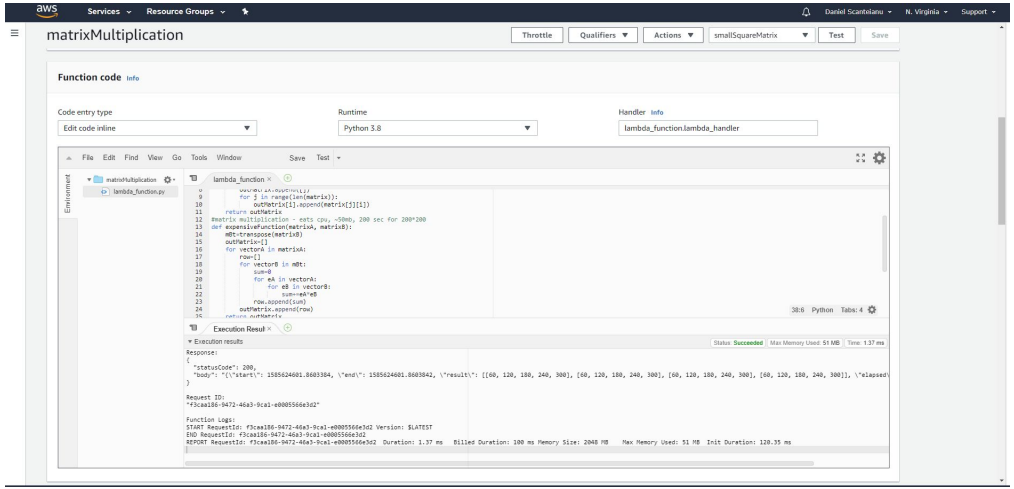
Black Box Benchmarking FAAS Platforms

Daniel Scanteianu OS/2 Project

What is FAAS/Serverless?

- Function As A Service
- Serverless
  - Cloud provider handles server
  - You write some code
  - Cloud provider deploys
  - Function gets triggered
  - Cloud provider handles scaling up and scaling down
  - You get billed per invocation/execution time
- Uses containers under the hood



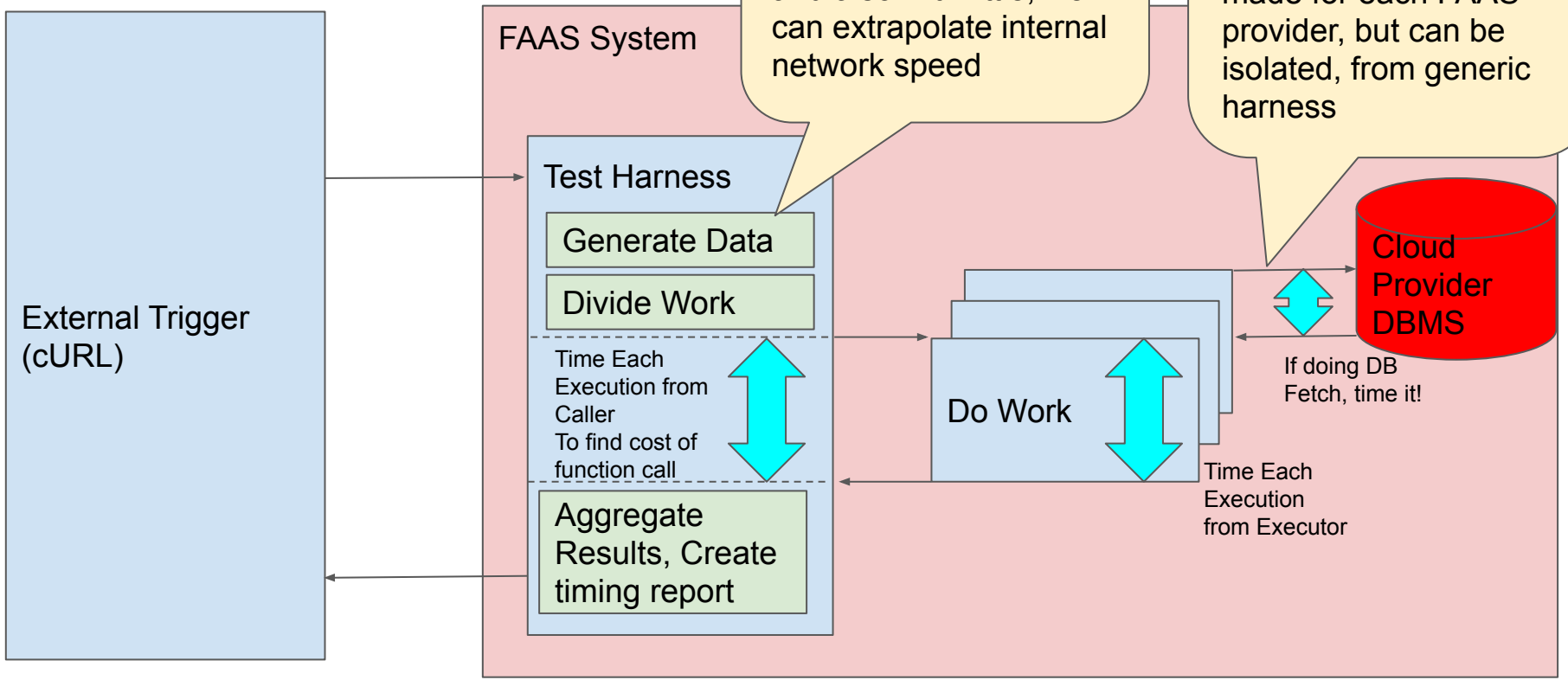
Business Motivation

- Some websites and companies are moving their entire systems to FAAS
  - Someone else can handle the SRE part of owning a company
    - Provisioning servers
    - Maintaining servers, having data centers
    - Managing OS, other things on said servers
  - Scale on demand
- Several providers of FAAS
  - AWS, MS Azure, Google Cloud, IBM, others
  - No Standard
    - Apache OpenWhisk - adopted by IBM - is leading open source contender
    - Migrating over to another provider could be costly
- Goal: choose the serverless provider that is most performant for your type of workflow
  - If you are the serverless provider, tune your system for a certain kind of workflow

High Level Strategy

- Goal: Get a nuanced view of how performant cloud FAAS providers are along different axes
  - Sub-goal: define these axes
- Create workloads that exercise different capabilities of underlying FAAS infrastructure
  - Infrastructure time
    - Startup time
    - Network bandwidth
  - CPU
- Provide a report of performance along each of these axes across many invocations, and also potentially at scale (ie: concurrent requests)
  - Remove as many confounding variables as possible

Benchmark Design



Benefits of Benchmark Design

- Harness sits inside data center
  - My home network latency is not part of the equation
  - Simulates invocation coming from another part of the workflow
  - If host computer and target computer clocks are reasonably well synced, can separate invocation latency from response latency
- If coded modularly, can be used to profile multiple lambda providers
- Gives me a more complete idea of what working with different FAAS providers is like

Possible Characteristics to benchmark

- CPU intensive workflows
  - Matrix multiplication
- Network intensive workflows - network throughput vs latency to be considered
  - Take large chunk of data, pass it to function, do some work on it, pass it back
- Monolithic workflows vs Parallel Workflows
  - Divide one of the cpu intensive tasks between a few workers,
  - see how fast it runs as a monolithic, single threaded workflow
  - See how fast it runs as a monolithic, multi threaded workflow, if possible
  - See how performance changes if each of these threads is given a function to be called in parallel from the parent function
- Read from database, do math, return small result? Or just read from db?
- Effects of language choice on any/all of above
- Effects of cold start on any of above

Previous Technical Findings/Background

- Lambdas sit in "preheated" language runtimes inside containers
  - Startup cost first time a function is run
    - Or if it's moved to another box, or not run in some time
- Lambdas are provisioned by RAM allocation
  - How much cpu, how fast it is, etc. is up to the provider
  - Providers may differ quite a bit in terms of cpu speed, number of cores
- Some expected latency with each invocation, due to having to start up the lambda

Possible novel network/latency experiment results

- Consider an automatic work-divider
  - If a large task is divided into n very small chunks, how many separate function calls (m) should be spun up, each taking n/m of these chunks and executing in order to maximize speed
- Recommend whether to send in data in request, or have functions read data from a database?
- Are some cloud service providers better at some kinds of workflows than others (ie: should you choose azure for a compute heavy workflow, but aws for a workflow that shuffles a lot of data around, but does less math)?

Real world applications

- Financial analytics - this is very close to home
  - Analytics platform, or workflows like screening, factor backtests, and aggregation of large data sets, can be implemented as FAAS
  - Data fetch and transform applications can be written in faas
  - Storage and infrastructure design decisions can be optimized for FAAS bottlenecks
- Private cloud owners can leverage this testing framework to evaluate the quality of their setup (ie: if they spin up an open whisk cluster)
- Potential faas users can choose a provider based on a much more granular understanding of the performance of the faas system

So Far...

- Read papers, trying to find a viable project
- Did background research, trying to see what others have done in this space
- Wrote some sample benchmarks
  - Matrix multiplication
    - CPU Intensive
  - Average of all elements of a matrix
    - Size of matrix much larger in proportion to size of computation
- Did some sample tests!

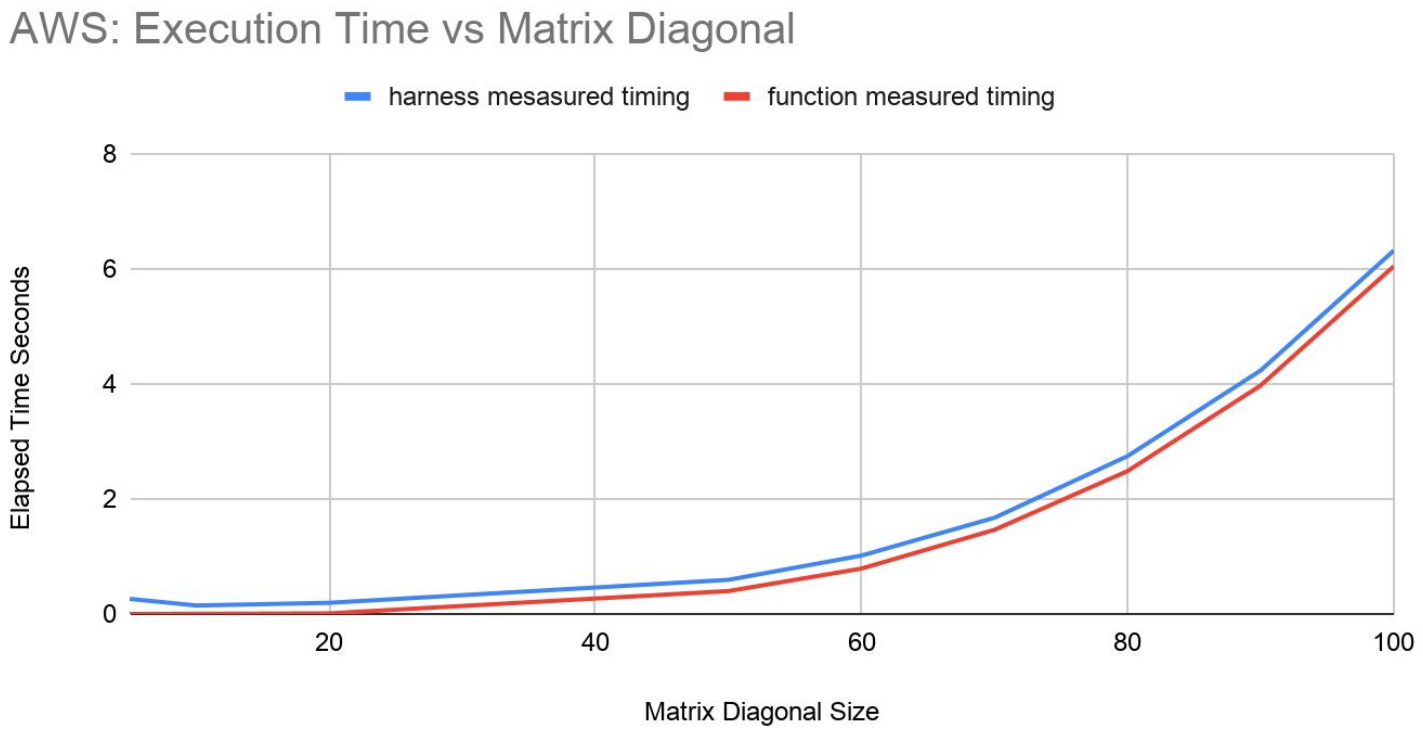
So Far...

- Read papers, did background research
- Designed the harness for running tests from multiple frameworks with minimal change to the tests themselves
- Wrote my first benchmark: Matrix Multiplication
  - Given a square matrix with side length n, O(n^4)
  - Computation Heavy
  - Not super memory intensive
  - Someone else also used this as a benchmark
  - Very similar to some financial analytics usecases
  - Targets:
    - Measure latency of network
    - Measure startup/delivery latency differences
    - Measure performance differences
- Did some sample tests

Test: AWS Matrix Multiplication

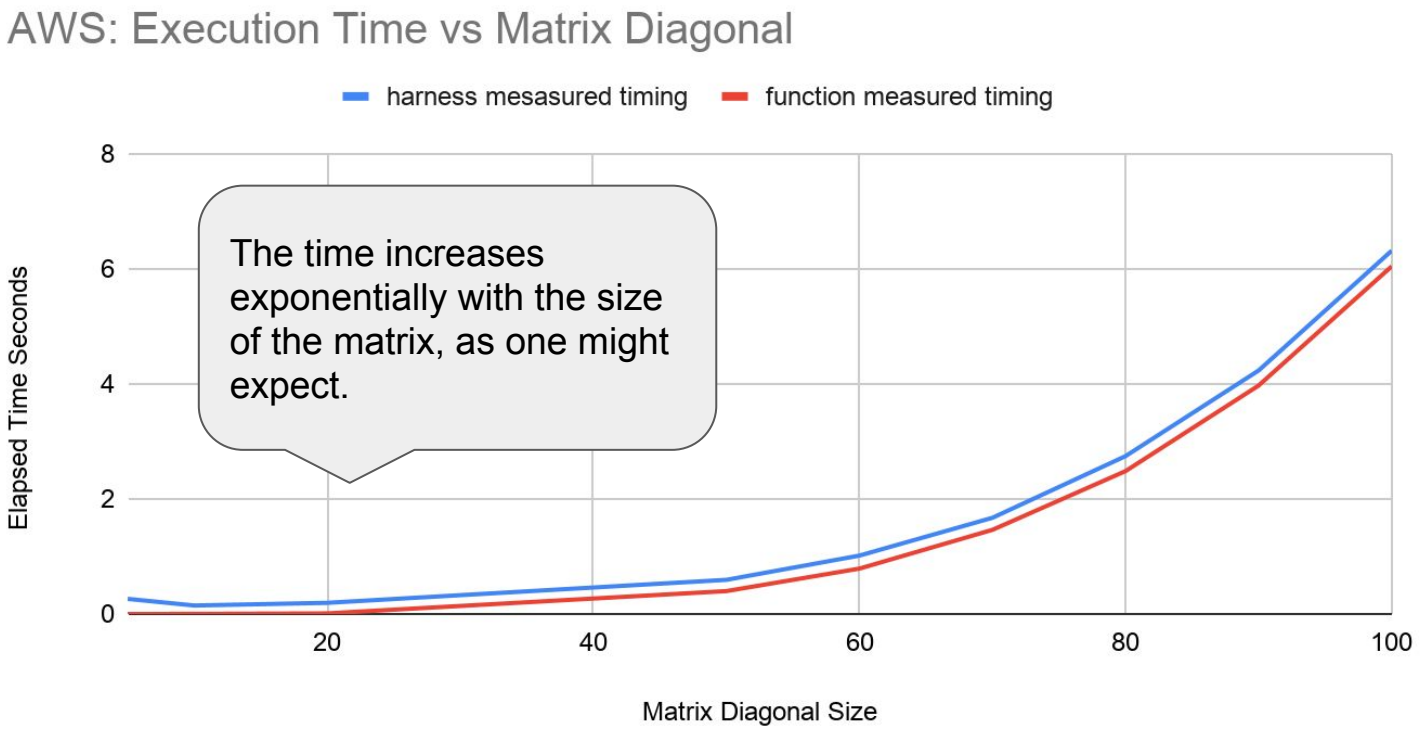
- Standard Matrix Multiplication Algorithm
- Targets:
  - Measure latency of network
  - Measure invocation/response delivery latency
    - Depends how well the clocks are synced
- Config
  - Python 3.8
  - 2048 mb memory
  - Synchronous invocation of target
- Measurements
  - Side length 5,10,20,50,60,70,80,90,100
    - Originally 5,10,50,100, but graphs were too "low resolution"

Test: AWS Matrix Multiplication

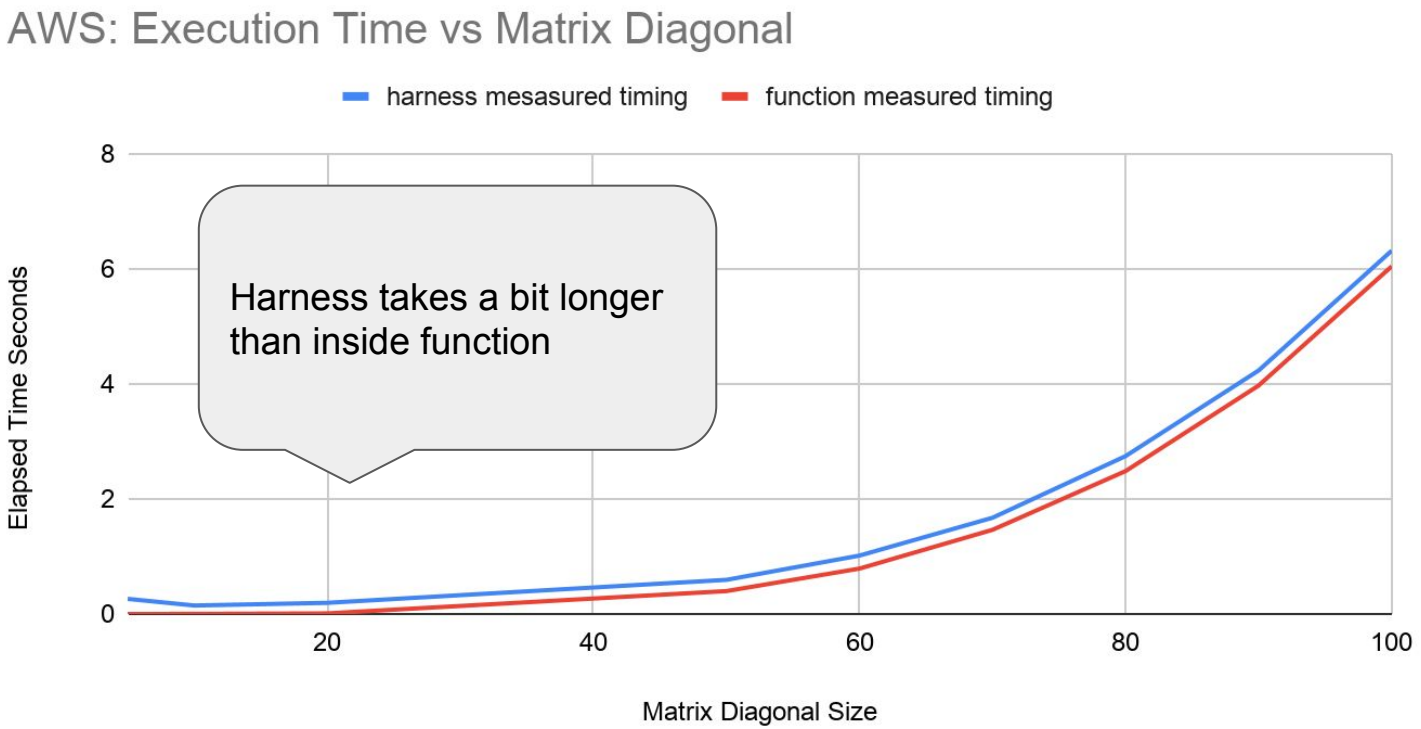


Average of 3 runs - I did 5 in the first test, and the numbers were close enough

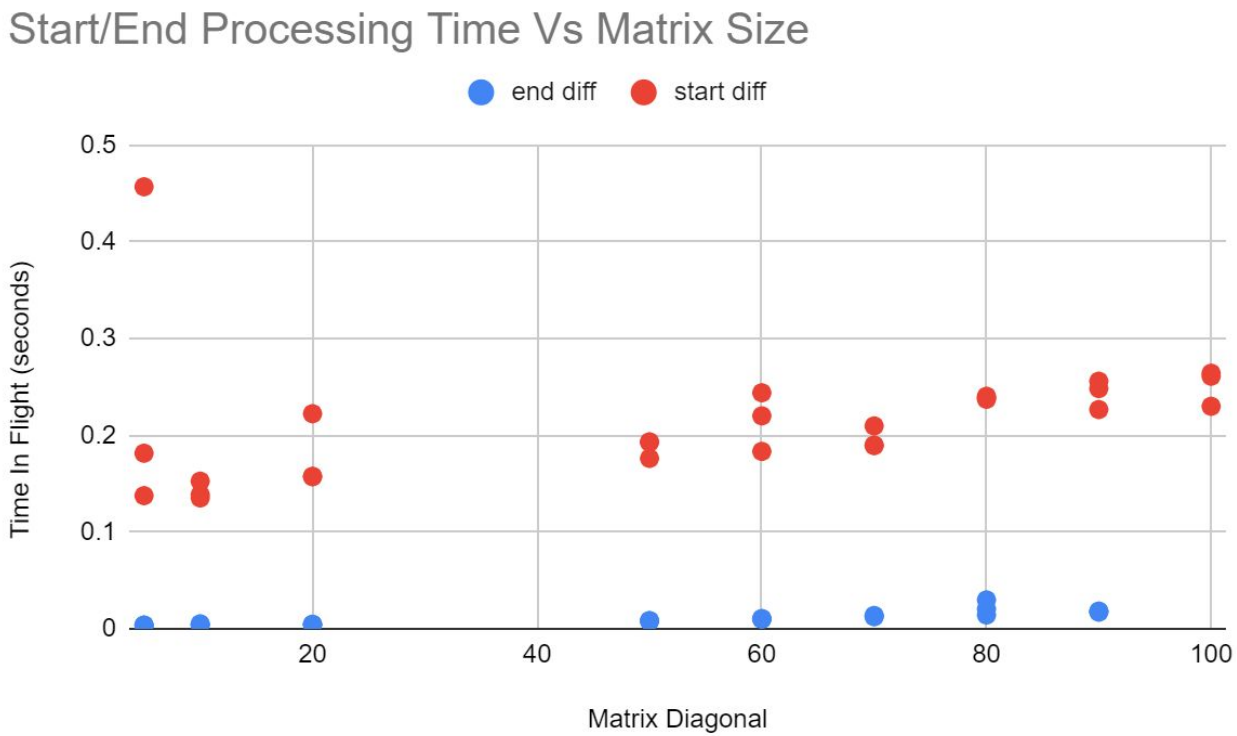
Test: AWS Matrix Multiplication



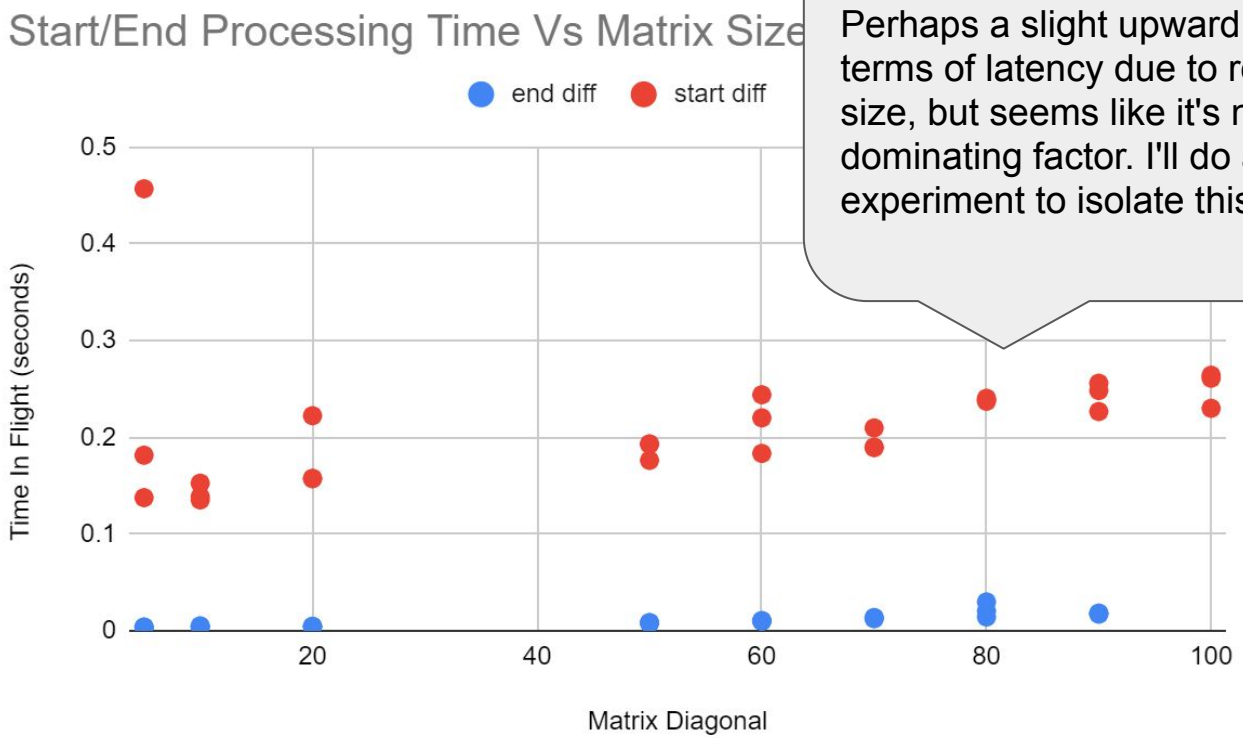
Test: AWS Matrix Multiplication



Test: AWS Matrix Multiplication

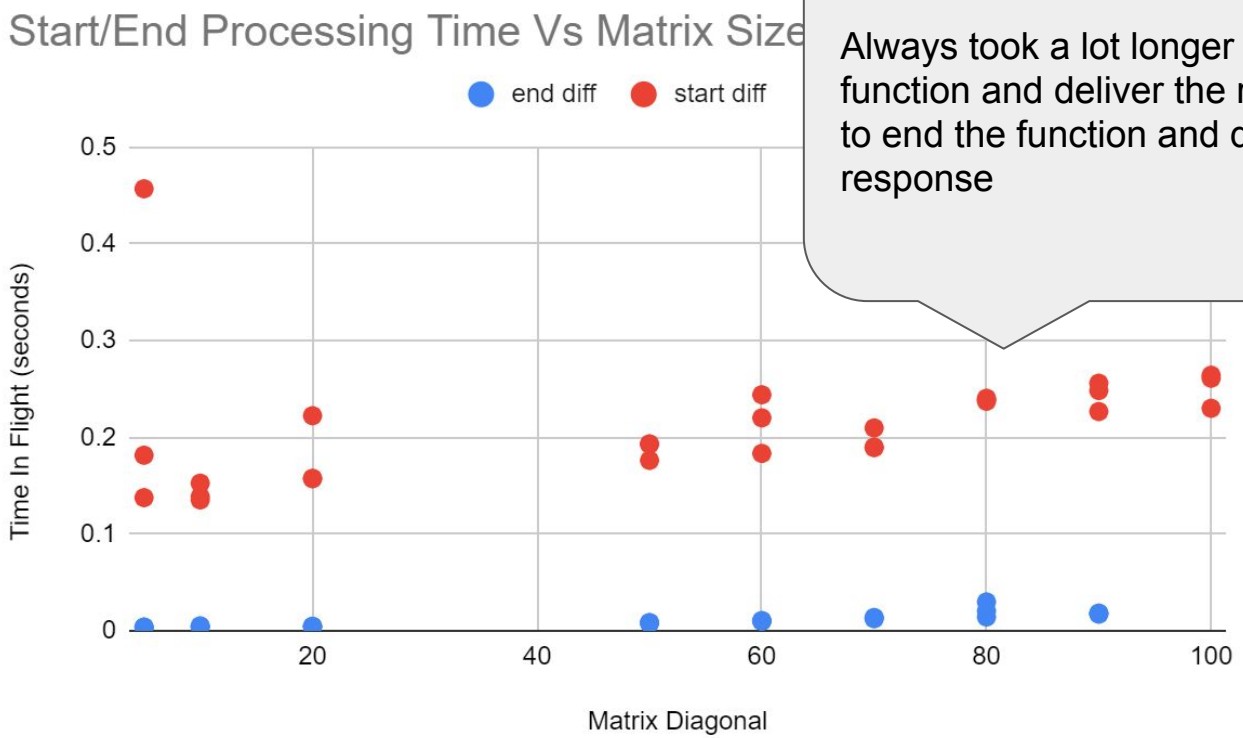


Test: AWS Matrix Multiplication



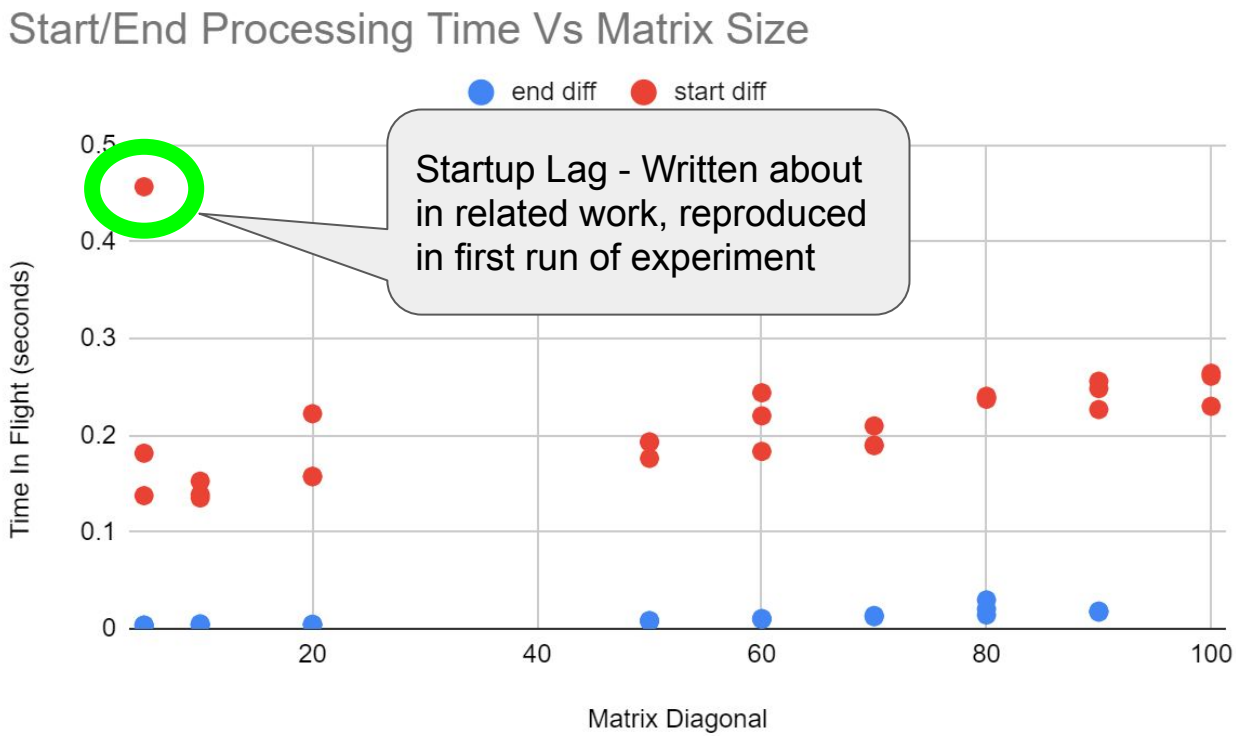
Perhaps a slight upward trend in terms of latency due to response size, but seems like it's not the dominating factor. I'll do another experiment to isolate this.

Test: AWS Matrix Multiplication



Always took a lot longer to start the function and deliver the request than to end the function and deliver response

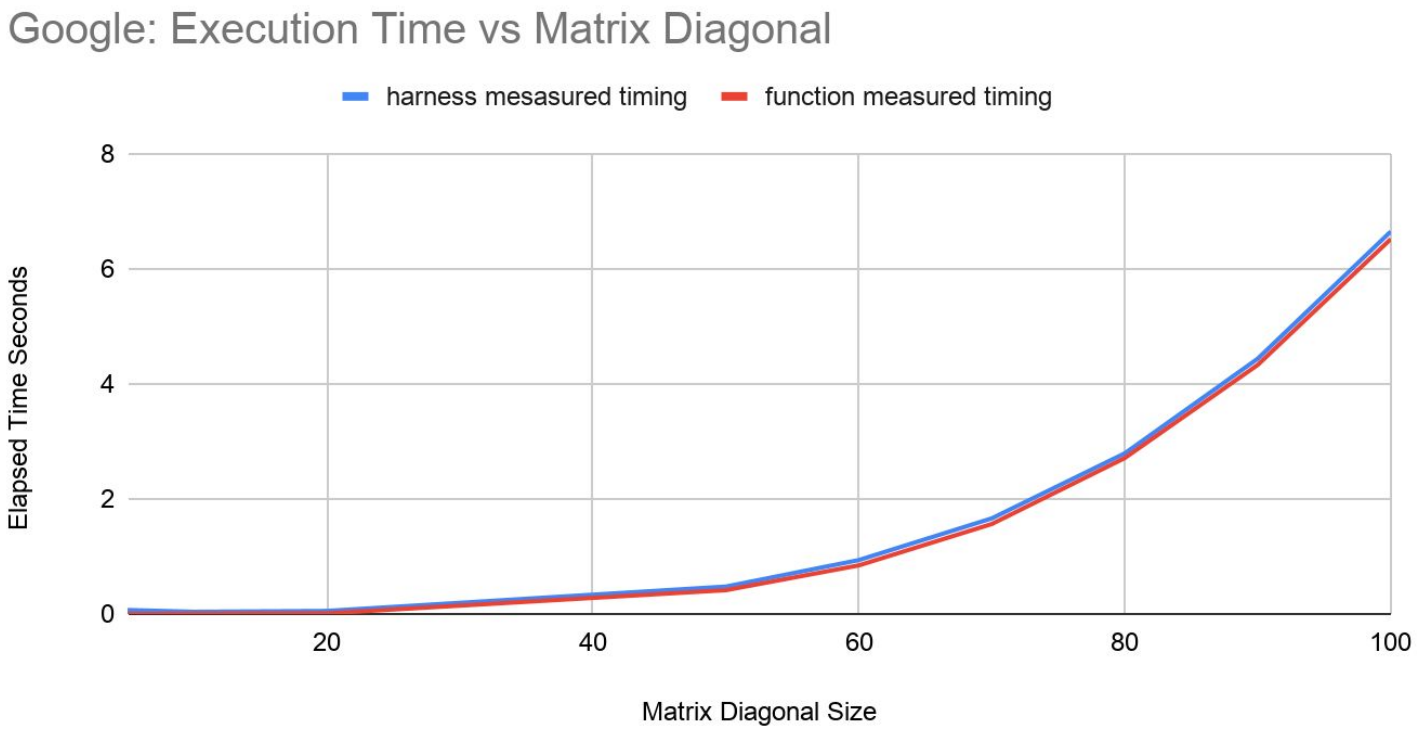
Test: AWS Matrix Multiplication



Test: AWS Matrix Multiplication

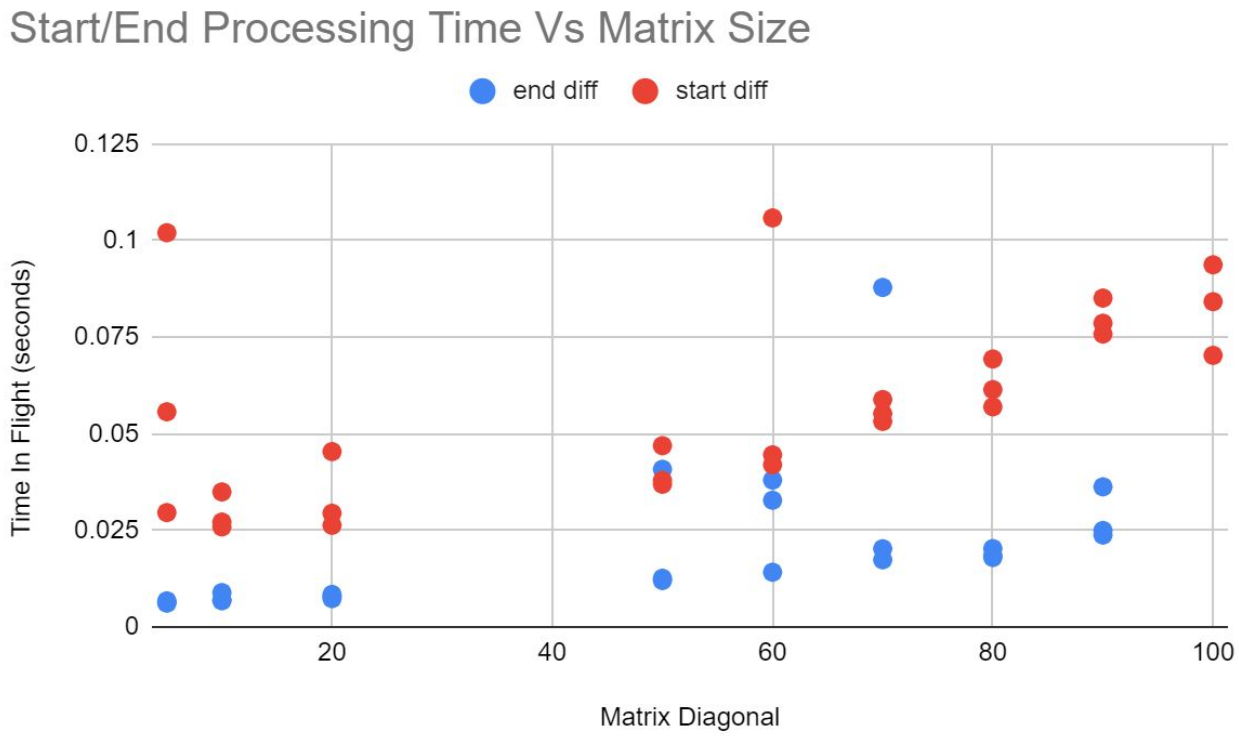
- Standard Matrix Multiplication Algorithm
- Targets:
  - Measure latency of network
  - Measure invocation/response delivery latency
    - Depends how well the clocks are synced
- Config
  - Python 3.7 (3.8 unavailable, I doubt it matters)
  - 2048 mb memory
  - Rest invocation of target
- Measurements
  - Side length 5,10,20,50,60,70,80,90,100
    - Originally 5,10,50,100, but graphs were too "low resolution"

Test: Google Matrix Multiplication

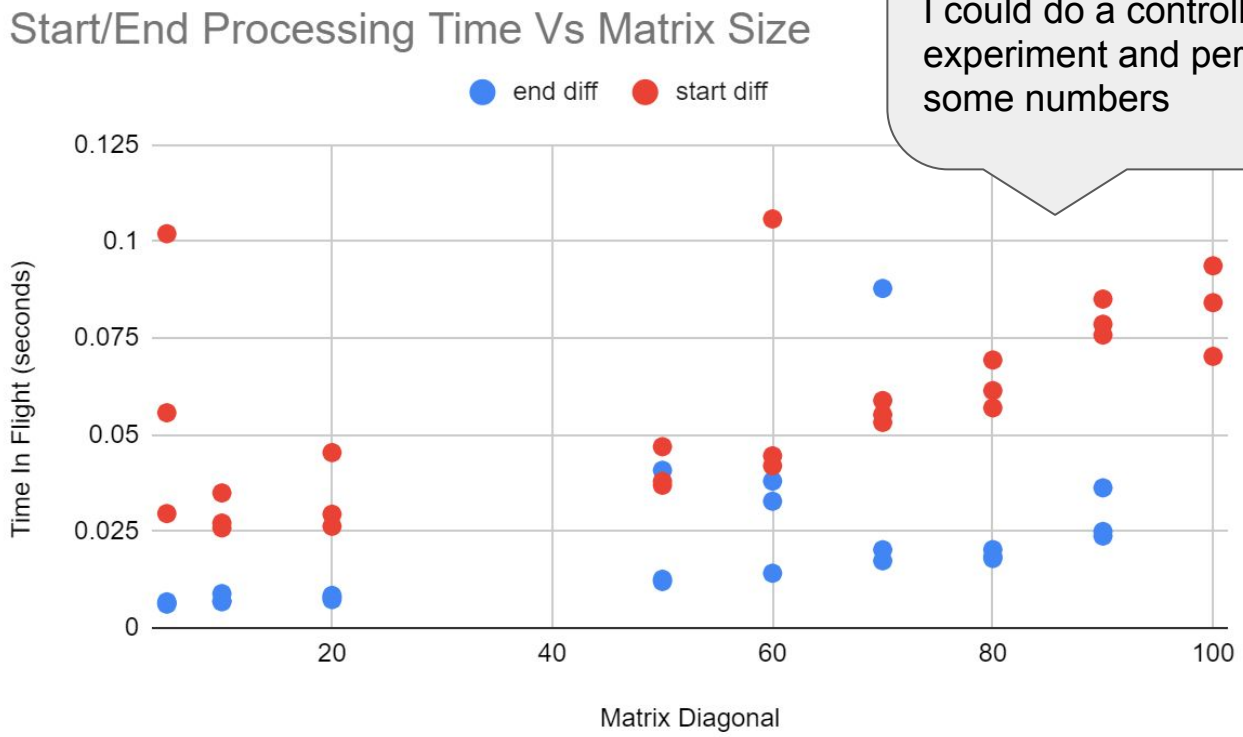


Average of 3 runs

Test: Google Matrix Multiplication

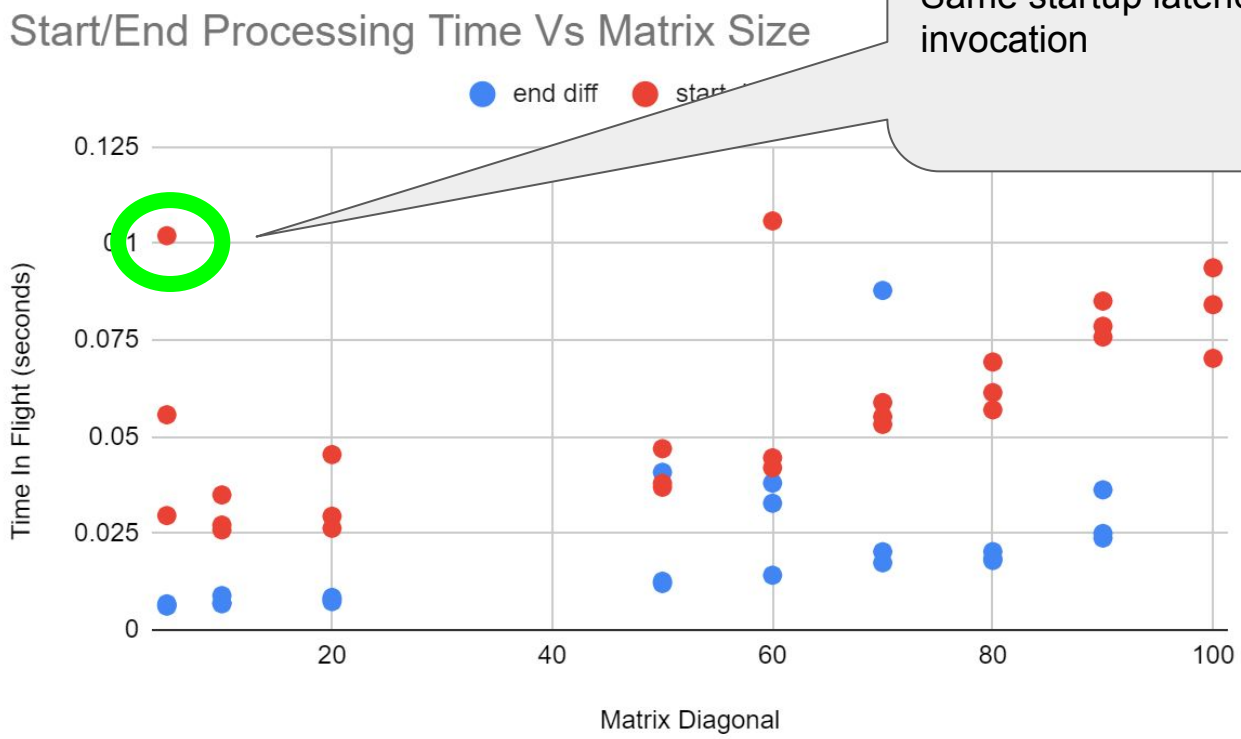


Test: Google Matrix Multiplication



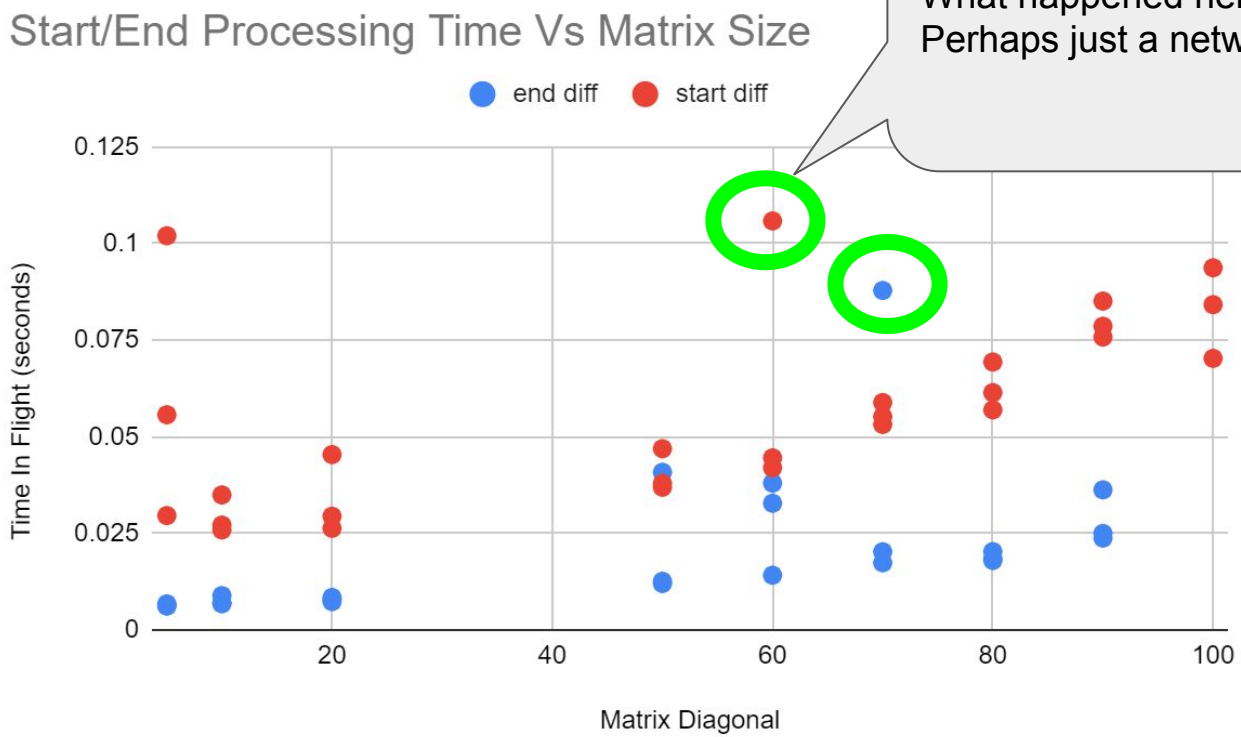
Perhaps more variability than Amazon, and start and end latencies seem closer.
I could do a controlled experiment and perhaps get some numbers

Test: Google Matrix Multiplication



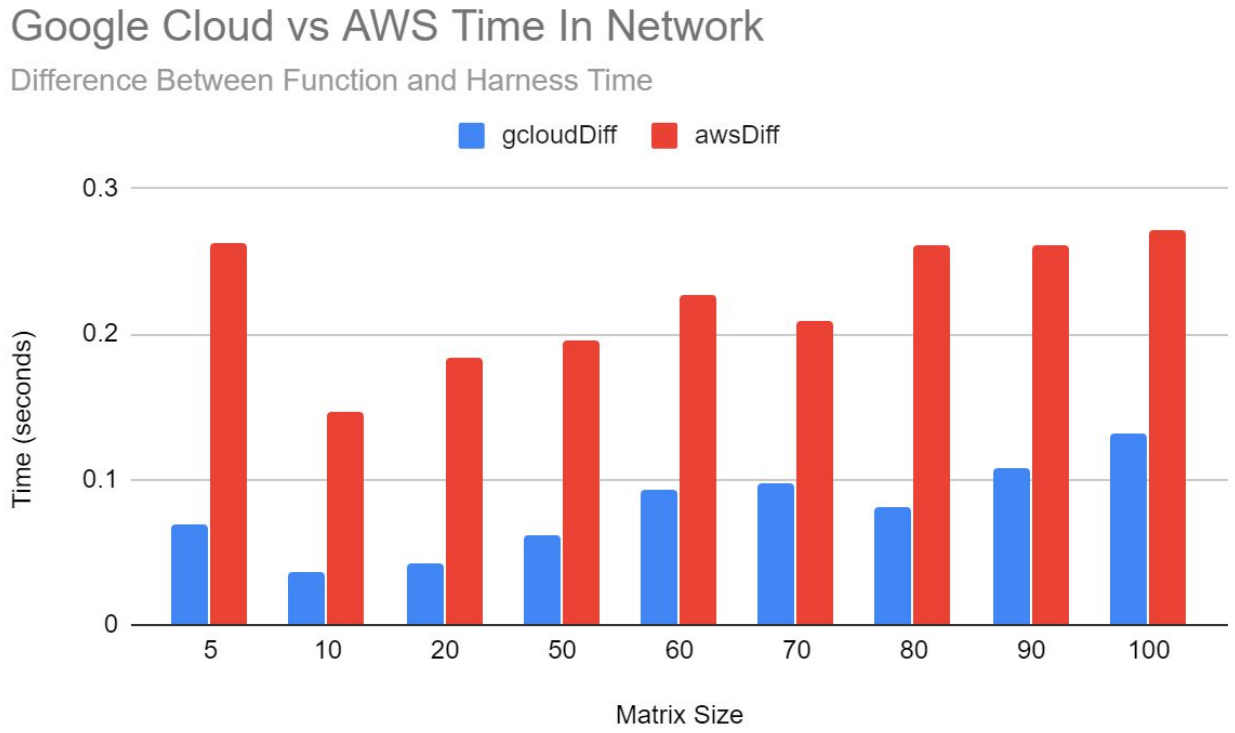
Same startup latency on first invocation

Test: Google Matrix Multiplication



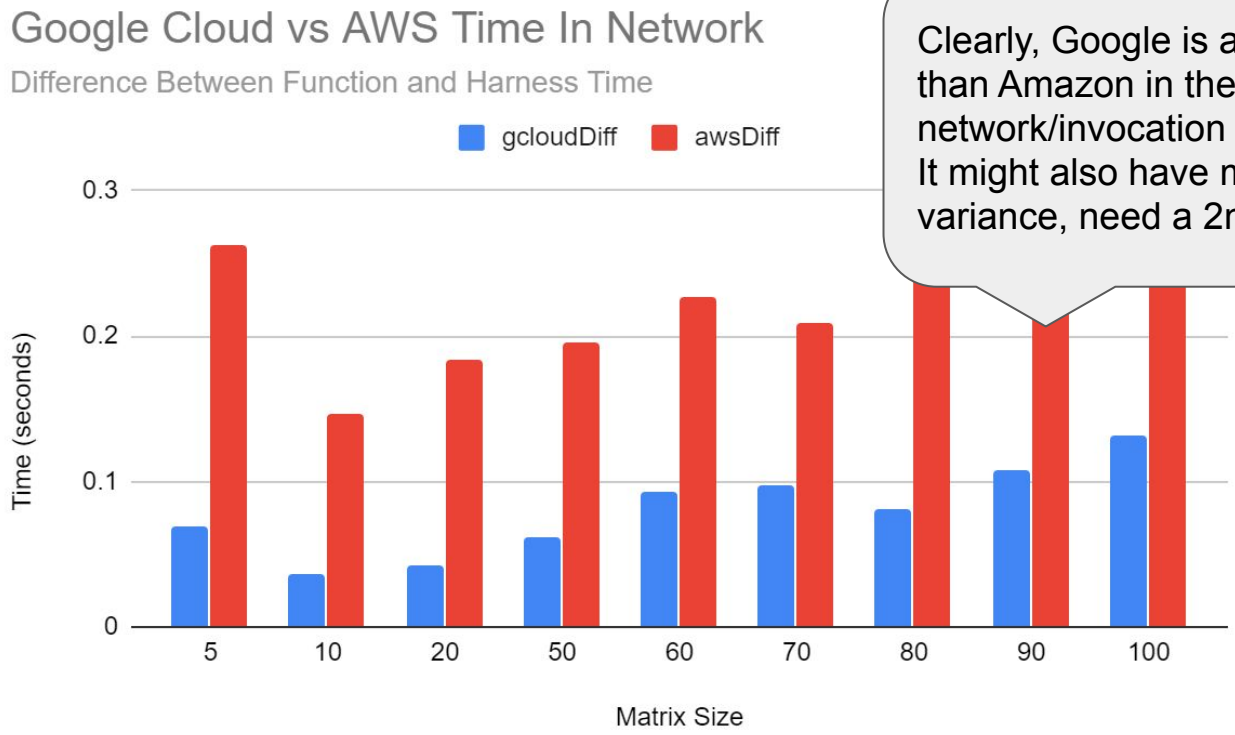
What happened here?
Perhaps just a network fluke?

Google Vs Amazon - Time Between Harness and Function



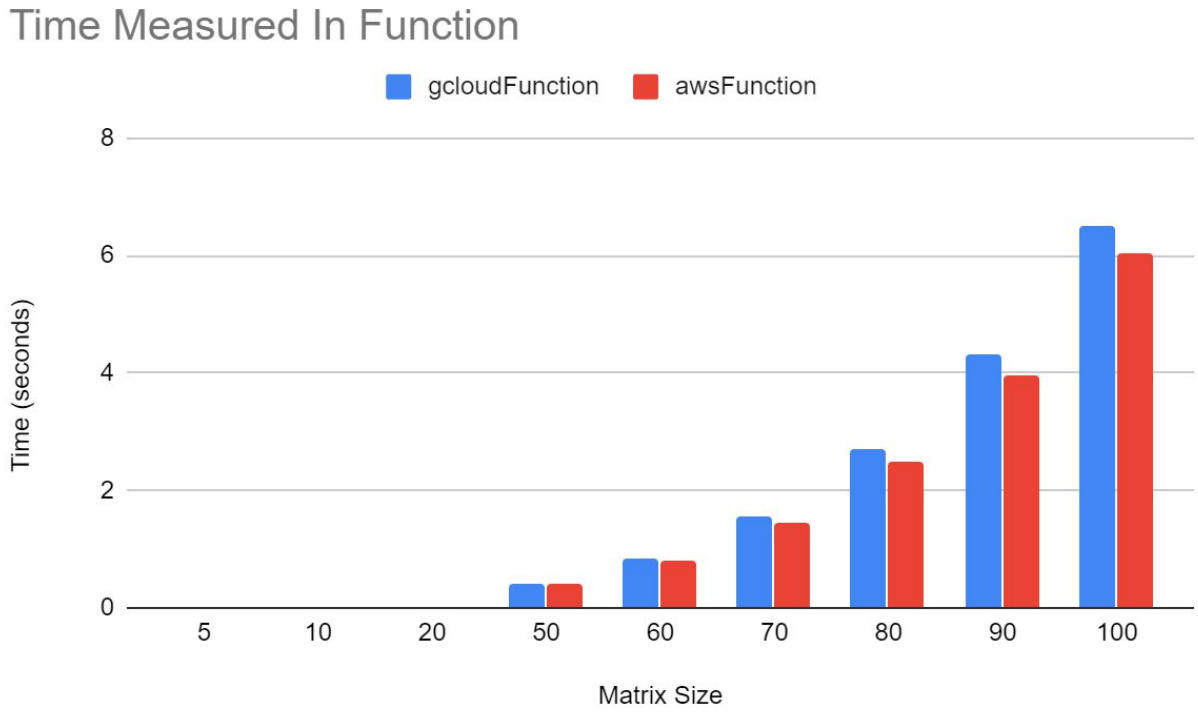
Average of 3 runs

Google Vs Amazon - Time Between Harness and Function



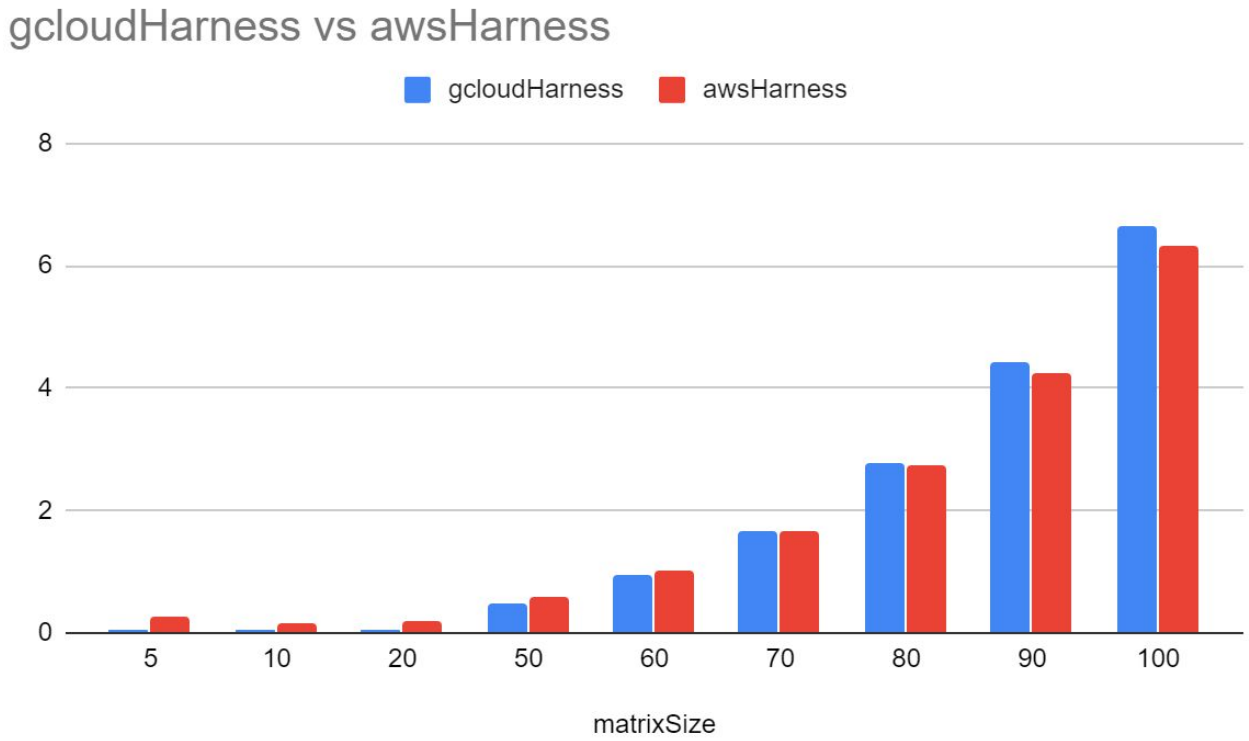
Clearly, Google is a lot faster than Amazon in the network/invocation domain. It might also have more variance, need a 2nd test.

Google Vs Amazon - Time in Function



Average of 3 runs

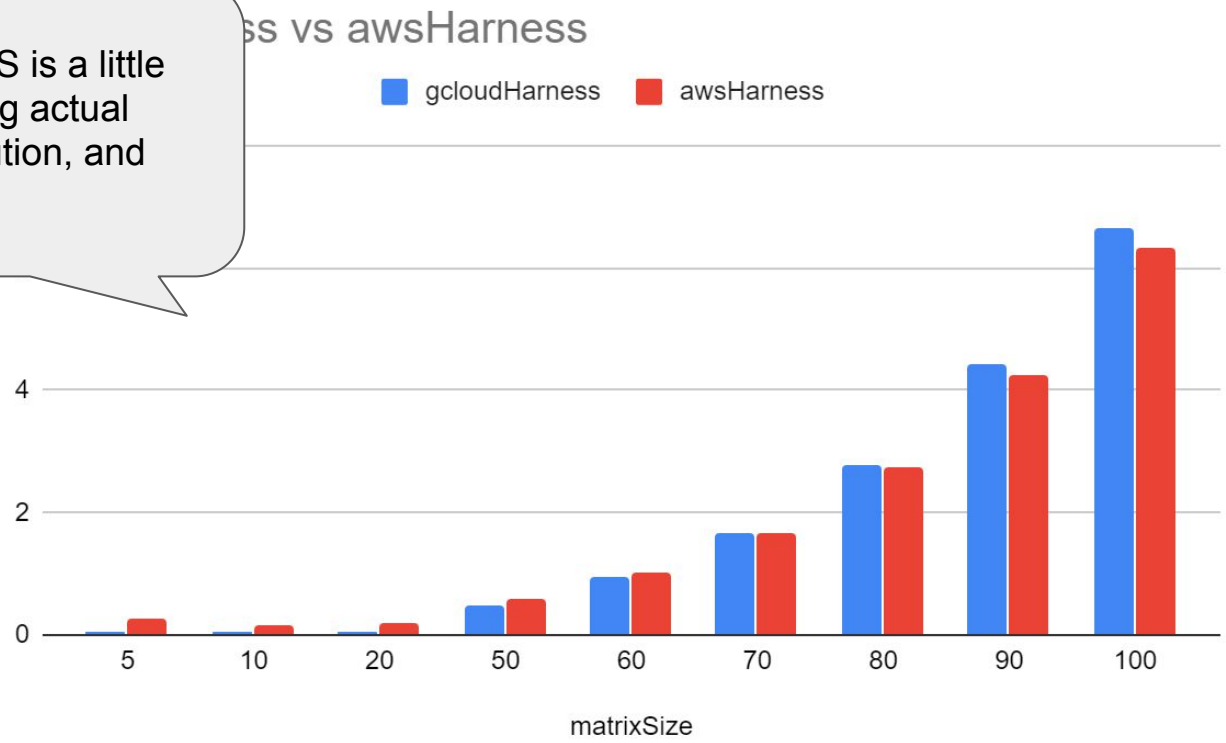
Google Vs Amazon - Time in Harness



Average of 3 runs

Google Vs Amazon - Time in Harness

Looks like AWS is a little bit faster during actual function execution, and overall



Average of 3 runs

Experiment Conclusions

- Amazon Cloud has a faster compute infrastructure for the 2 gb memory compute instance
  - Might need to vary the memory axis as well, to get different cpu setups
  - Doubt they have different cpus, just more or fewer of them
  - Wasn't using threading
- Google cloud might be better for network-intensive workflows
  - Need separate experiment to isolate this component

User Interface: Amazon Vs Google

- Permissioning:
  - Amazon - needed to configure one lambda to have permission to call another lambda with IAM
  - Google - can have lambda public accessible by default, can configure with IAM, or can have accessible to another lambda in the same package
- Runtimes/Language Options
  - AWS has many
  - Google has fewer
- Console
  - AWS was much faster to edit, save changes, test, could save tests
  - Google was slow, couldn't save tests and rerun them
    - Unexpected errors occurred

Programming: Amazon Vs Google

- Google - no internal interface, just beautiful HTTP/REST
- Amazon - poorly documented boto3 library needed, have to specify things at http level
  - CLI documentation did not work with python popen() - one would have thought aws cli was available in lambda container

```
def invokeTestFunction(inputObj):
    start=time.time()
    resp = requests.post(url='https://us-central1-reco-269716.cloudfunctions.net/matrixMultiplication', json=inputObj)
    end=time.time()
    fnOut=resp.json()
    return {"calledFunctionOutput":fnOut,"start":start,"end":end,"elapsed":(end-start)}
```

```
def invokeTestFunction(inputObj):
    client = boto3.client('lambda')
    inputJson=json.dumps(inputObj).encode('utf-8')
    start=time.time()
    response = client.invoke(
        FunctionName='945931053125:function:matrixMultiplication',
        InvocationType='RequestResponse',
        LogType='None',
        Payload=inputJson,
    )
    end=time.time()
    output = json.loads(response['Payload'].read())
    fnOut=json.loads(output['body'])
    return {"calledFunctionOutput":fnOut,"start":start,"end":end,"elapsed":(end-start)}
```

Next Experiments

- CPU intensive workflows
  - Matrix multiplication
- Network intensive workflows - network throughput vs latency to be considered
  - Take large chunk of data, pass it to function, do some work on it, pass it back
- Read from database, do math, return small result? Or just read from db?
- Profile IBM, Microsoft (same tests)
- Parallelism/Throughput
  - Bottleneck shared resource like DB?
  - Already been studied
  - Do lambdas give you multiple processors
    - Spawn 2 lambdas vs spawn 2 threads?

Do you have any ideas for good experiments?

Opinion on Providers ...so far

- AWS is a more mature cloud platform
  - More proprietary
  - Better UI
  - More robust permissioning with no loopholes
  - No unexpected errors happened at all
- Google is trying hard to make its platform more accessible
  - Lower startup learning time
  - Seems built on top of flask, and other things
  - Easy http invocation of lambda
  - URL from which to access
  - Flask developers will be familiar

Related Work

- https://arxiv.org/pdf/1905.11707.pdf Proposes a very similar infrastructure, has no results on proprietary systems
- https://oaciss.uoregon.edu/icpp18/views/includes/files/pos115s2-file1.pdf Also very similar, does look at aws, aims for GFLOPS, doesn't look at network speed
- https://www.faastest.com/ A commercial offering that seems similar, but I don't think has such granular measurements
- https://www.researchgate.net/profile/Johannes_Manner/publication/335691397_Impact_of_Application_Load_in_Function_as_a_Service/links/5d7614a7299bf1cb80931928/Impact-of-Application-Load-in-Function-as-a-Service.pdf Talks about exclusively load-testing and cpu time
- Very similar, but doesn't look at network throughput vs latency https://www.ise.tu-berlin.de/fileadmin/fg308/publications/2020/Online_Preprint___SAC_2020__Benchmarking_Elasticity_of_FaaS_Platforms_as_a_Foundation_for_Objective_driven_Design_of_Serverless_Applications.pdf

Related Work

- https://digitalcollection.zhaw.ch/bitstream/11475/7130/1/faaster-better-cheaper-archive.pdf Looks at concurrency testing actual cloud providers with an image processing framework
- https://www.cs.colostate.edu/~shrideep/papers/ServerlessComputing-IC2E-2018.pdf looks at concurrency and stress testing aws and azure as well
- https://www.usenix.org/system/files/conference/atc18/atc18-akkus.pdf startup latency considered, idle memory cost as well
- https://medium.com/the-theam-journey/benchmarking-aws-lambda-runtimes-in-2019-part-i-b1ee459a293d a blog post about using different lagnuage runtimes

Next Experiments

- CPU intensive workflows
  - Matrix multiplication
- Network intensive workflows - network throughput vs latency to be considered
  - Take large chunk of data, pass it to function, do some work on it, pass it back
- Read from database, do math, return small result? Or just read from db?
- Parallelism/Throughput
  - Bottleneck shared resource like DB?
  - Already been studied
  - Do lambdas give you multiple processors
    - Spawn 2 lambdas vs spawn 2 threads?

Do you have any ideas for good experiments?