Evaluation of Production Serverless Computing Environments

Hyungro Lee, Kumar Satyam and Geoffrey C. Fox

July 2, 2018 Third International Workshop on Serverless Computing (WoSC), San Francisco, CA

Indiana University Bloomington

Background

- Minimum granularity of infrastructure provisioning
- Rich set of event handlers (triggers, invocation methods)
- Support in distributed data processing
- Pay-as-you-execute
  - Cost effectiveness (Migrating IaaS to FaaS)

Problems

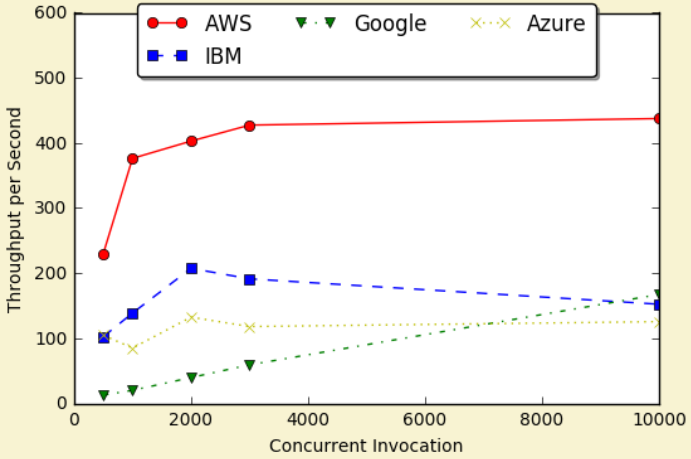
- Equivalent behaviors to function performance
- Elasticity for concurrent executions
- DevOps issues- continuous development/integration of functions
- Early stage of development on public clouds

Contribution

- Comprehensive performance evaluation of existing the serverless platforms w.r.t CPU, File I/O and network intensive workloads
- Summary of available features, runtimes and limitations

Function Throughput

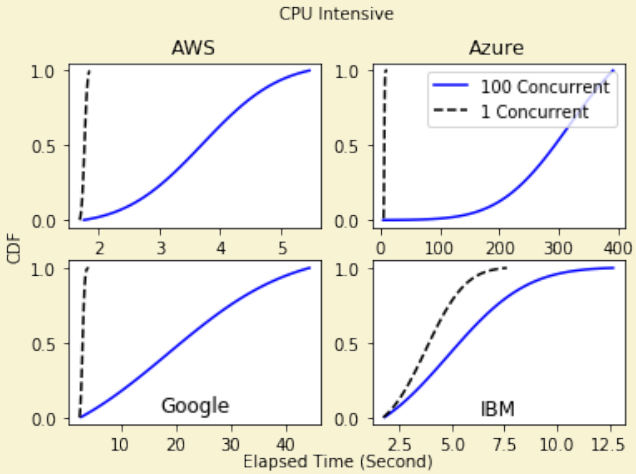
- 500, 1k, 2k, 3k and 10k concurrent invocations
- Trigger:
  - Amazon: Python boto3 library
  - Google: bucket storage
  - Azure: HTTP REST API
  - IBM: HTTP REST API



CPU Intensive Function

Function task
- Matrix multiplication with a size of 512
- Written by javascript (nodeJS runtime)

Configuration
- Amazon: 1.5GB Mem/5min timeout
- Azure: N/A mem/10min
- Google:2G mem/9min
- IBM: 512M mem/5min

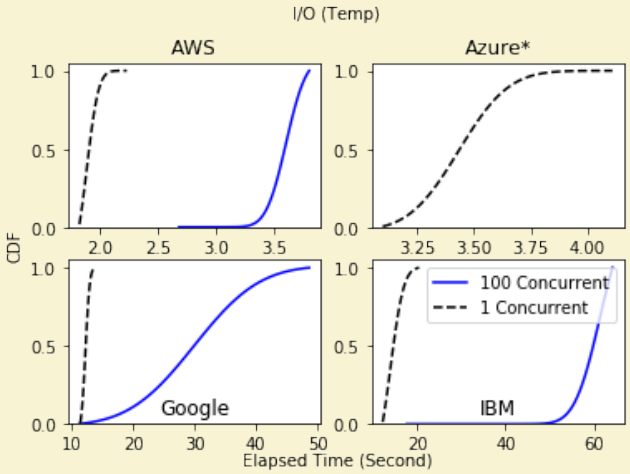


Platform	Execution set	Mean in sec	Diff
AWS Lambda	1 func invocation x100 times	1.77	
	100 func invocations x1 time	3.72	2.11x
Azure Functions	1x100	6.78	
	100x1	319.24	47.06x
Google Functions	1x100	3.09	
	100x1	18.79	6.07x
IBM OpenWhisk*	1x100	3.80	
	100x1	4.88	1.28x

File I/O Intensive Function

Function task:

- 1 random write + 1 random read
- 100MB size of a file in a temp directory
- read of 512bytes with random offset
- write with fsync
- Python runtime



Configuration

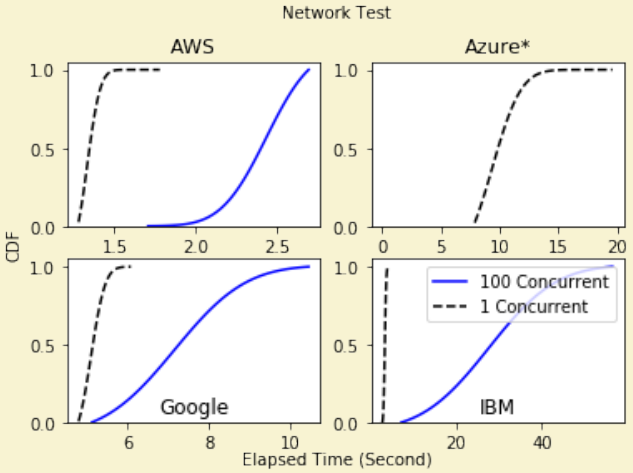
- Amazon: 1.5GB Mem/5min timeout
- Azure: N/A mem/10min
- Google:2G mem/9min
- IBM: 512M mem/5min

Platform	Execution set	Mean in sec (std)	Diff	Read (MB/s)	Write (MB/s)
AWS Lambda	1 func invocation x100 times	1.88 (0.08)		152.98	82.98
	100 func invocations x1 time	3.61 (0.14)	1.92x	92.95	39.49
Azure Functions	1x100	3.44 (0.17)		423.92	44.14
	100x1	failed (device busy)	-	-	-
Google Functions	1x100	12.26 (0.55)		55.88	9.44
	100x1	30.11 (8.39)	2.46x	54.14	3.57
IBM OpenWhisk*	1x100	14.04 (2.33)		68.23	7.86
	100x1	61.55 (4.49)	4.38x	33.89	0.50

Network Intensive Function

Function task:

- Transfer 100MB size of a file from object storage
- AWS S3/Azure Blob/Google Bucket/IBM Object storage
- nodeJS runtime



Configuration

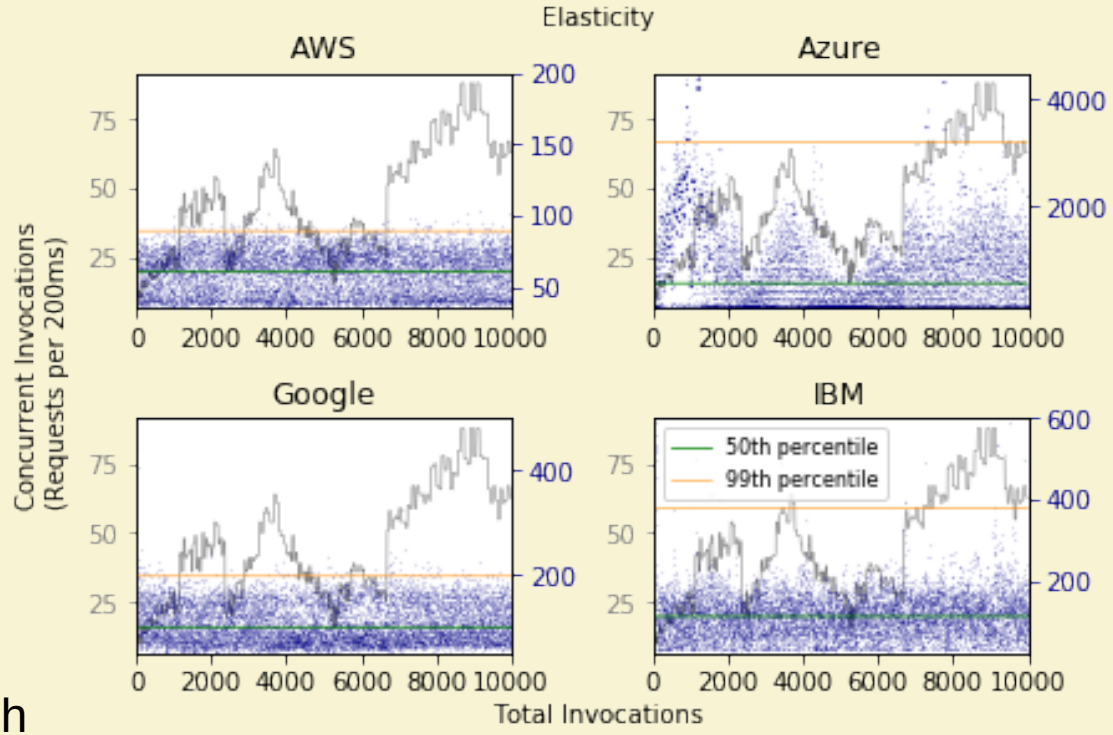
- Amazon: 1.5GB mem/5min timeout
- Azure: N/A mem/10min
- Google:2G mem/9min
- IBM: 512M mem/5min

Platform	Execution set	Mean in sec (std)	Diff
AWS Lambda	1 func invocation x100 times	1.34 (0.06)	
	100 func invocations x1 time	2.44 (0.21)	1.82x
Azure Functions	1x100	9.42 (1.93)	
	100x1	failed	-
Google Functions	1x100	5.12 (0.27)	
	100x1	7.19 (1.37)	1.40x
IBM OpenWhisk*	1x100	3.61 (0.26)	
	100x1	27.97 (12.25)	7.75x

Elasticity

Function task
- building a small binary tree in 100 ms
- 10,000 invocations made in a minute
- 200ms interval per invocation with 10 to 90 concurrency

Figure Details
- gray line: Number of invocations
- blue dot: response time of each function
- yellow horizontal line: 99%
- green horizontal line: 50%

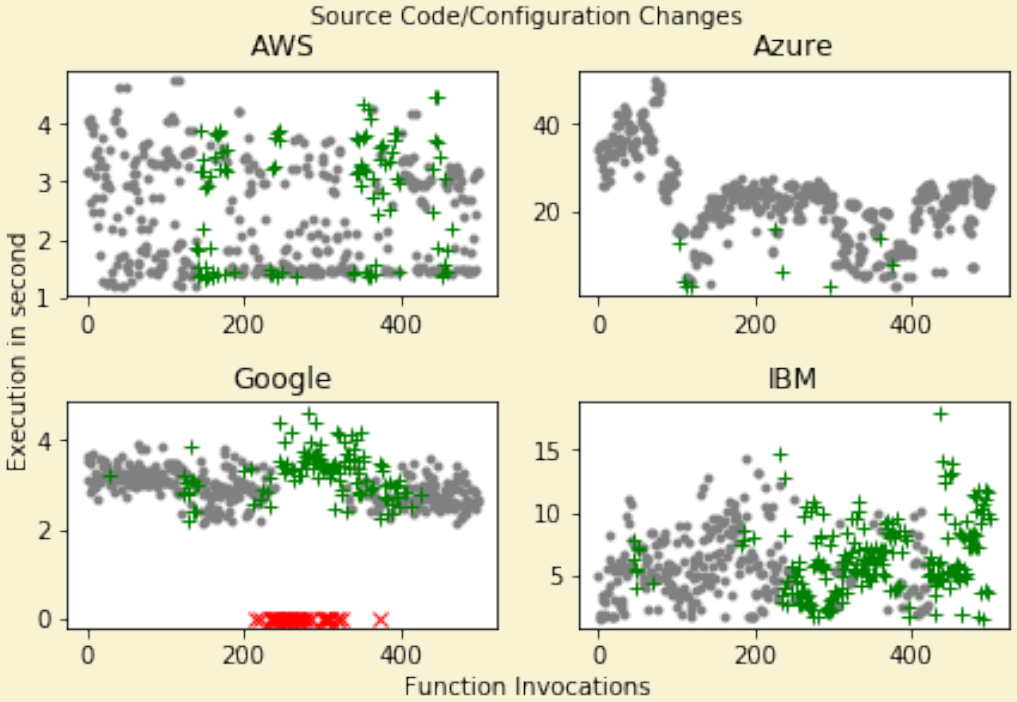


Continuous Development

Function task
- Small computations run in 1-2 seconds
- 500 total invocations
- 10 concurrency
- completed in 10 seconds

Actions
- Function code is updated prior 200 invocations
- Configuration is changed in the next 200 invocations

Figure Details
- gray dot: response time of each function
- green +: new instance
- red x: failed instance



Feature comparison

Item	AWS Lambda	Azure Functions	Google Functions	IBM OpenWhisk
Runtime	Node.js – 4.3, 6.10 Python - 2.7, 3.6 Java 8 C# - 1.0, 2.0 Golang 1.x	(Default) Node.js - 6.11, 8.4 C# 1.0, 2.0 F# 4.6, (Experimental) batch, bash, php, powershell, python 2.7, typescript, Java 8	Node.js 6.11.5 (Python 2.7)	Node.js – 6,8 Python – 2.7, 3.6 Java 8, C#, swift, php, docker
Memory Limit	128 to 3008MB (with 64mb increments)	1536MB (actual usage)	128 to 2048MB (with 256, 512, and 1024MB in between)	128 to 512MB (with 32mb increments)
Timeout	300 sec	600 sec	540 sec	600 sec
Code size	50MB (250MB – compressed)	n/a	100MB (500MB – compressed)	48MB
Triggers	19 triggers (e.g. S3, dynamoDB, CloudWatch Logs, Events)	17 triggers (e.g. Blob storage, Cosmos DB, Event Hubs)	3 triggers (e.g. HTTP, Pub/Sub, Storage Bucket)	6 triggers (e.g. Cloudant, Message Hub, Github)
Base OS	Amazon Linux	Windows NT	Debian GNU/Linux 8 (Jessie)	Alpine Linux

Conclusion

- Concurrent executions for distributed workloads
- Elasticity for dynamic applications

Future Work

- Open source serverless framework
  - OpenWhisk, Kubeless, fnproject, fission
- Additional runtimes with extra libraries
  - tensorflow, numpy
- Common functions to share in public
- Secured layer for connected services

Thank you!

Questions?