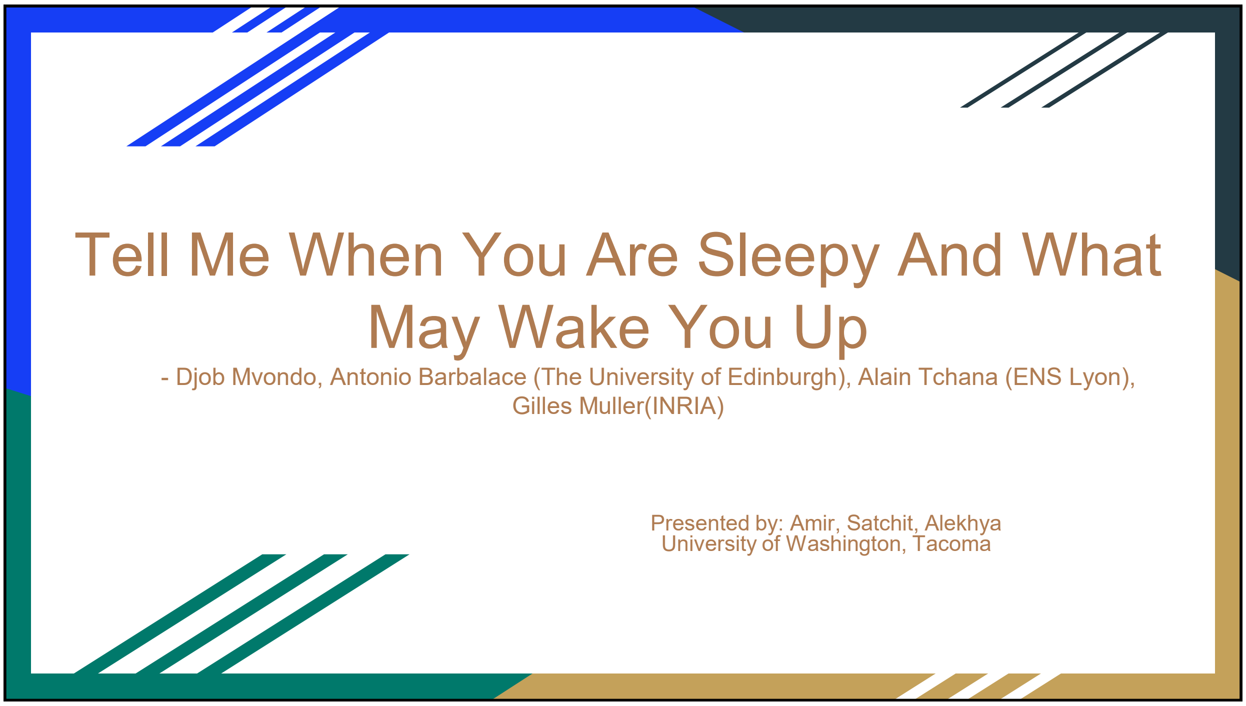
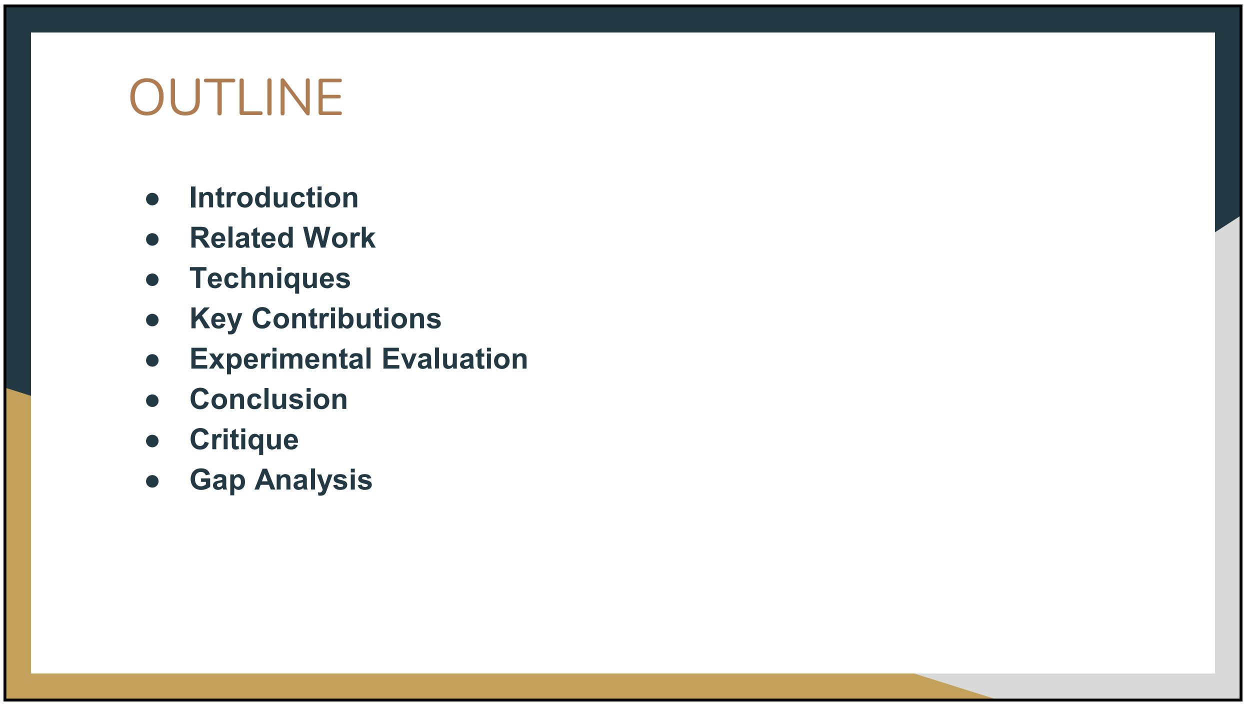


Tell Me When You Are Sleepy And What May Wake You Up

- Djob Mvondo, Antonio Barbalace (The University of Edinburgh), Alain Tchana (ENS Lyon), Gilles Muller(INRIA)

Presented by: Amir, Satchit, Alekhya
University of Washington, Tacoma

1



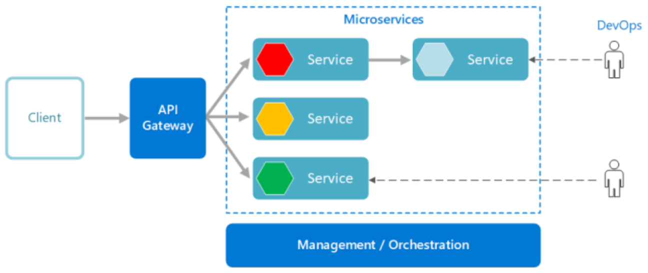
OUTLINE

- Introduction
- Related Work
- Techniques
- Key Contributions
- Experimental Evaluation
- Conclusion
- Critique
- Gap Analysis

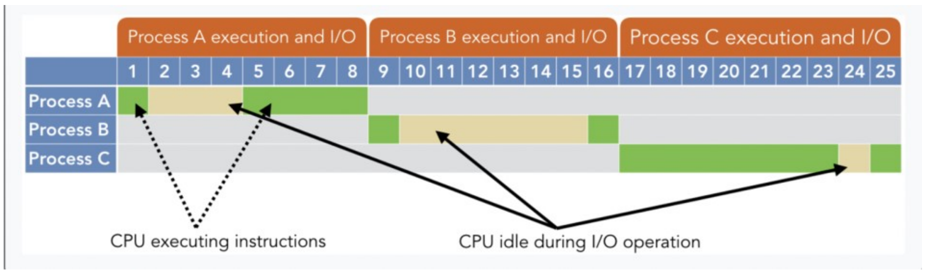
2

Introduction

- Cloud computing is the delivery of different services through the Internet.
- Edge computing is the practice of capturing, processing, and analyzing data near where it is created.
- Generally applications based on cloud or edge computing are divided into micro services which are then implemented through different services, like Virtual Machines, Containers, Functions e.t.c.



3



- Sometimes situations arise where some of the services or multiple instances of a service are running on the same server. When this happens, not all the services are in process, some may be in an idle state. These idle units would be consuming CPU time,

- In the above example, the yellow part of processes execution indicates the time CPU is idle and it can be observed that the amount of time CPU spends waiting is almost similar to the time it spends in processing.

4

Introduction continued

The Problem: Units in an idle state are consuming excessive CPU time which can be used for other cloud utility functions or other execution units.

This is a problem because idle units can spend up to 70% of CPU time waiting where the CPU is not utilized but the cost of usage keeps increasing.

The wastage of CPU resource can grow significantly when we have situations of overcommitting resources.

If we figure out a way to moderate the CPU idle time and figure out a way to schedule operations without wastage, it would mean we would have more processing power and processing time in our hands for the same cost. The optimization of resources when working with cloud computing is a priority to achieve efficiency.

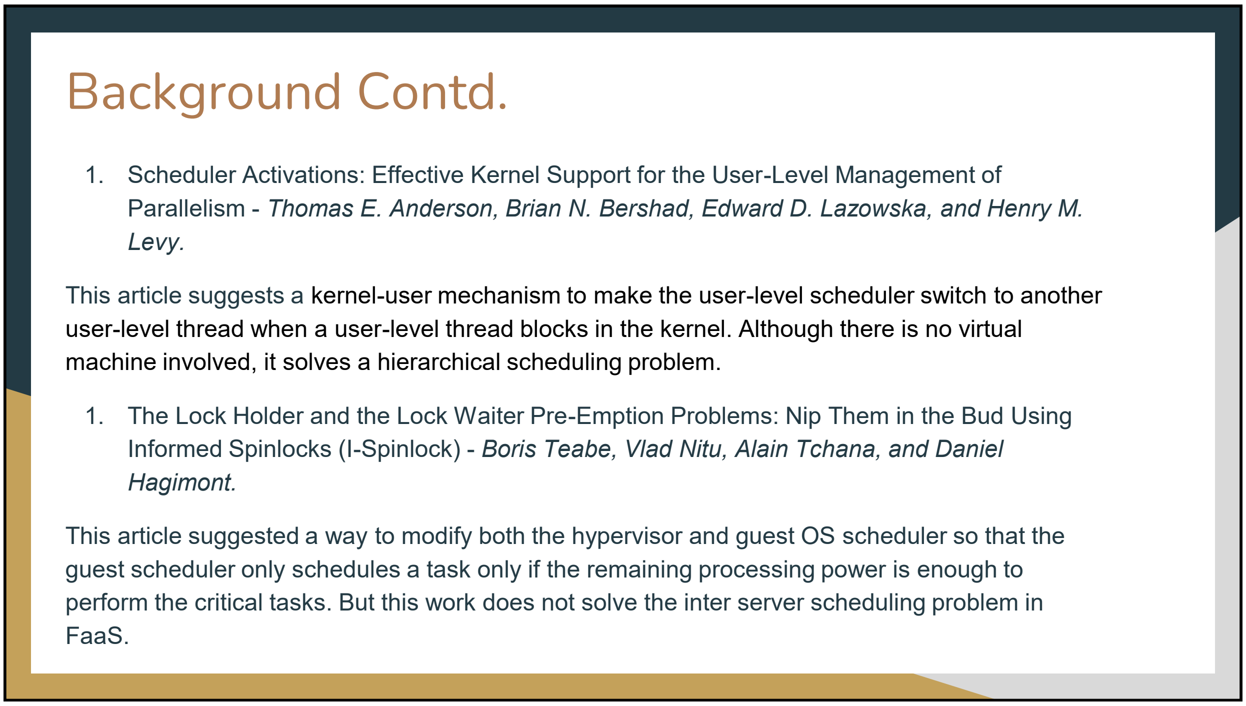
5

Background

The author of the paper states these articles as their reference for pointing out that schedulers do not work properly in specific workload situations.

1. vSMT-IO: Improving I/O Performance and Efficiency on SMT Processors in Virtualized Clouds - *Weiwei Jia, Jianchen Shan, Tsz On Li, Xiaowei Shang, Heming Cui and Xiaoning Ding*
2. Preemptable Ticket Spinlocks: Improving Consolidated Performance in the Cloud - *Jiannan Ouyang and John R. Lange. 2013*
3. Opportunistic Spinlocks: Achieving Virtual Machine Scalability in the Clouds - *Sanidhya Kashyap, Changwoo Min, and Taesoo Kim. 2016.*

6



Background Contd.

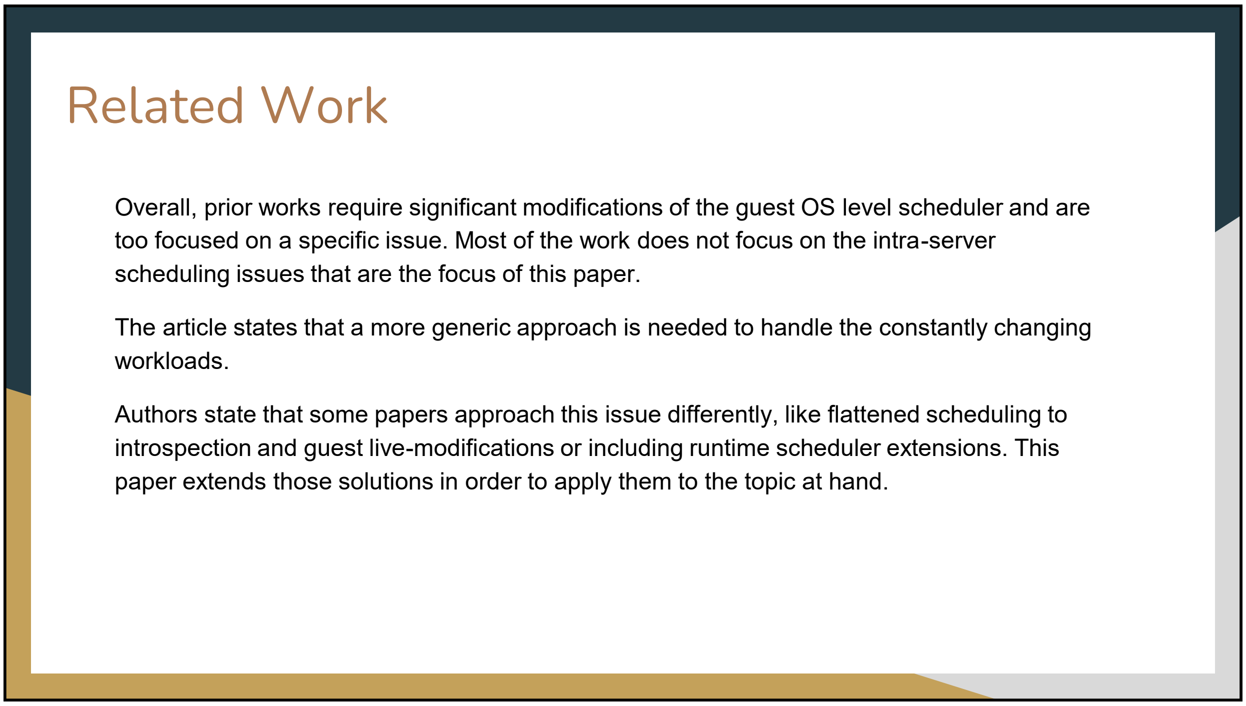
1. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism - *Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy.*

This article suggests a kernel-user mechanism to make the user-level scheduler switch to another user-level thread when a user-level thread blocks in the kernel. Although there is no virtual machine involved, it solves a hierarchical scheduling problem.

1. The Lock Holder and the Lock Waiter Pre-Emption Problems: Nip Them in the Bud Using Informed Spinlocks (I-Spinlock) - *Boris Teabe, Vlad Nitu, Alain Tchana, and Daniel Hagimont.*

This article suggested a way to modify both the hypervisor and guest OS scheduler so that the guest scheduler only schedules a task only if the remaining processing power is enough to perform the critical tasks. But this work does not solve the inter server scheduling problem in FaaS.

7



Related Work

Overall, prior works require significant modifications of the guest OS level scheduler and are too focused on a specific issue. Most of the work does not focus on the intra-server scheduling issues that are the focus of this paper.

The article states that a more generic approach is needed to handle the constantly changing workloads.

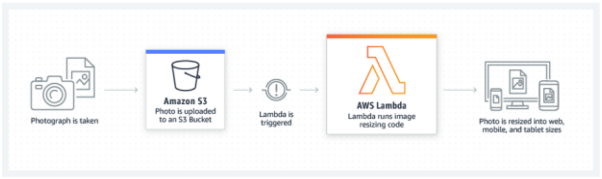
Authors state that some papers approach this issue differently, like flattened scheduling to introspection and guest live-modifications or including runtime scheduler extensions. This paper extends those solutions in order to apply them to the topic at hand.

8

Summary

Current host OS schedulers do not perform as expected when faced with specific workloads. The combination of scheduler hierarchy with the black box nature of Virtual Machines is the cause of these problems.

In this paper, the authors demonstrate that these limitations are also the cause of reduced number of microservers that can be placed on the same server. Due to the insufficient knowledge about the way microservices communicate, avoidable latencies are formed between triggers and executions.


Photograph is taken
Amazon S3
Photo is uploaded to an S3 Bucket
Lambda is triggered
AWS Lambda
Lambda runs image resizing code
Photo is resized into web, mobile, and tablet sizes

An example of a trigger - when a photo is uploaded, it automatically runs a lambda function for resizing the image.

Avoidable latency is when even after a photo is sent to S3, the function is not triggered due to scheduling issues.

9

Summary and Proposals of Approaches

This paper proposes a redesign of the scheduling ways in such a way that it includes:

(a) accompanying VMs/tasks with knowledge about their working model

(b) monitoring VMs/tasks for events that change their internal and external state.

Proposed Approaches:

- **Approach 1**: The cloud orchestrator provides scheduling information in the form of a manifest file to the OS scheduler
- **Approach 2**: The VM provides contextual information about the microprocesses to the scheduler of the server.
- **Approach 3**: Instead of having the contextual information about the microprocesses, it will reconstruct itself with a form of VM introspection (similar to interactive scheduling algorithms)

10

Pros and Cons of Proposed Approaches

- **Approach 1**:
  - Pros: Secure
  - Cons: Requires information be provided in domain specific language with sets of rules or as a precompiled program written to support it
- **Approach 2**:
  - Pros: Happens at runtime, so it is fast
  - Cons: Not secure
- **Approach 3**:
  - Pros: The scheduler would have VM introspection making it efficient
  - Cons: Not secure

11

Prototype design

- Goal: Oversubscribing CPU resources
- Doesn't affect service latencies
- Extensible hyperservor/ host OS scheduler
- Can be customizable at runtime
- Information about customization shipped together with VM and built by the cloud itself

(a combination of Approach 1 and Approach 2)

12

Key Contributions

Observations:

- A microservice may not be doing any processing not only while it is waiting
- Microservices communicate mostly with network packets and it is easy to identify what triggers/events change an inactive or idle microservice into an active one.

Takeaways:

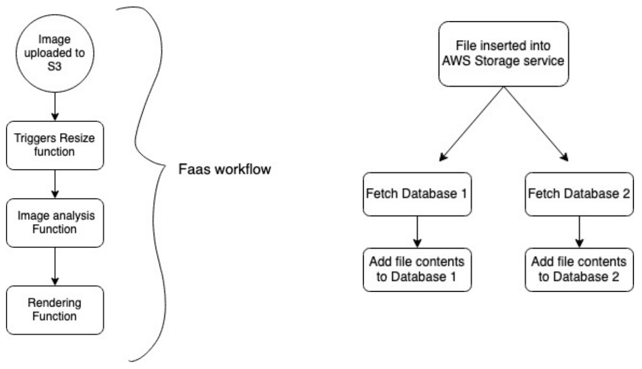
- To use many number of microservices on a single server, we need to find a new design for scheduling.
- The information about services which is constantly changing over time should be provided to the host OS scheduler.
- Security of services should be taken into account.

13

Experiment

An experiment was conducted to trigger interdependent workloads) on a single server.

FaaS platforms are used for pipeline of functions and hence this was used for the experiment.



14

Experiment continued

An experiment was conducted to trigger inter dependent services on a single server. FaaS platform was chosen to host the workload.

- Two chain microservices are created and hosted on single server
- First - 5 microservices.

This service performed image processing and consisted of 5 image processing functions (blurring, edging, resize, gamma, and sepia filtering) to generate a thumbnail

- Second - 3 microservices

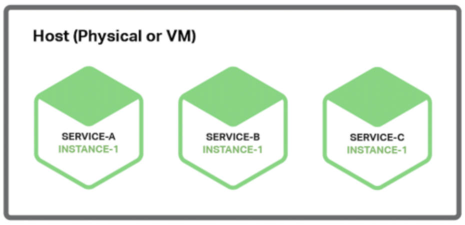
This service performed online compiling with gg and performed a 3-stage compilation of hello world in C and llvm build.

15

Techniques

- Chains of microservices on a single server were used

Chain of 5 microservices and chain of 3 microservices are used, as 82% of all use cases consist of applications that use five or less different functions



An example of microservices on a single server

16

Techniques

- To ease deployment, Amazon Firecracker was used.
- The results were reassessed by using FaaS containers for the same experiment.

For a better evaluation of the experiment, it was done in two different network latency situations.

1. Local Network - An amazon EC2 instance on the local computer was used to run the functions in an a1.metal server.
2. Remote Network - Functions were run in a lab server with the data stored in remote AWS S3 service. This resulted in higher network latency.

17

Experiment - Eval Metrics

Function-level metrics:

1. trigTime - the latency between the trigger and the start of the chain execution
2. avgInterTime - the inter function latency
3. execTime - the total chain execution time

The input and output are stored in Amazon S3 and read from that database.

18

Experiment continued

- The experiment is repeated while changing the number of colocated chains from 0 to 50.
- For every run, the total idle time of CPU is collected.

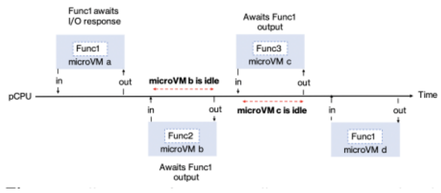


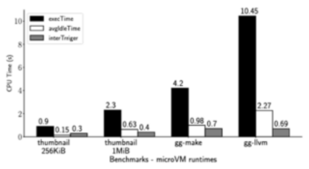
Diagram from the article showing micro VM idle times. VMs b and c are scheduled and then wait for output from VM a, which results in wasted CPU time.

19

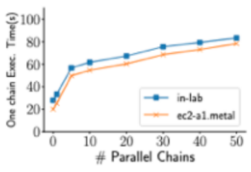
Experiment - Results

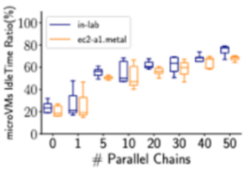
- The local network model sees an increase in execution time by 2.9x and the remote network model increases by 3.92x.
- Average VM idle time ratio ranges from 20% to 75% with the remote network model and from 18% to 70% for the local network model. An increment of 3.73x and 3.79x is observed for the remote and local network models respectively.

[5]With respect to each micro-VM total runtime



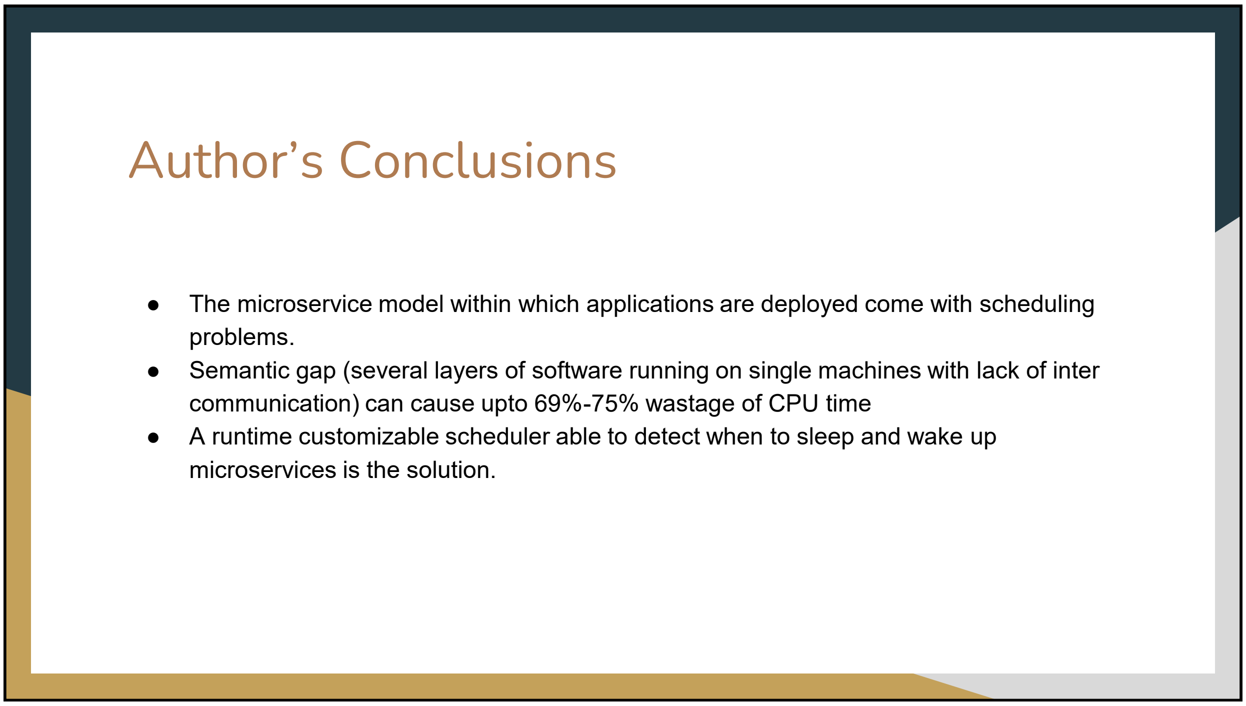
When running multiple microservices, the total execution time increases hugely with the number of invocations, ranging up to 4 times the original time.





Running on local networks, the CPU idle time goes upto 27% of total runtime.

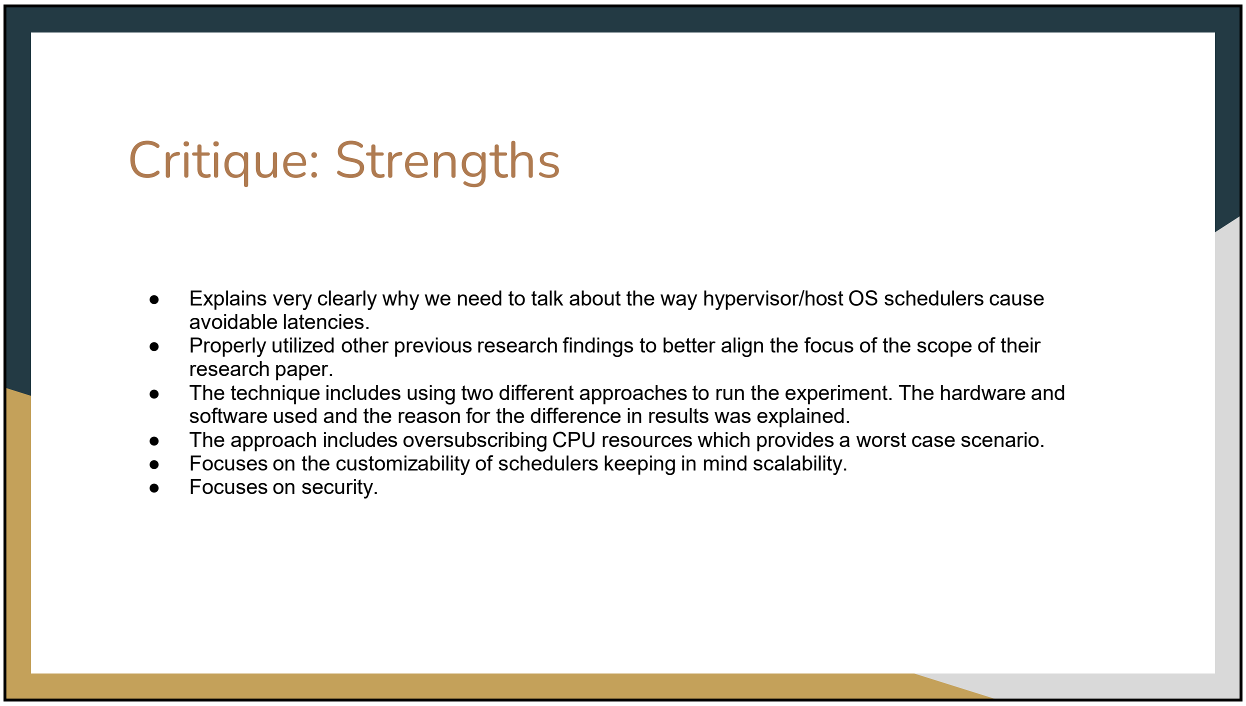
20



Author’s Conclusions

- The microservice model within which applications are deployed come with scheduling problems.
- Semantic gap (several layers of software running on single machines with lack of inter communication) can cause upto 69%-75% wastage of CPU time
- A runtime customizable scheduler able to detect when to sleep and wake up microservices is the solution.

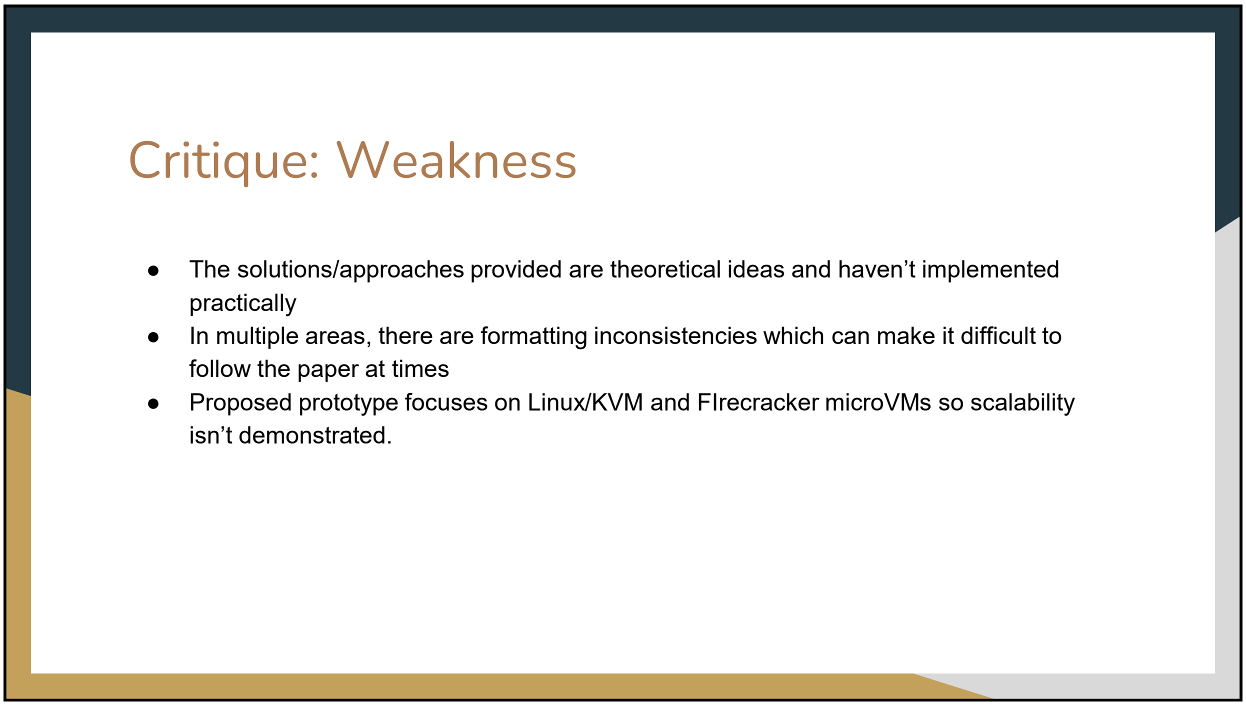
21



Critique: Strengths

- Explains very clearly why we need to talk about the way hypervisor/host OS schedulers cause avoidable latencies.
- Properly utilized other previous research findings to better align the focus of the scope of their research paper.
- The technique includes using two different approaches to run the experiment. The hardware and software used and the reason for the difference in results was explained.
- The approach includes oversubscribing CPU resources which provides a worst case scenario.
- Focuses on the customizability of schedulers keeping in mind scalability.
- Focuses on security.

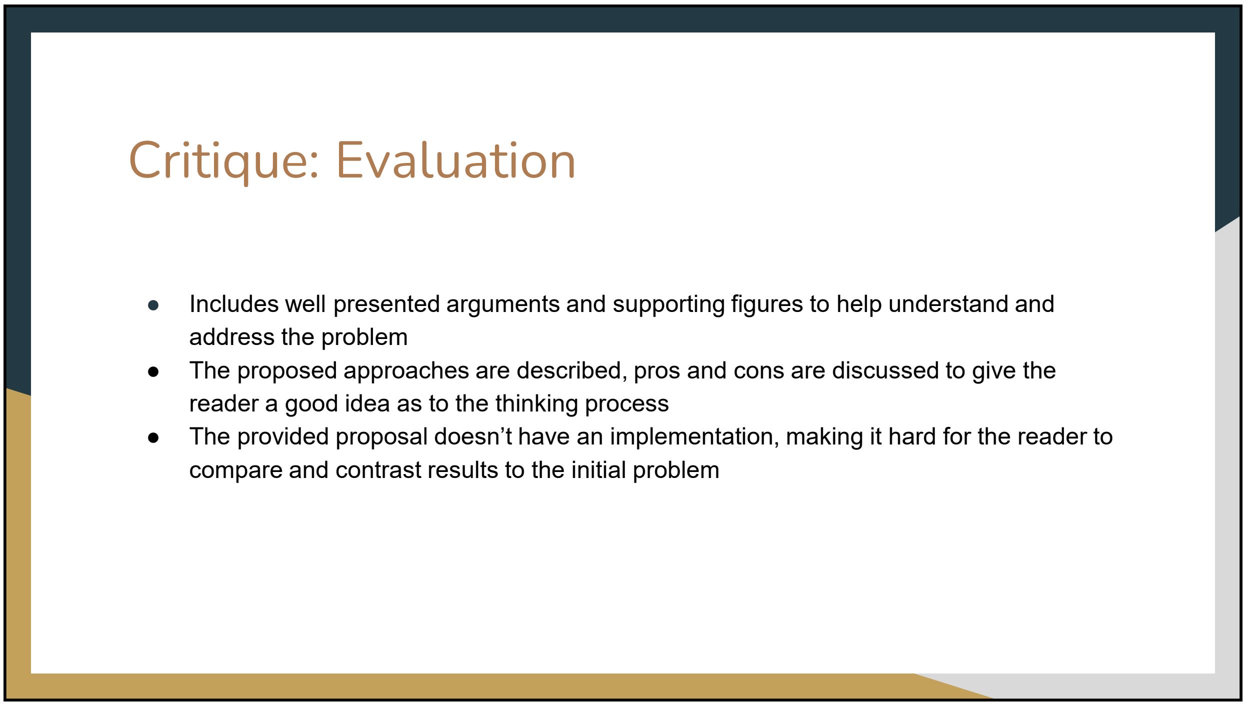
22



Critique: Weakness

- The solutions/approaches provided are theoretical ideas and haven't implemented practically
- In multiple areas, there are formatting inconsistencies which can make it difficult to follow the paper at times
- Proposed prototype focuses on Linux/KVM and FIrecracker microVMs so scalability isn't demonstrated.

23



Critique: Evaluation

- Includes well presented arguments and supporting figures to help understand and address the problem
- The proposed approaches are described, pros and cons are discussed to give the reader a good idea as to the thinking process
- The provided proposal doesn't have an implementation, making it hard for the reader to compare and contrast results to the initial problem

24

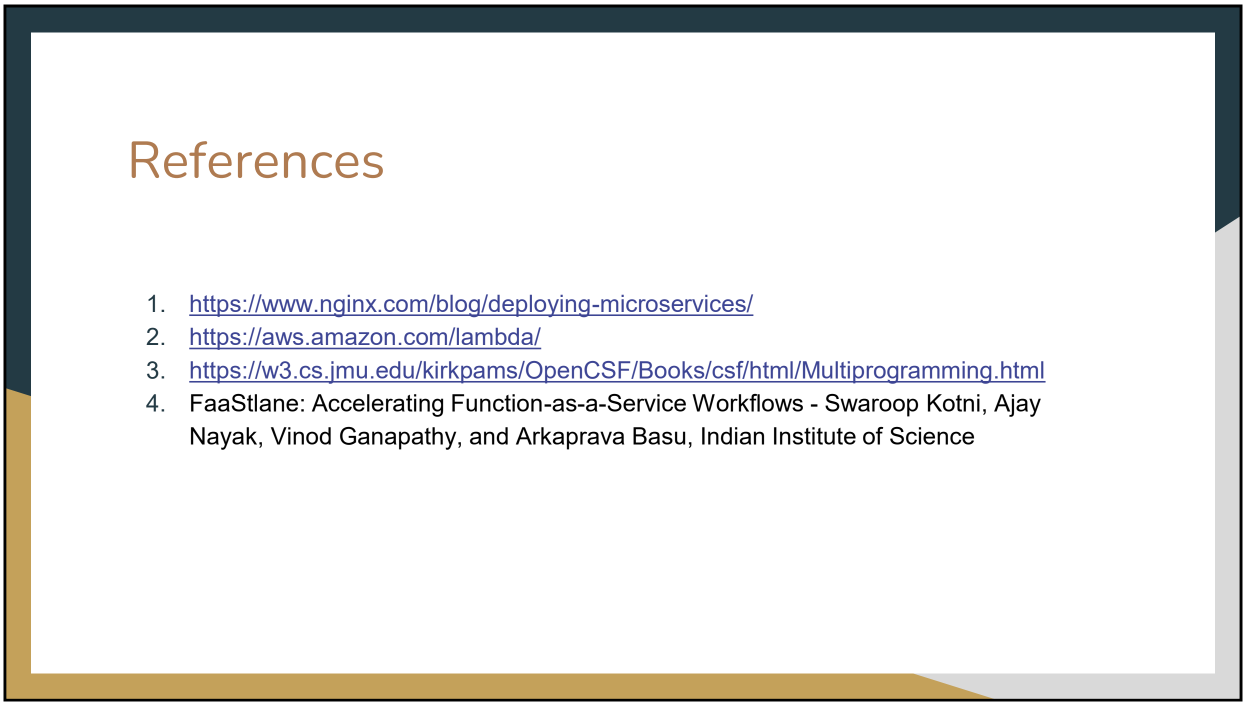
Identify: GAPS

- Implementation of proposed solutions not provided
- No data to read since it is an ongoing project for the authors

25

Questions

26



References

1. https://www.nginx.com/blog/deploying-microservices/
2. https://aws.amazon.com/lambda/
3. https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/Multiprogramming.html
4. FaaStlane: Accelerating Function-as-a-Service Workflows - Swaroop Kotni, Ajay Nayak, Vinod Ganapathy, and Arkaprava Basu, Indian Institute of Science

27