Northeastern University - Seattle

CS6650 Building Scalable Distributed Systems

Professor Ian Gorton

Building Scalable Distributed Systems

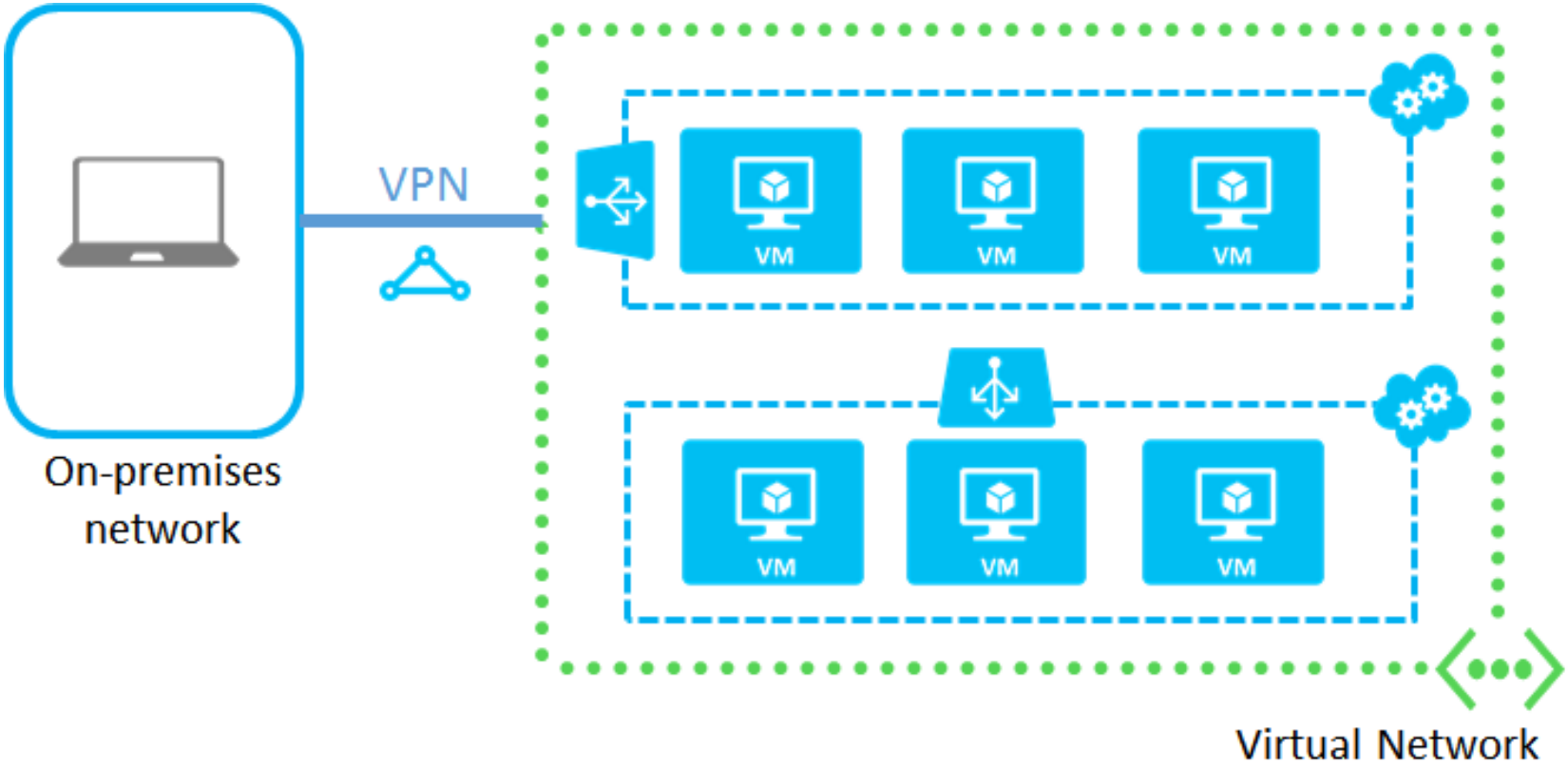
Week 10 – Serverless Computing

Outline

- Why serverless computing?
- Google App Engine Overview
- Some Experimental Results
- AWS Lambda Overview

Why Serverless Computing

Think traditional cloud



VM-based processing

Provision and configure VMs

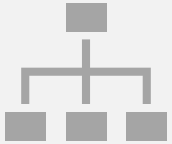
- Choose instance type
- Install software
- Instance monitoring
- On-going maintenance

As load grows

- Load balancing
- Increase database capacities
- Application monitoring
- Scale down during low usage periods to save costs

VM-based Cloud Apps – Costs

On-going monitoring and management



System administration

Potentially unused resources

Serverless Computing

Aka function-as-a-service (FaaS)

No VMs to provision and manage

No costs if not running code

Auto-scaling up and down

FaaS platforms

Examples:

- AWS lambda
- Google App Engine
- Azure Functions

Simply upload code to deploy:

- Each platforms supports a variety of languages/frameworks

Automatic horizontal scaling

- Configurable for specific workloads

Triggered by:

- HTTP request
- Events (eg S3 object update, SQS message arrival)

Trade-offs?

Restrictions

Stateless

- All state needs to be externalized/persistent between invocations

Execution duration

- Maximum typically 60-300 seconds by default
- Configurable

Startup latency

- Cold starts
- Warm starts
- Application/language/environment dependent

Costs

- Pay per invocation of a function
  - **Requests:** around $0.2 per 1M executions across the board, across all providers
  - **CPU & RAM: ~** $0.000067 per GB-second
  - Can be more expensive than VMs if utilization high
- API Gateway …..

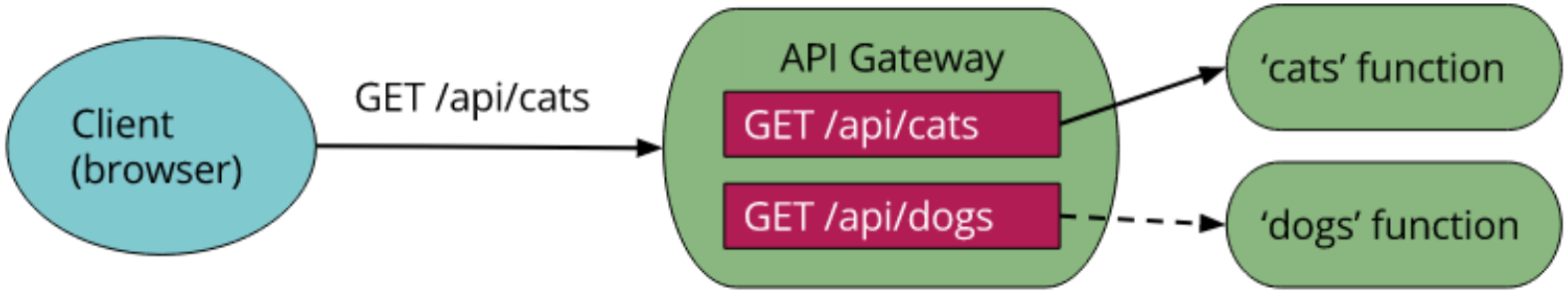
	IBM OpenWhisk	AWS Lambda	Azure Functions	GCP Functions
Requests	N/A	$0.2 per 1M requests	$0.2 per 1M requests	$0.4 Per 1M requests
GB/s (compute time)	$0.000017 per GB-s	$0.000016 per GB-s	$0.000016 per GB-s	$0.00000231/GB-s
Data Transfer (IN)	Free	$0.1-$0.2 between VPCs/regions	Free	Free
Data Transfer (OUT)	$0.05- $0.09 per GB	$0.05- $0.09 per GB	$0.05- $0.09 per GB	$0.12 per GB
API Gateway	Free	$3.50 per 1M calls	Free	$3.00 per 1M calls

https://medium.com/@amiram_26122/the-hidden-costs-of-serverless-6ced7844780b

API Gateway

- Similar to the façade pattern
  - E.g. HTTP, WebSockets
- Clients call Gateway and Gateway routes request to application (microservice) endpoints
  - Authorization
  - Monitoring
  - API version management
  - Other stuff input validation, traffic tetc
- Results sent to API Gateway and relayed to client
- Cloud-provider managed server

API Gateway

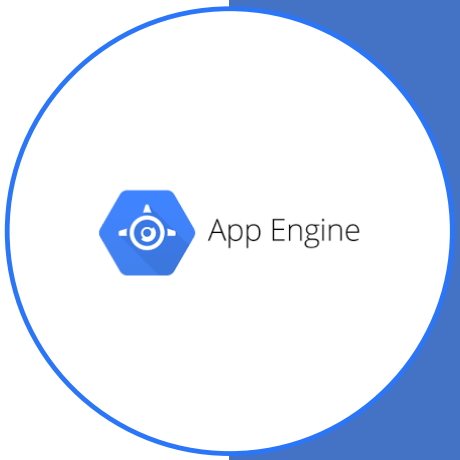


https://martinfowler.com/articles/serverless.html#WhatIsServerless

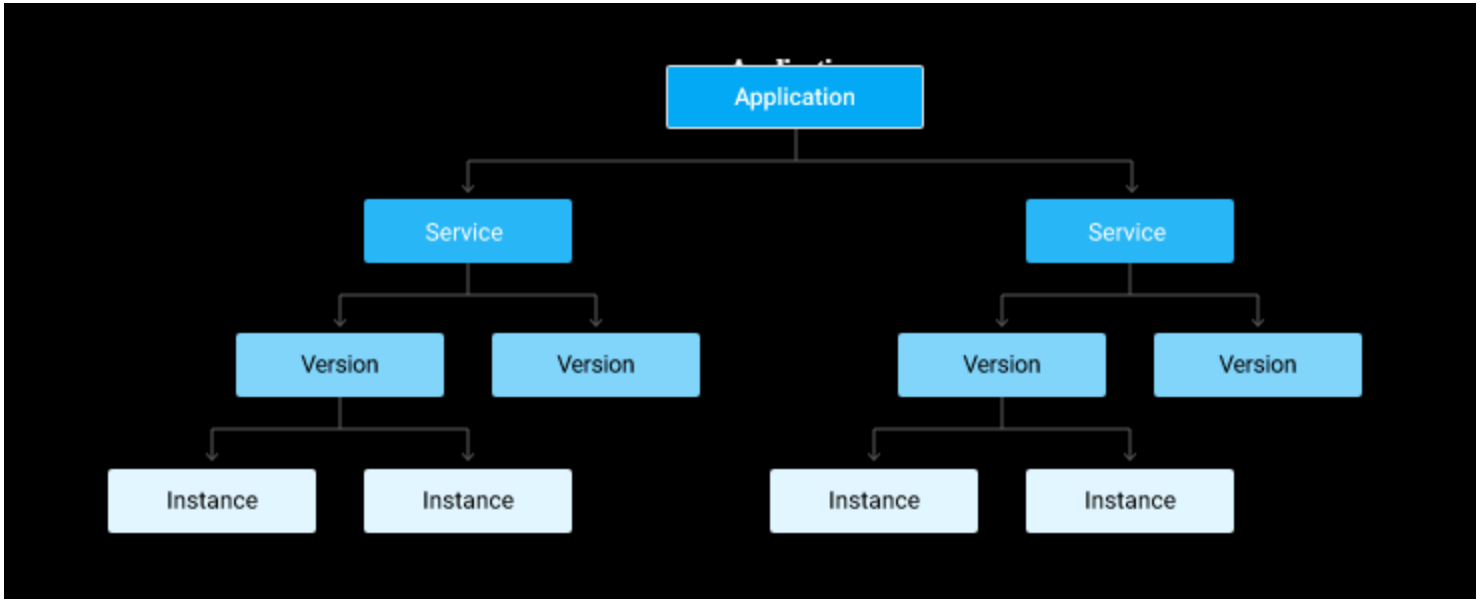
Google App Engine

GAE

- GAE app comprises
  - One or more services
  - Services can be in different languages/configured differently
  - Service can have multiple deployed versions
  - Versions have multiple instances that handle requests



GAE Overview



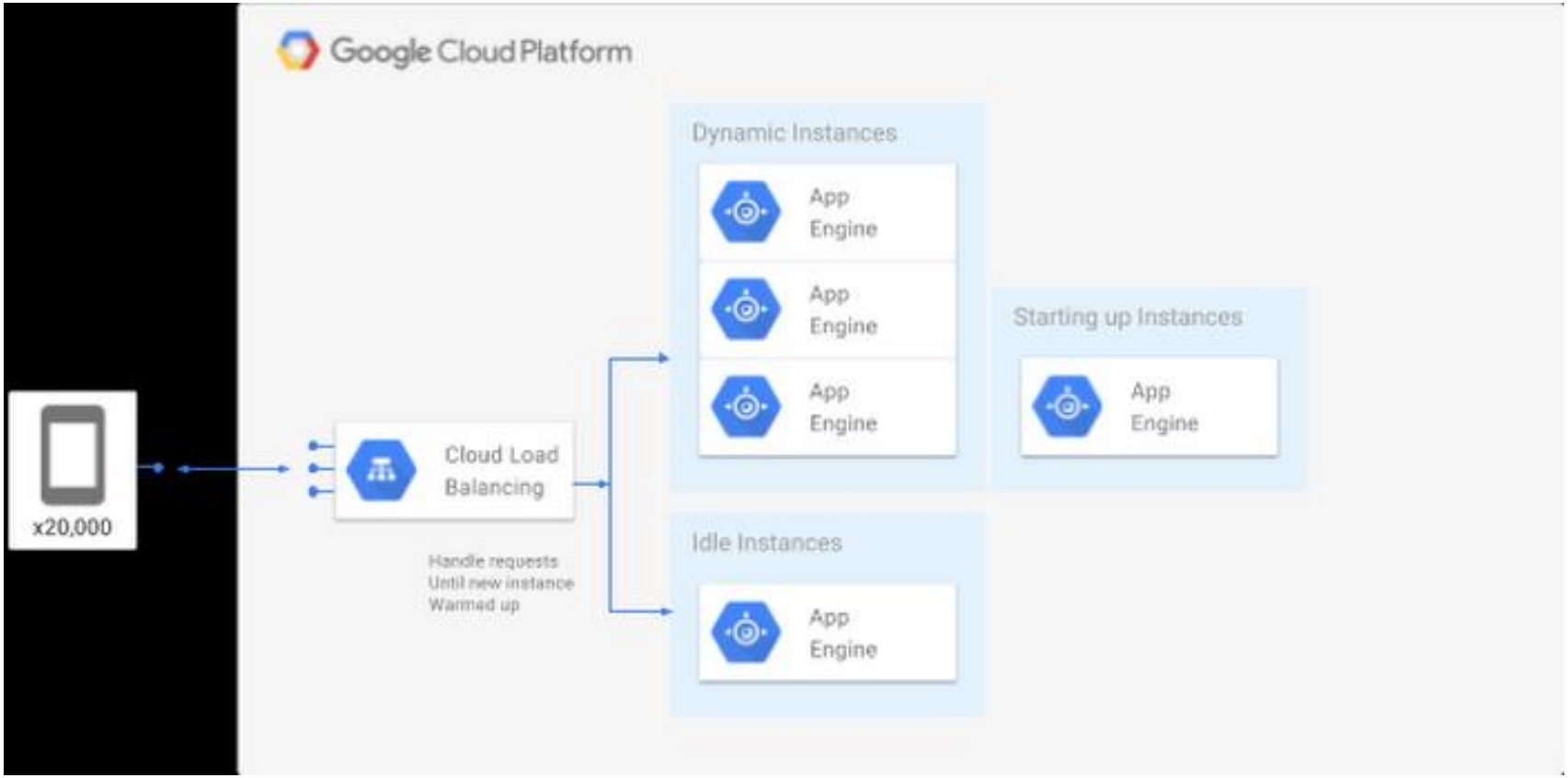
App Engine Services

- Equivalent to a microservice
  - App source code
  - API
  - Configuration
  - Version
- Can use traffic splitting to perform A/B tests across different versions
- Services/versions need unique names, eg:
  - http://my-version.my-service.my-project-id.appspot.com

Instances

- Auto-scaled by default
- Requires a new runtime to be deployed
  - JVM
  - Python VM
  - Etc
- Startup costs per instance
- Min/max instances
- Unused instances removed (scale down)
  - Pay by 15 mins intervals
- Instance resources (CPU/memory)

GAE Request Handling

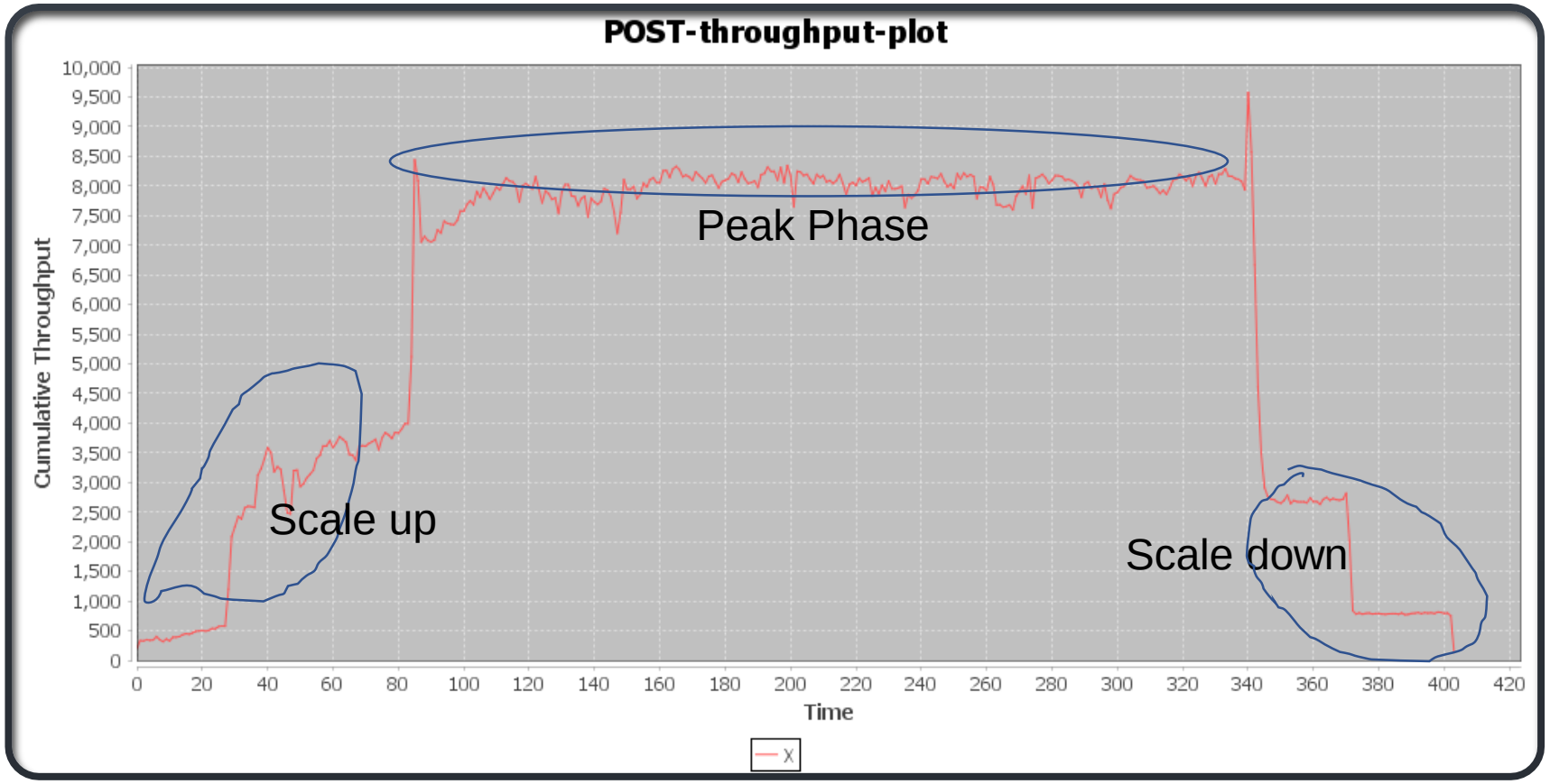


Auto-scaling – app.yaml

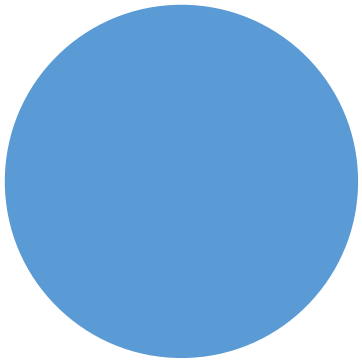
- By default:
  - When an instance reaches 60% CPU, the service is scaled
  - When an instance has 10 concurrent requests, the service is scaled
- Actually - by default it is 6, as the target throughput utilizations is set to 0.6 (* number of concurrent requests)
- Can specify minimum and maximum wait times for a request before scaling a service
  - Default 30 msec


IT'S COMPLICATED

Some Experimental Results



Typical Load test profile

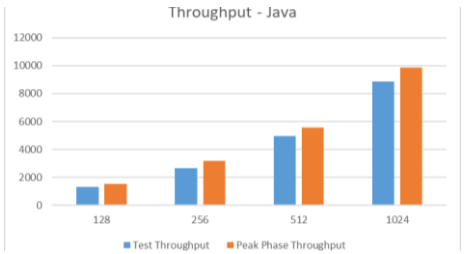


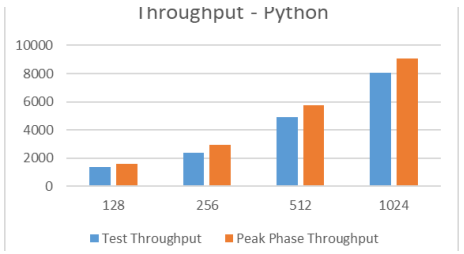

So let's start simple

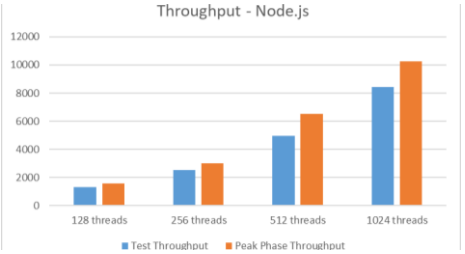
How much does programming language effect performance and costs?

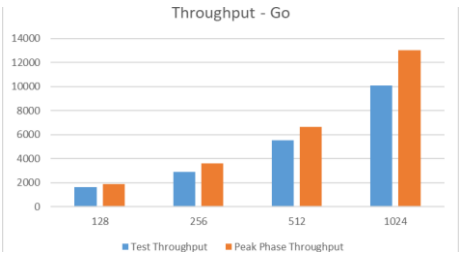
Experiment

- Google app engine server
  - Same API
  - Same database model and operations (Google Datastore)
  - Same client and test load
  - Same B1 instances, default autoscaling settings
- 4 server implementations
  - Go
  - Java
  - Python
  - Node.js

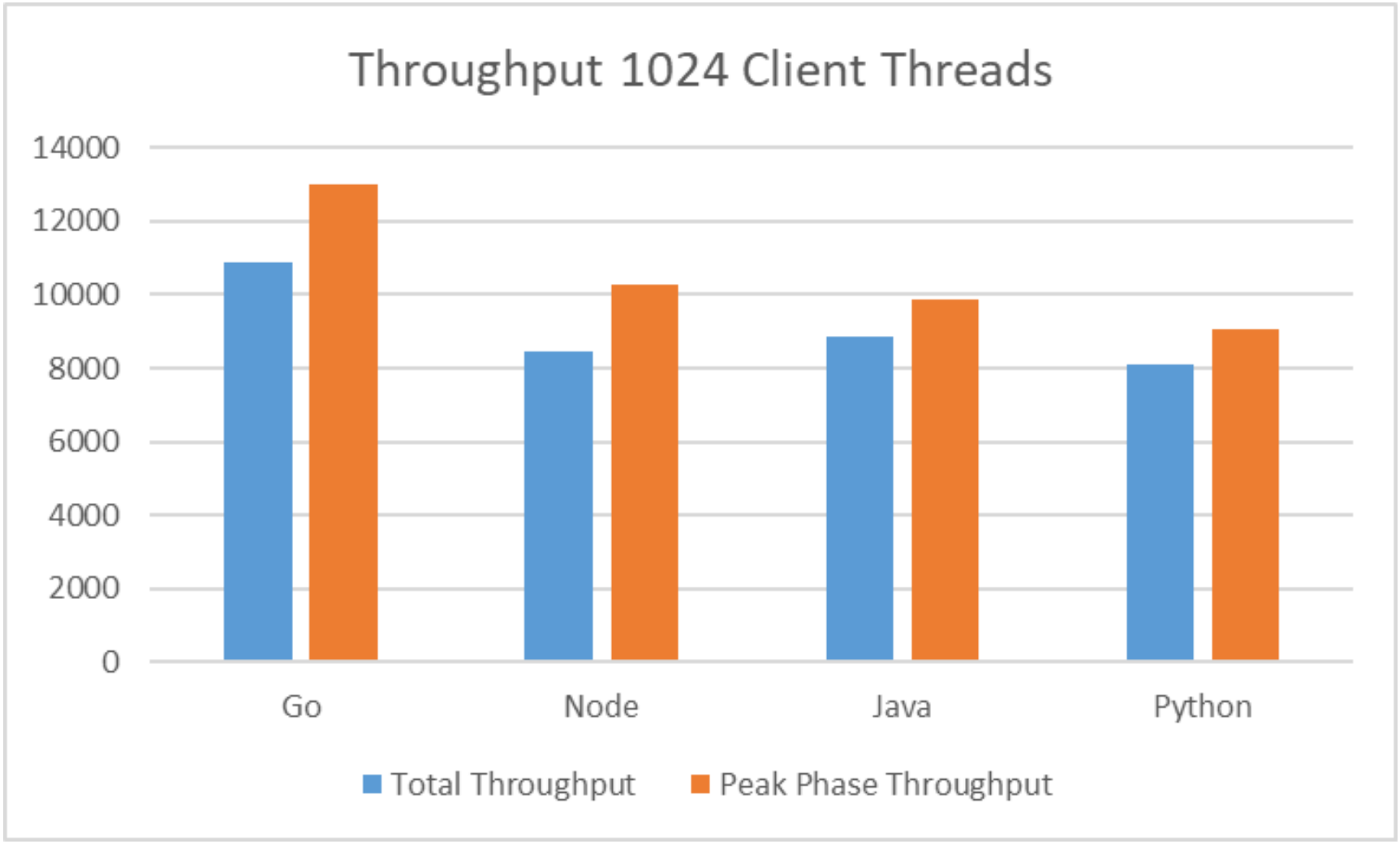




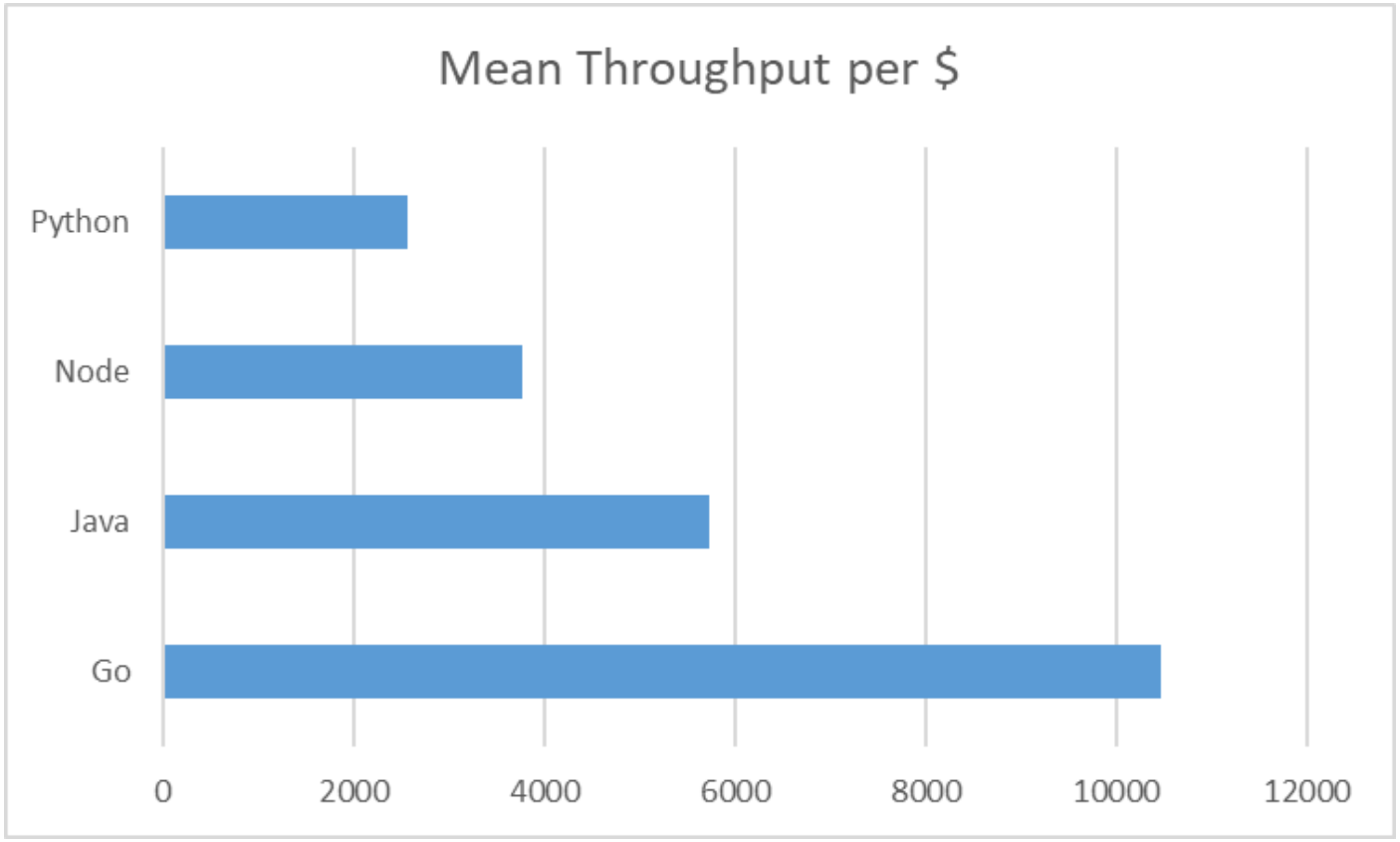




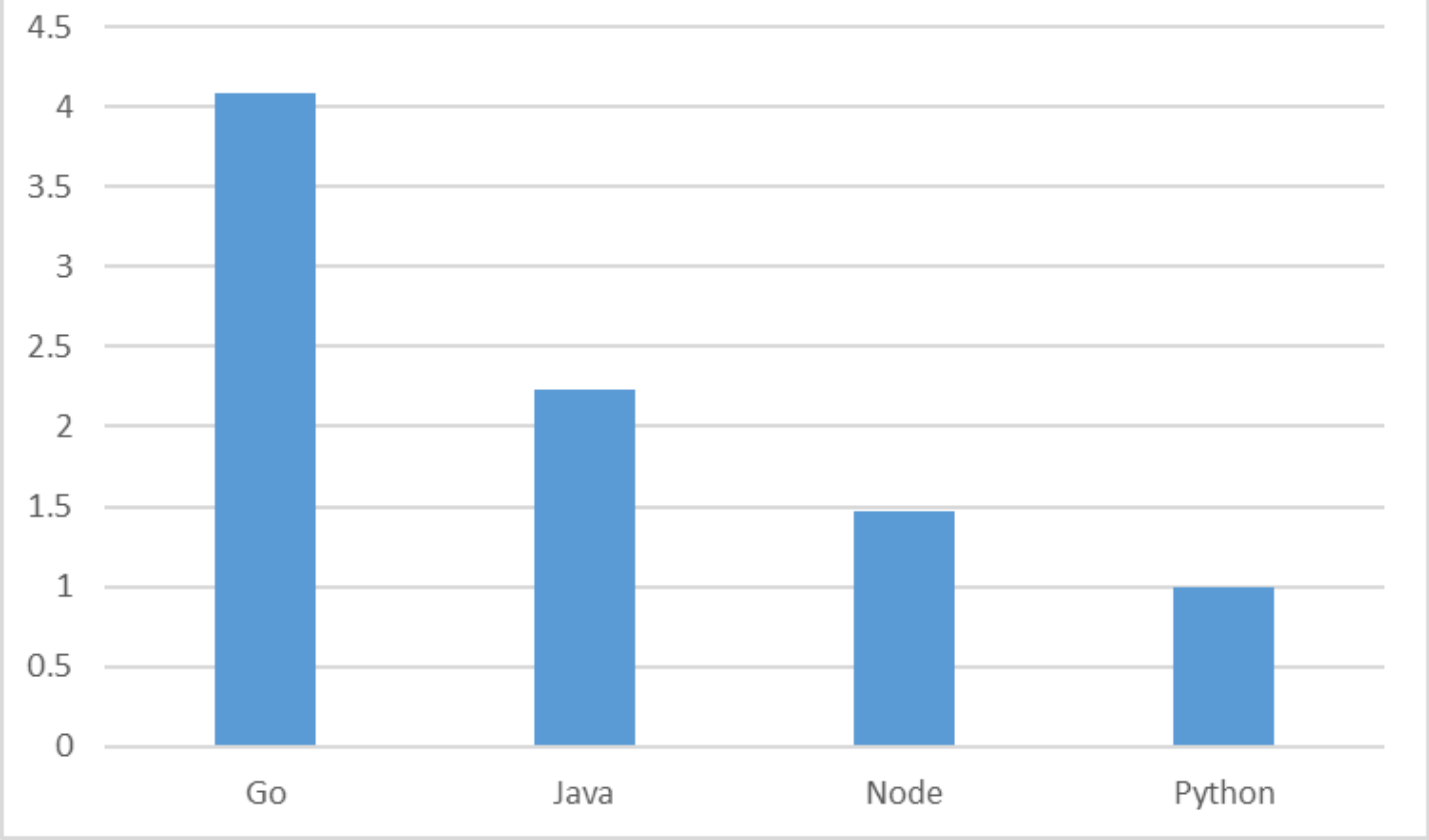
Programming language selection


Throughput 1024 Client Threads
14000
12000
10000
8000
6000
4000
2000
0
Go
Node
Java
Python
Total Throughput
Peak Phase Throughput

Language cost comparison



Price performance ratio



Programming language choice can reduce cloud costs by up to ~4x


CHOOSE WISELY.

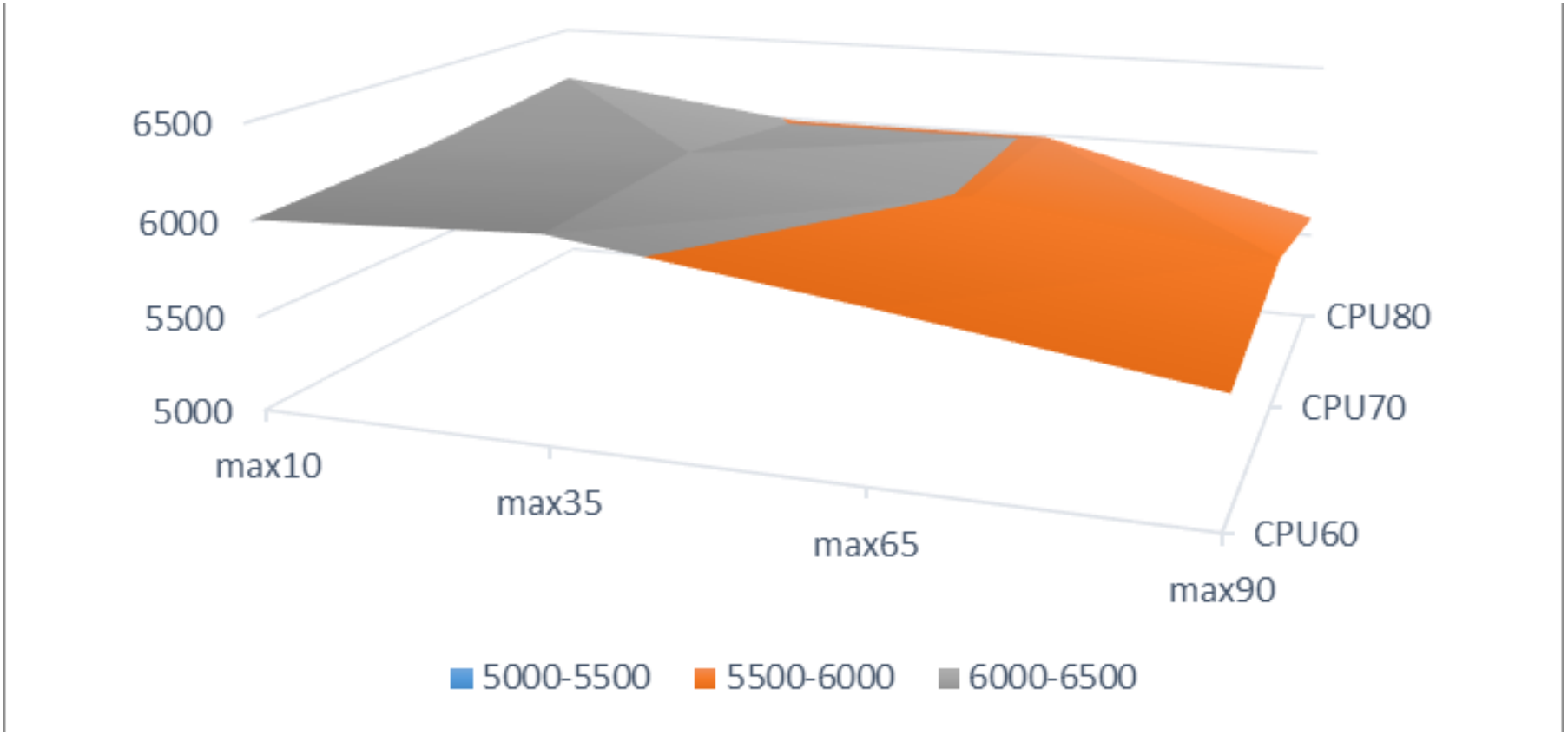
Autoscaling parameters experiment

parameter	default
target_cpu_utilization	0.6
target_throughput_utilization	0.6
max_concurrent_requests	10
max_pending_latency	30
min_pending_latency	30

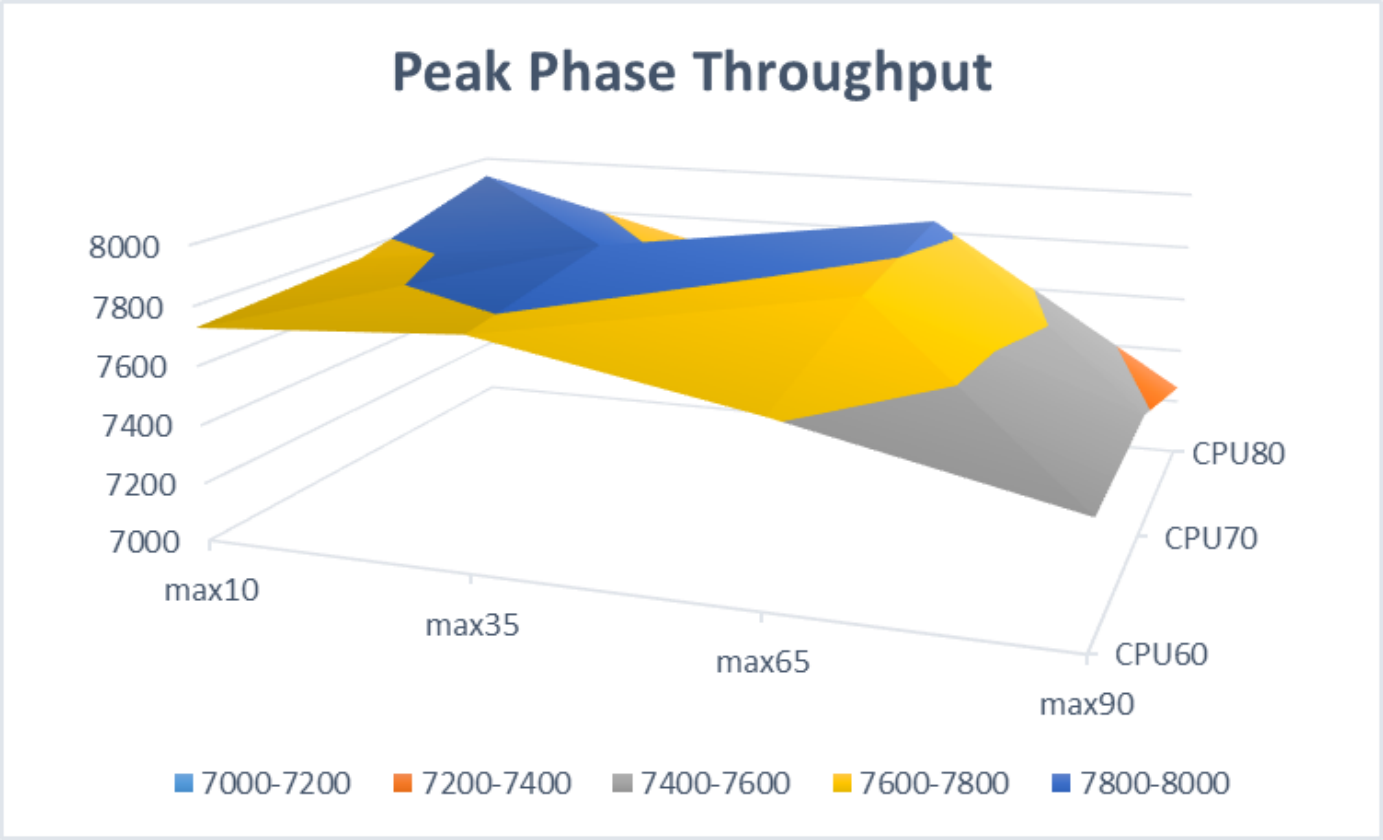
Autoscaling app Engine Parameter study

Metric?	maxc10	maxc35	maxc65	maxc90
CPU60				
CPU70				
CPU80				

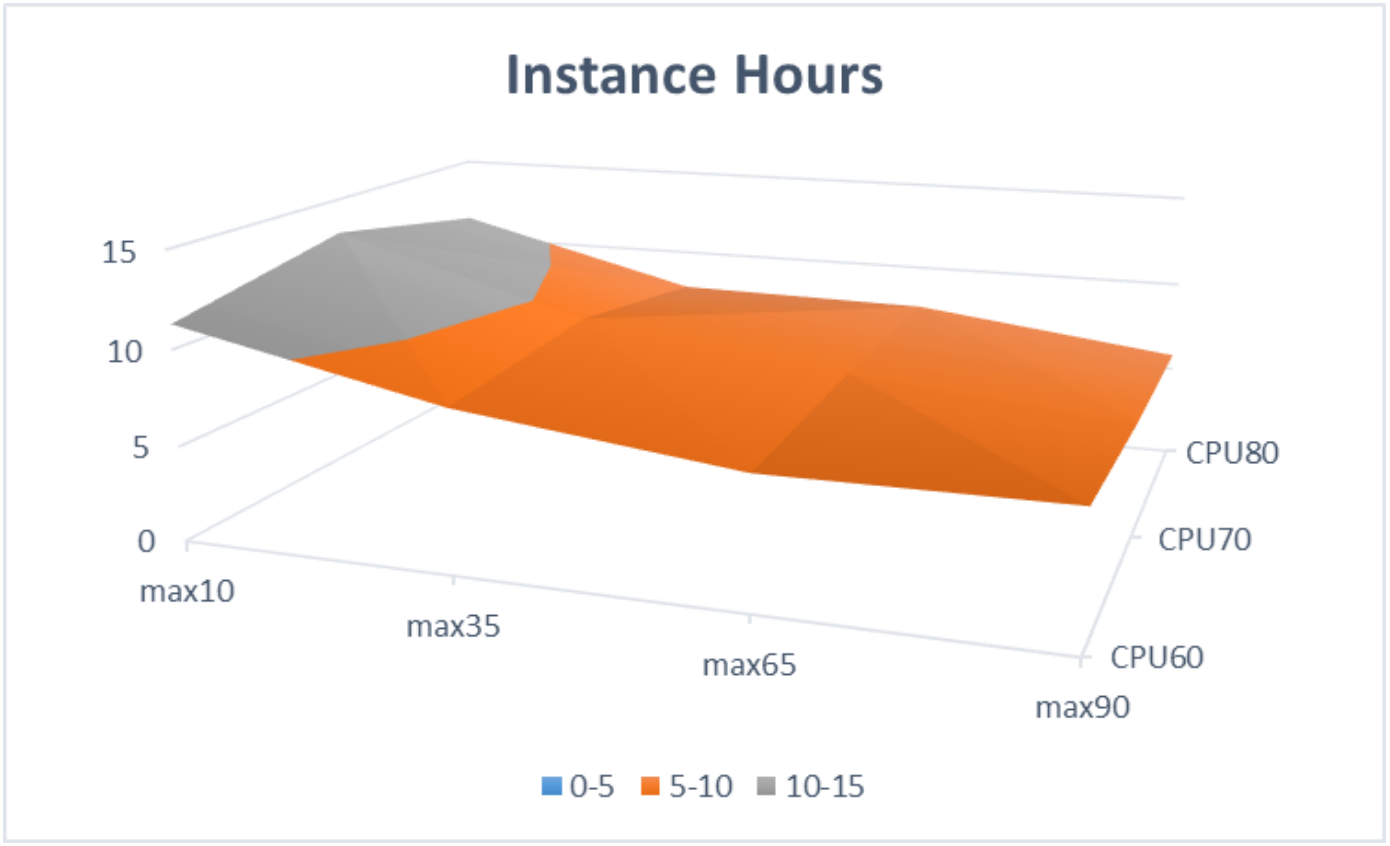
Study design with Go server



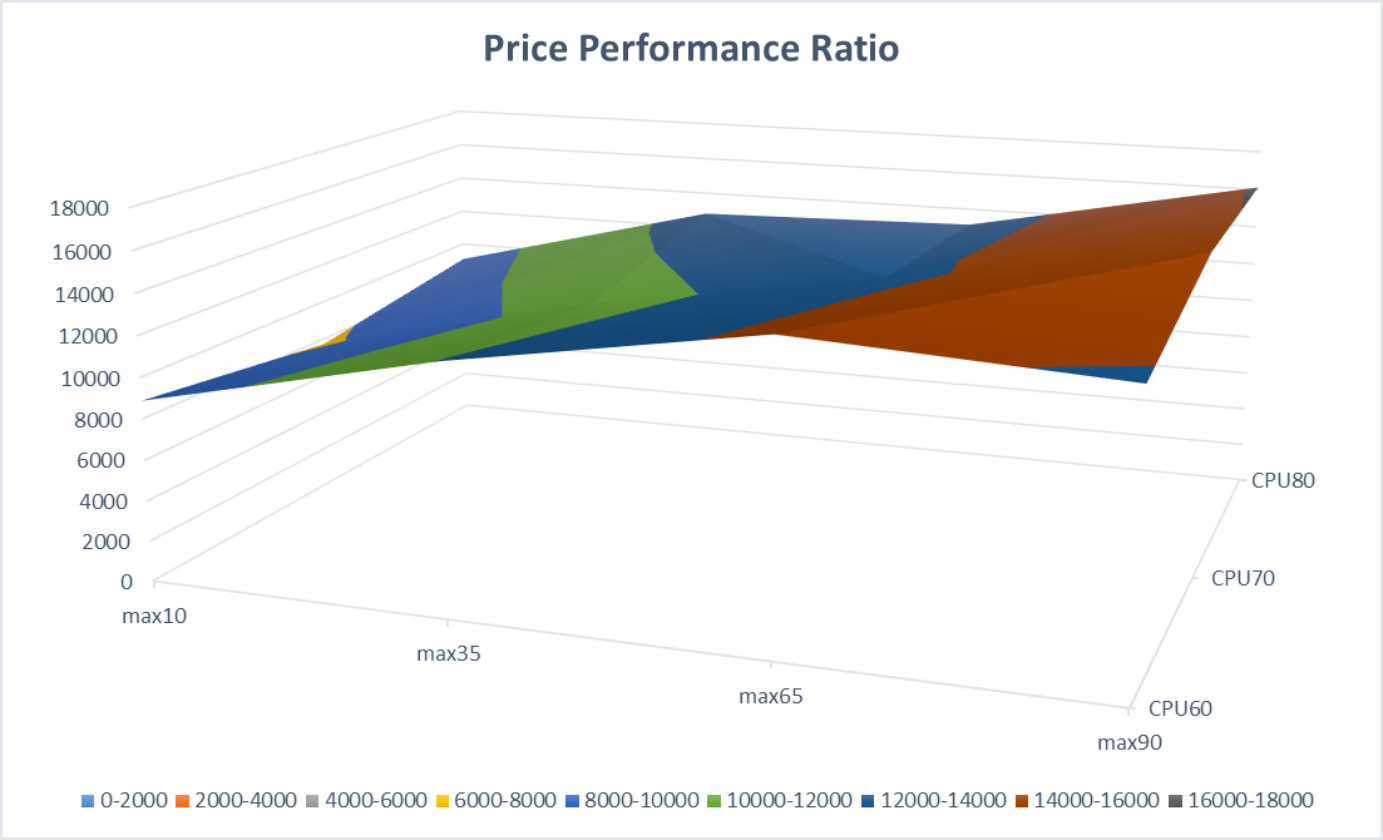
Mean Throughput



Peak Phase Throughput



Instance Hours



Price Performance Ratio

Some take aways …

throuhput mean	max10	max35	max65	max90
CPU60	6006	6067	5860	5636
CPU70	6064	6121	5993	5793
CPU80	6178	5988	5989	5605

instance hours	max10	max35	max65	max90
CPU60	11	8	7	7
CPU70	13	9	7	6
CPU80	11	8	8	6

- Higher performance possible at ~20% cheaper than default setting

Some take aways …

throuhput mean	max10	max35	max65	max90
CPU60	6006	6067	5860	5636
CPU70	6064	6121	5993	5793
CPU80	6178	5988	5989	5605

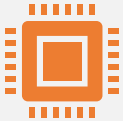
instance hours	max10	max35	max65	max90
CPU60	11	8	7	7
CPU70	13	9	7	6
CPU80	11	8	8	6

- Can achieve ~96% of default configuration performance at ~54% of default configuration costs


4GIFs.com

AWS Lambda

AWS Lambda



Servers are stateless Lambda functions

Triggered by HTTP or events on AWS services, e.g.:

- S3 bucket
- DynamoDB table

Charged based on 100ms metering intervals

Scales automatically

Lambda Example

```
// example.Hello::myHandler
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class Hello {
    public String myHandler(int myCount, Context context) {
        LambdaLogger logger = context.getLogger();
        logger.log("received : " + myCount);
        return String.valueOf(myCount);
    }
}
```

POJO Example – Count Cats

```
public class CountCatsInput {
  private String bucketName;
  private String key;

  public String getBucketName() { return bucketName; }
  public void setBucketName(String value) { bucketName = value; }

  public String getKey() { return key; }
  public void setKey(String value) { key = value; }
}

public class CountCatsOutput {

  private int count;
  public int getCount() { return count; }
  public void setCount(int value) { count = value; }
}
```

POJO Example – Define Lambda

```
import com.amazonaws.services.lambda.invoke.LambdaFunction;

public interface CatService {
  @LambdaFunction(functionName="CountCats")
  CountCatsOutput countCats(CountCatsInput input);
}
```

Example – Call Lambda Function

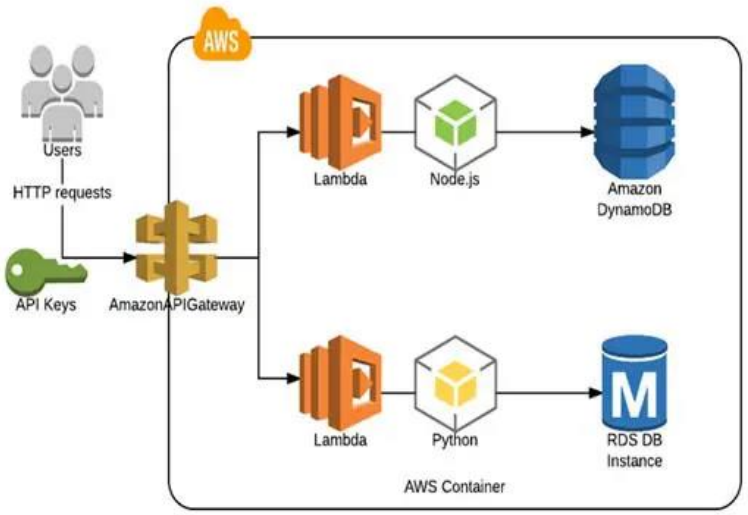
```
import com.amazonaws.services.lambda.AWSLambdaClientBuilder;
import com.amazonaws.services.lambda.invoke.LambdaInvokerFactory;

// Create proxy for Lambda function
final CatService catService = LambdaInvokerFactory.builder()
 .lambdaClient(AWSLambdaClientBuilder.defaultClient())
 .build(CatService.class);

// Finally, we invoke our service using this proxy object:
CountCatsInput input = new CountCatsInput();
input.setBucketName("pictures-of-cats");
input.setKey("three-cute-cats");

int cats = catService.countCats(input).getCount();
```

API Gateway



- Expose Lambda function as a HTTP endpoint
- Map HTTP request to Lambda function

AWS Lambda

- Multiple language support
  - No specific frameworks required
- Specify memory for functions and AWS Lambda allocates proportional CPU power, network bandwidth, and disk I/O
  - And charges ☺

Memory (MB)	Free tier seconds per month	Price per 100ms ($)
128	3,200,000	0.000000208
192	2,133,333	0.000000313
256	1,600,000	0.000000417
320	1,280,000	0.000000521
384	1,066,667	0.000000625

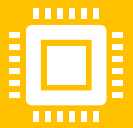
Scaling

- First request loads function and runs handler method
- Same function instance maybe resused after requests finished
- As more requests arrive, Lambda routes to available instances and creates new ones
- If a burst arrives, Lambda throttles instances
  - Between 500 and 3000 (region dependent)
- Can scale 500 instances per minute after initial burst
- Can also reserve/cap concurrency on a function
  - Provide known maximum capacity (costs)

Summary

Serverless platforms available for all major cloud providers

Aim to make deployment and management ‘admin free’



Examples GAE and AWS Lambda

Different features and scaling approaches