Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads

Sadjad Fouladi, Riad S. Wahby, and Brennan Shacklett, *Stanford University;* Karthikeyan Vasuki Balasubramaniam, *University of California, San Diego;* William Zeng, *Stanford University;* Rahul Bhalerao, *University of California, San Diego;* Anirudh Sivaraman, *Massachusetts Institute of Technology;* George Porter, *University of California, San Diego;* Keith Winstein, *Stanford University*

(thanks for the images)

Background: AWS lambda

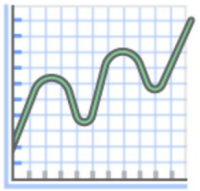
AWS has recently offered the AWS-lambda service

Vast stateless computational resources usable for short amounts of time cheaply

Has the potential to democratize cloud computing

No Servers to Manage

AWS Lambda automatically runs your code without requiring you to provision or manage servers. Just write the code and upload it to Lambda.



Continuous Scaling

AWS Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.

Subsecond Metering

With AWS Lambda, you are charged for every 100ms your code executes and the number of times your code is triggered. You don't pay anything when your code isn't running.

The idea

The authors proposes to use AWS lambda to provide massive parallelism cheaply, useful in for example in video encoding and interactive video editing

Google Docs *for Video*?

"Apply this awesome filter to my video."

Related work - lightweight virtualization

There are many batch-processing frameworks (Hadoop, Mapreduce) for coarse-grained parallelism, the authors considers more fine-grained parallelism

Lightweight cloud computing has previously been used for web-microservices, but not for compute heavy jobs.

“After the submission of this paper, we sent a preprint to a colleague who then developed PyWren, a framework that executes thousands of Python threads on AWS Lambda”

Technical contribution - mu

The authors implements a library for massive parallel computations on AWS lambda

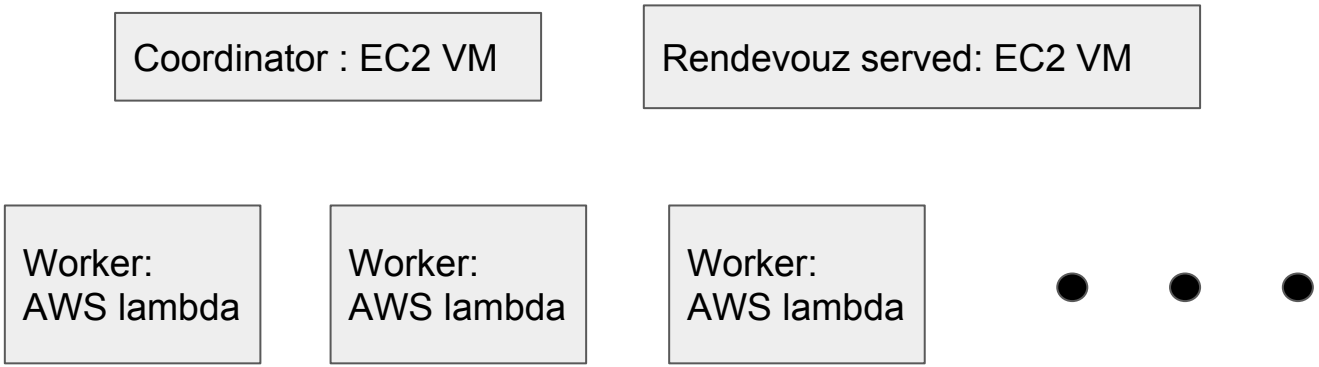
Challenges include:

- Lambda functions must be installed before launched, which can take a long time
- The timing of worker invocation is unpredictable
- Workers can run for at most 5 min

mu: implementation details

A central, long lived, coordinator launches short lived jobs through the lambda API using HTTP

Short lived workers receive instructions from the coordinator, and communicates through a rendevouz server

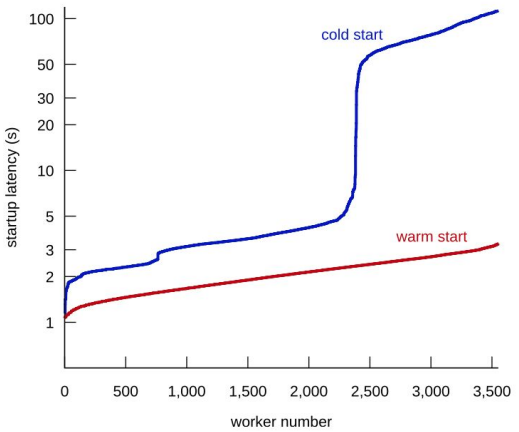


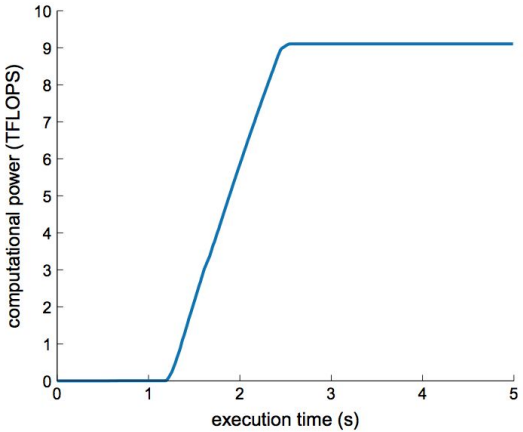
mu: micro benchmarks

The authors perform some basic experiments on linear algebra benchmarks

We see (upper picture) that it takes longer time due to rate limiting logic to set up many workers on a “cold start” while “warm starts” are much faster

Within seconds we however have access to vast computational resources





Background: video encoding

Around 70% of consumer web traffic is accounted for by videos

Video compression is used but requires vast computational resources for high resolution videos which makes providing low latency video encoding challenging

The massive parallelism of mu can be used here

Related work - parallel video encoding

Parallelism for video encoding has been explored previously

Separate patches of the video stream can be encoded in parallel, and different ranges of frames can be encoded in parallel

Some systems let workers find natural subsections, such as scenes in a movie, to work on, the authors consider a more fine grained parallelism

Technical contribution: parallel video encoding

In video encoding the dependency between frames makes it possible to “figure out” what should be in one frame given the earlier frame, which enables compression

Typically a compressed video stores a “keyframe” which is a complete but expensive specification of a frame, and then stores following “interframes” cheaply by figuring out what should follow the “keyframe”

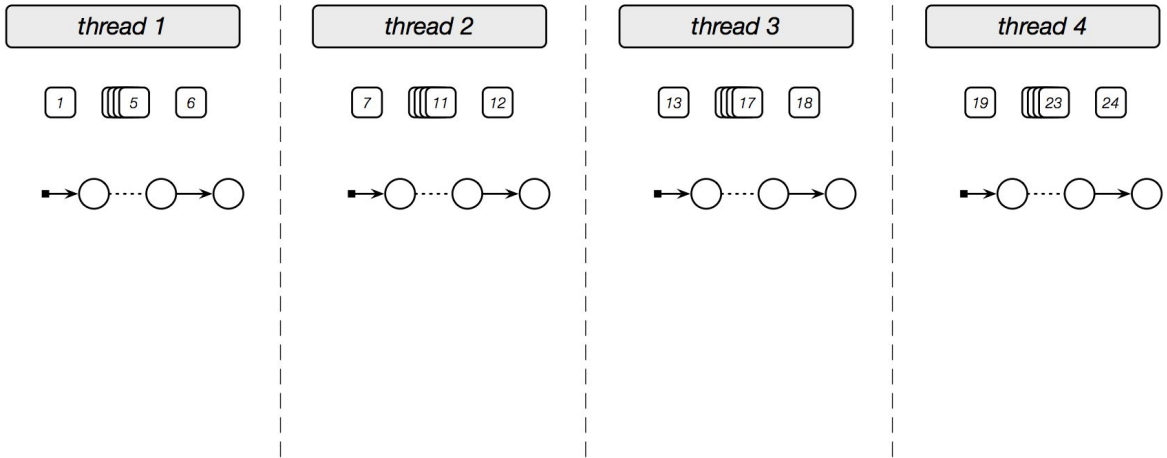
By insertion more keyframes we get parallelism at the cost of compression

The authors proposes a method of using virtual keyframes to enable massive fine-grained parallelism in video encoding

In a 4K video @15Mbps, a key frame is **~1 MB**, but an interframe is **~25 KB**.

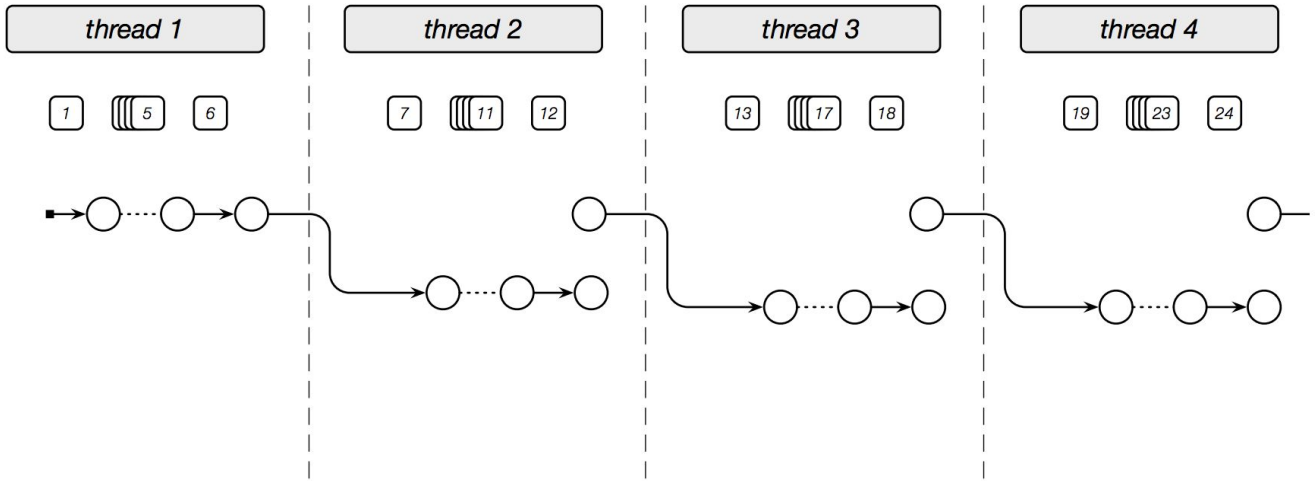
Details: parallel video encoding

1. The video is split into smaller parts, and each part is given to a single lambda worker
2. In parallel the workers encode their respective part, using the first frame as an expensive keyframe



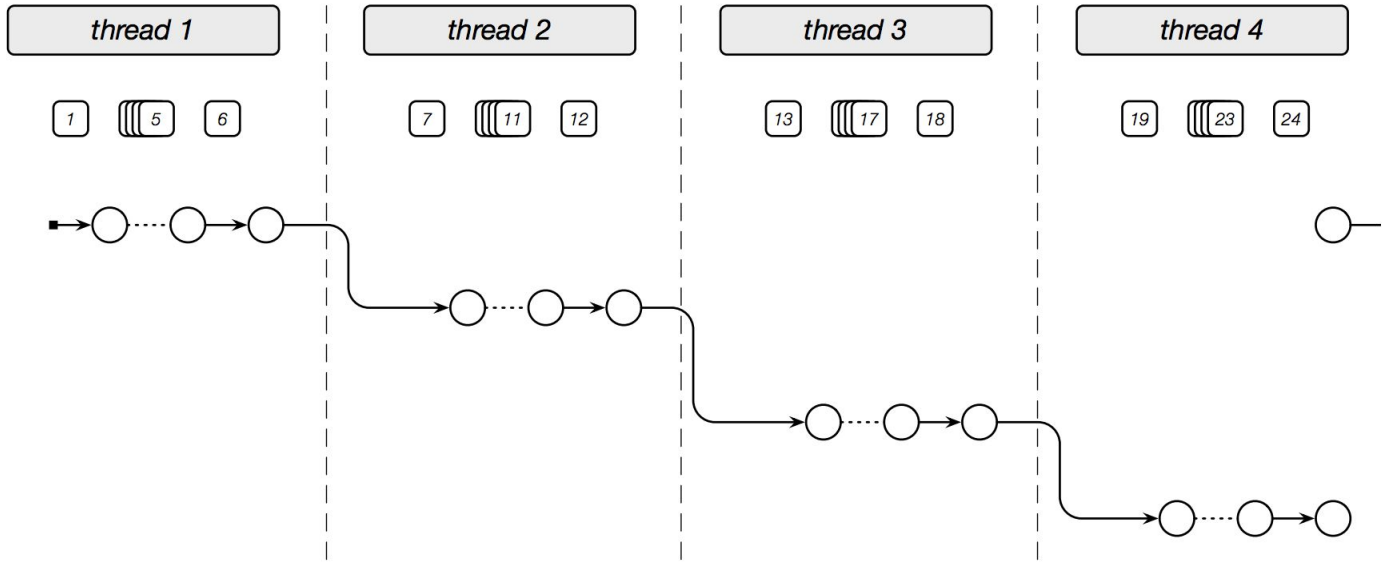
Details: parallel video encoding

3. In parallel, the workers uses the compressed frame before its keyframe to change its expensive keyframe to a normal, compressed frame



Details: parallel video encoding

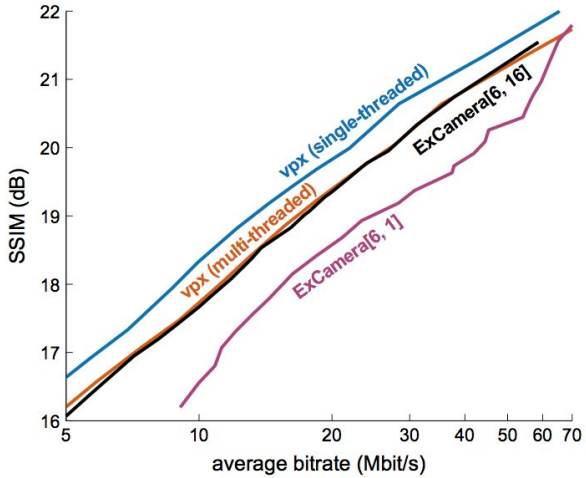
4. Serially, we “rebase” the frames which is cheap as we use the already provided prediction models



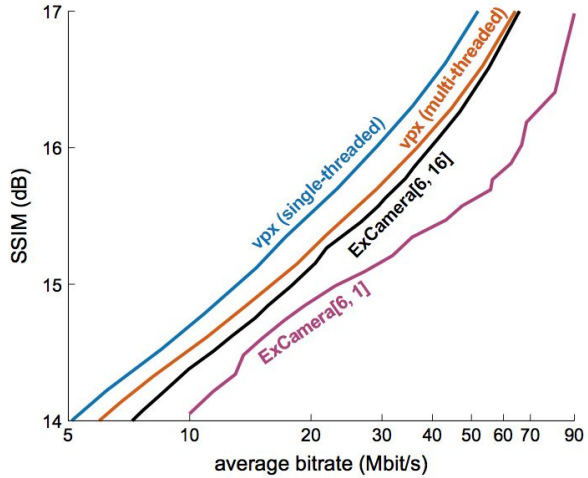
Results

The system almost matches the performance of popular alternatives,

with much higher degree of parallelism



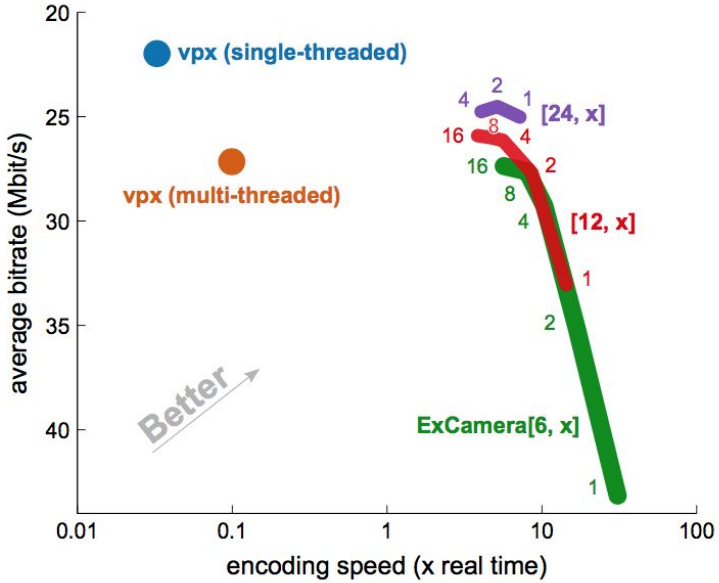
(a) Sintel (animated)



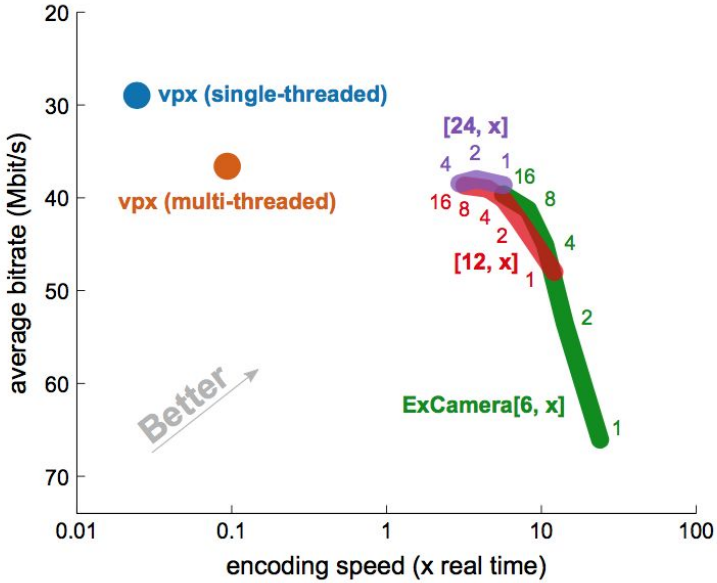
(b) Tears of Steel (live-action)

Results

Encoding is however much faster



(a) Sintel (SSIM of 20 dB)



(b) Tears of Steel (SSIM of 16 dB)

Shortcomings

The system is susceptible to worker failure

As rebasing is done sequentially, workers spend a lot of time waiting

The authors say that the compression rate for their keyframe->interframe frame method is bad

Shortcomings, higher level

It is mostly useful for very high resolution videos

Many jobs doesn’t require fine grained parallelism

When is latency an issue?

Future directions

The idea of using AWS lambda as for turn-key supercomputing is interesting

Are there other potential applications where latency is important?

Is it possible to do video encoding with deep learning?