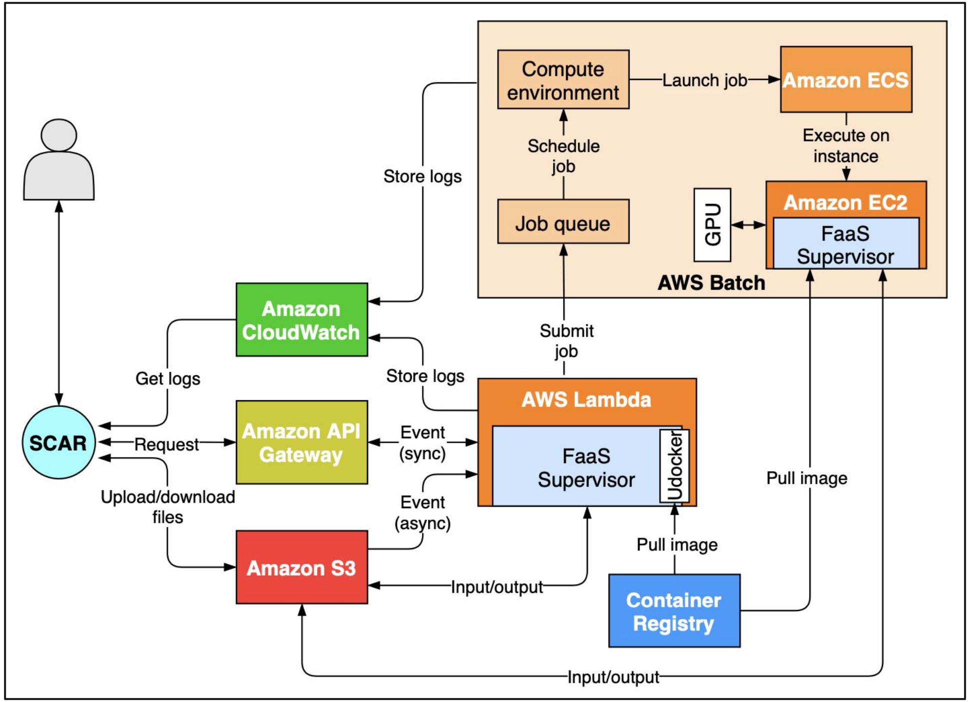
Studied **Sonic -** a dynamic and hybrid approach to select the best global data passing method and function placement

No single data passing method is the best and depends on DAG parameters like input size, fanout etc and system parameters like network bandwidth

Possible Extension : Handle content-dependance in application DAGs & handle dynamic control flows (Do not currently handle conditional)

Ways of supporting GPU for serverless workflows (1)

SCAR Framework on AWS Lambda & AWS Batch



Lambda has a constrained execution environment (**Max Execution Time = 15 min & Max RAM = 10,240 MB**)

Ephemeral disk storage is limited to 512 MB

No GPU support is available

FaaS Supervisor to AWS Lambda to delegate request to AWS Batch whenever timeout exceeds OR request comes for GPU

Specify request and executions modes by a **Function Definition Language** (FDL)

SCAR Setup & Workload

CPU workload on AWS Batch:
m5.xlarge instances (4 vcpus), Intel Xeon Platinum (Skylake-SP) processor (3.2 GHz), 16 GB RAM

GPU workload on Batch:
p2.xlarge, with 1 NVIDIA Tesla K80 GPU, 4 vCPUs, and 61 GB of RAM, and g3s.xlarge with 1 NVIDIA Tesla M60 GPU, 4vCPUs, and 30.5 GB memory

Workload: A Multimedia File Processing Workflow application which does frame-level **object detection** in video together with the **inclusion of subtitles** from the audio transcript (**ffmpeg** : lambda-batch, 1024 MB, **audio2srt** : lambda-batch, 1024 MB, **YOLOv3** :batch with 128 MB lambda)

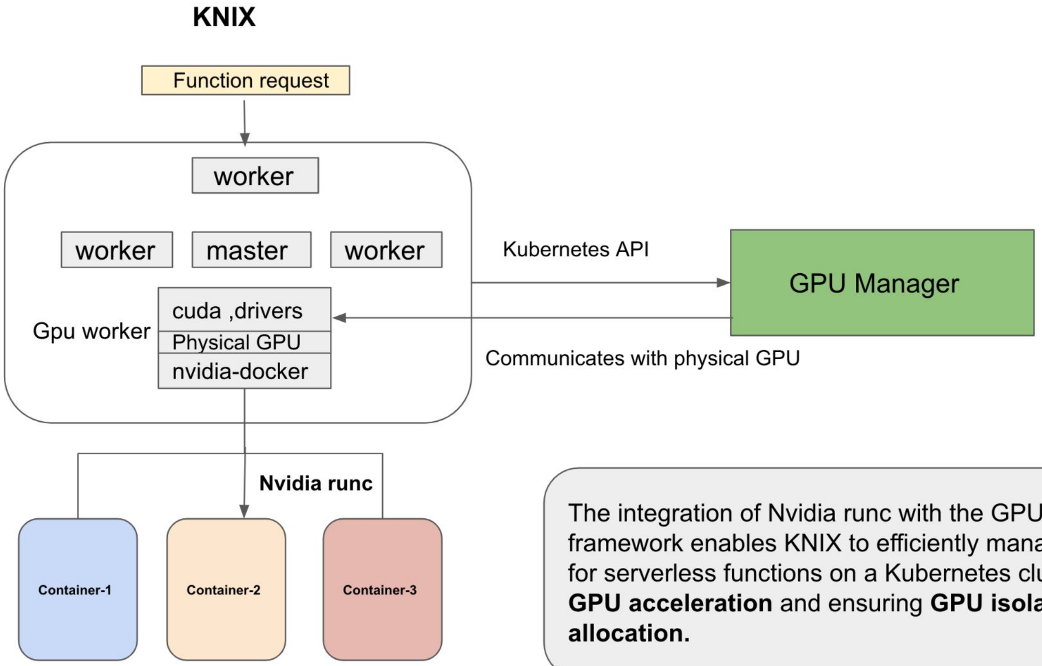
Negative Point

Time taken by the scheduler to launch and terminate instances is considerably longer than that of other Functions as a Service platforms.

Usage of traditional virtual machines instead of the container-based microVMs used by AWS Lambda

Ways of supporting GPU for serverless workflows (2)

KNIX : GPU Sharing Framework to provide fractional GPUs



The integration of Nvidia runc with the GPU Manager framework enables KNIX to efficiently manage GPU resources for serverless functions on a Kubernetes cluster, providing **GPU acceleration** and ensuring **GPU isolation** and **resource allocation.**

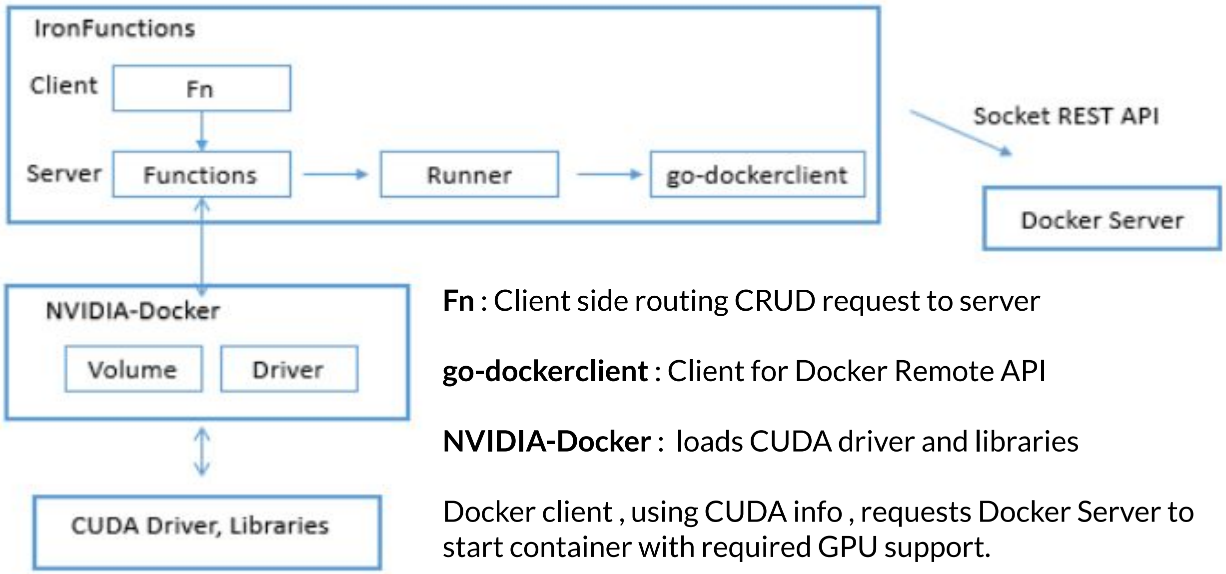
Leverage GPU Sharing Framework to partition physical GPUs into vGPUs

Fractional allocation of GPUs across functions in a workflow via vGPUs

(**Temporal Multiplexing**)

Ways of supporting GPU for serverless workflows (3)

API Based Integration of NVIDIA-Docker with Serverless Platform (IronFunctions)



Fn : Client side routing CRUD request to server

go-dockerclient : Client for Docker Remote API

NVIDIA-Docker : loads CUDA driver and libraries

Docker client , using CUDA info , requests Docker Server to start container with required GPU support.

Existing Challenge

The process of communicating for GPU use in both local and remote environments is complex and involves a lot of overhead

Purchasing a GPU instance from a cloud computing service provider & install drivers and libraries in those VM

Calculating Most Likely Path

```
parents ← workflow.root;
while parent in parents do
    siblings ← parent.children;
    foreach sibling ∈ siblings do
        sibling.prob ← parent.branch [sibling ] *
          parent.prob;
        parents.Append (sibling);
    end
    mlp.Append (Max (siblings));
end
```

Conclusion

Studied various aspects of resource management and their solutions like JIT along MLP for mitigating CCS , DAG workload characterization , Fusion and Bundling to optimize DAG execution and reduce configuration search space by per-function profiling rather per-pipeline profiling

Studied various data passing methods and no single data passing method is the best. Depends on parameters like fanout, input size, network bandwidth etc

Studied various ways of provisioning GPU for serverless workflows

References

- Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. "**Xanadu:Mitigating Cascading Cold Starts in Serverless Function Chain Deployments**". In: Proceedings of the 21st International Middleware Conference. Middleware '20. Delft, Netherlands: Association for Computing Machinery, 2020, pp. 356–370. isbn: 9781450381536. doi: 10.1145/3423211.3425690. url: https : / / doi . org / 10 . 1145 / 3423211.3425690

- Jaewook Kim et al. "**GPU Enabled Serverless Computing Framework**".In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). 2018, pp. 533–540. doi:10.1109/PDP2018.2018.00090

- Ashraf Mahgoub et al. "**WISEFUSE: Workload Characterization and DAG Transformation for Serverless Workflows**". In: Proc. ACM Meas Anal.Comput. Syst. 6.2 (June 2022). doi: 10 . 1145 / 3530892. url:https://doi.org/10.1145/3530892

- Sebasti´an Risco and Germ´an Molt´o. "**GPU-Enabled Serverless Workflows for Efficient Multimedia Processing**". In: Applied Sciences 11.4 (2021). issn: 2076-3417. doi: 10 . 3390 / app11041438. url: https : //www.mdpi.com/2076-3417/11/4/1438.

- Francisco Romero et al. "**Llama: A Heterogeneous amp; Serverless Framework for Auto-Tuning Video Analytics Pipelines**". In: Proceedings of the ACM Symposium on Cloud Computing. SoCC '21. Seattle, WA, USA: Association for Computing Machinery, 2021, pp. 1–17. isbn: 9781450386388. doi: 10.1145/3472883.3486972. url: https: //doi.org/10.1145/3472883.3486972.

- Klaus Satzke et al. "**Efficient GPU Sharing for Serverless Workflows**". In: Proceedings of the 1st Workshop on High Performance Serverless Computing. HiPS '21. Virtual Event, Sweden: Association for Computing Machinery, 2021, pp. 17–24. isbn: 9781450383882. doi: 10 . 1145/3452413.3464785. url: https://doi.org/10.1145/3452413. 3464785.

THANK YOU !!