Using AWS Lambda to Solve Big Data Problems

Brian Smith, CSE, UNR
Max Wiegant, CSE, UNR
Vladislav Savranschi, CSE, UNR

Dr. Feng Yan, Assistant Professor
Department of Computer Science and Engineering, UNR

Abstract

The problem with existing big data solutions is that they are cater to large existing systems that have existing big data problems. In other words, the system has dedicated systems running and there is are no *pay as you go* solutions available. Many big data solutions that do exist, like Mapreduce, only cater to one language. AWS Lambda brings a new dynamic to big data solutions, allowing users to pay as they go and easily scale up and down when they need to. Lambda is a service by AWS that allows users to write event driven code that runs certain tasks in parallel separate from the system they're running from, which allows bursts of capacity to handle large tasks. Additionally, you can run Lambda in any language you feel like, and bring any library from that language along with it, allowing enormous new opportunities for working with big data.

Introduction

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of the Amazon Web Services. Using a snippet of code that needs to be run, the code is executed every time a particular event is triggered. For instance, by adding an image to the S3 bucket, a Lambda function can be made to detect an event and then run some code to convert that image into another format on the fly. Lambda only runs the specific job, then kills itself, which means if you upload 100 photos at once, 100 Lambda jobs will be generated and they will all execute in parallel then kill themselves when they have completed their task.

The purpose of this paper is to show how effective Lambda is at tackling common problems you find in big data. More and more, companies are looking at ways to implement big data early on, so it makes scaling their business easier later on. Lambda makes it easy and cheap to do this, because the user pays for what they use, and it only runs when it's executing something, so the user doesn't pay for the excess capacity. This is particularly useful for new businesses that want big data scaling early on without paying the fee that comes along with it.

Even for small menial tasks such as image conversion or data retrieval, Lambda proves to be ideal for any task from any language. It's like running a function separate from your code, with separate resources from your system, while still maintaining a native like experience. In fact, Lambda is also good for small tasks like "Internet of things" devices that only communicate every week or month, and don't need a full fledged server to be running 24/7 to accommodate them. Lambda just runs when they make a request to it, and only charges for when it runs.

This paper hopes to illustrate how AWS Lambda is a great solution to many big data problems, by giving examples and results on how it performs under certain tests.

Background or Related Work

The purpose of this project is to show several use cases and advantages of using AWS Lambda to solve big data problems. To show the capabilities of using Lambda, we will put together several examples that demonstrate how Lambda can tackle big data computing, data retrieval, and data analysis problems. Video conversion and image conversion is a great way to show how Lambda can solve big data computing problems. To show this, we could put several thousand images into an S3 bucket and then use Lambda to spin up several hundred instances to convert or modify them in parallel. To demonstrate big data retrieval and analysis, we could create a small application that grabs Airbnb data and cross references it with Zillow data to come up with the best listings. Instead of just getting information about a single area, Lambda can target many places at once and get a big data result all at once.

Benefits of using AWS Lambda:

- Runs code, then terminates
- Run code without provisioning or managing servers
- Continuous scaling
- Each instance runs using a separate IP address.
- Great for large jobs or small (Big Data or Internet of Things)
- Supports many languages (Node.js, Java, Python, C#, etc.)

Some related work has already been done to demonstrate how Lambda can be used to solve big data problems. A blogpost by Amazon gives an example of a healthcare data warehouse that can analyse large datasets in the healthcare field and generate valuable information quickly back to the users using Lambda [1].

Another example of how Lambda is being used now, is using Mapreduce and Lambda together. Lambda provides virtually infinite scaling while mapreduce provides a familiar user experience for people that have used mapreduce before [2].

Motivation/Significance/Challenge

The contributors of this paper hope that by writing this paper, the reader can understand the strengths AWS Lambda has with big data. Without running a dedicated system, Lambda significantly reduces the cost of doing big computations, retrieving large data sets, or analysing big data sets. Since Lambda is pay as you go, the user doesn't have to continuously worry about getting charged for excess capacity. Each member will be responsible for showing how Lambda can be used in different ways and demonstrating why it's significant for big data.

Brian Smith
Motivation: To test and see how AWS Lambda could handle computing large data sets.
Significance: Lambda can scale up easily to handle large tasks in parallel. Lambda can also be used to save money, if there isn't many tasks to compute.
Challenge: Getting an example to show what Lambda can accomplish, without accidentally going over budget.

Vladislav Savranschi
Motivation: Analyzing all listings in an area may be useful to learn about opportunities in the housing market. This is a practical real-life problem for many companies working with Big Data.
Significance: Airbnb provides a very large amount of data which is difficult to sort through and evaluate by hand at a large scale.
Challenge: Retrieving all Airbnb listings in a wide area. Zip codes alone may sometimes pull street numbers from random states. Using zip codes together with states' acronym appears to solve this problem.

Max Wiegant
Motivation: To test and see how AWS Lambda can handle retrieving large data sets and then use it to get useful information.
Significance: Adds a new dynamic to big data, using external data sets and then analysing them all at once.
Challenge: Getting and parsing the data in an easy way, then interpreting it in a meaningful way.

Approach (detailed design, algorithm, etc.)

High Level Design
The purpose of using AWS Lambda is to allow for numerous simultaneous instances of a single function to be parallelized on numerous machines with separate IP addresses. This makes it perfect for running image analyzing tools, numerous API calls, and various computations on large data sets without hosting a server.

AWS Lambda
AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of the Amazon Web Services. It is a compute service that runs code in response to events and automatically manages the compute resources required by that code. It was introduced in 2014.

The purpose of Lambda, as compared to AWS EC2, is to simplify building smaller, on-demand applications that are responsive to events and new information.

Node.js
Node.js is an open-source, cross-platform JavaScript runtime environment for executing JavaScript code server-side. Node.js enables JavaScript to be used for server-side scripting.

Implementation

Lambda color analyser
To demonstrate how Lambda can do big data computations, this paper implements an image color analyzer that detects the most prominent colors in an image and then outputs the colors back to the user. This simple demonstration takes a second or two to analyse a single image, but takes a long time when it computes many images at once. To show the difference, the test will compare how long it takes on a local single host machine, and then on Lamba's parallel multi host machines.

The image analyser uses a few external libraries in it's code. Most notably, is the image analyser library "node-vibrant" [3]. node-vibrant is a simple library that takes an image buffer as an input and then outputs the prominent HEX, RGB, or HSL colors. The other libraries used are "q", a javascript promise library, and "aws-sdk", amazon's sdk for AWS [4].

In order to test the difference between the single host performance and the Lambda performance, there is two separate tests that are run. The first test simply takes the images from the local computer and then analyses them on the local machine. Each test analyses a batch of images and then reports it's execution time when it's done. The Lambda test is a little more complicated because it's done on AWS. To do the lambda test, the tester must first upload all the test images to S3, amazon's simple storage service, and then invoke those Lambda functions to run, from a local program that has all the IDs for each corresponding image in the S3 bucket. The time differences are then recorded and graphed.

It should be noted, that there are limitations to this approach. One limitation is, that you can only have 200 Lambda functions running

at once, which means that some Lambda requests will be queued up and waiting until there's an opening to run. Amazon does allow you to contact them to raise this limit, but in most use cases that won't be necessary [5]. Another limitation, has to do with the speed in which you invoke the AWS Lambda functions. The process of requesting a Lambda function to run takes a few milliseconds, but it adds up when you are making a large number of requests. Lastly, depending on your implementation, there are bandwidth limitations on getting an actual result from Lamba, because all the results have to be returned to the user in some way. This cost can be mitigated by moving data to AWS S3 or DynamoDB for less overhead sending and receiving data. The image analyser mitigated these factors by moving its image data to S3.

Get Listings from Airbnb

To demonstrate how Lambda can do big data computations on a practical real-world application, this paper implements the Airbnb API HTTP Get call in a lambda function called *GetListings*, which is overloaded to be run on a local machine as well as on the AWS Lambda service[6].

GetListings reliably utilizes zip codes and states, without having a need to input City, County, or Latitude and Longitude information. Since the unofficial Airbnb API was experimentally limited to return up to 1,000 listings at a time, Zip codes provide a good way to evenly split all listings within a city without having to worry about mapping or performing radius calculations when dealing with map traversal. Zip codes are geographically smaller in dense, urban city centers, and very large in sparse, rural country terrain. Most zip codes provide between 0-100 listings, with only the most populated urban areas (such as downtown Manhattan in New York City) exceeding 1,000 listings.

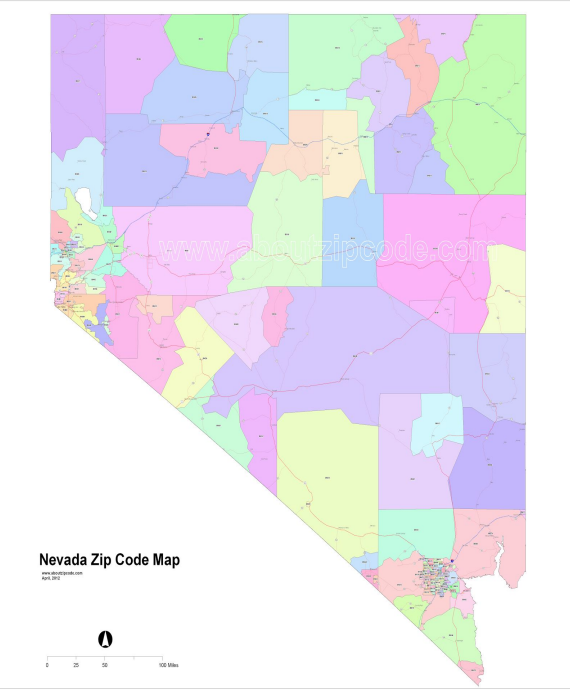

Fig. 1: *Map of Nevada zip codes. Urban city centers (Reno, Carson City, Las Vegas) are subdivided into many smaller zip codes, which allows for naturally partitioned set of listings, without having to worry about geography.*

The implementation can manually take console input for "zip" and "state" and plug in the variables into an HTTP URL for data retrieval [6]. The function is overloaded to take file input from *data.json*, which contains a list of zip codes and their respective states in a stringified JavaScript Object Notation (JSON), converted from a comma separated list of 41,712 US zipcodes and states titled *zipcodes.csv* (which can be found within the *\bin\ folder* folder within the source code).

Here is an example URL link used by the *GetListings* function implementation in its call to the unofficial Airbnb API to get all listings, just by using a zip code and state:

https://www.airbnb.com/api/v2/explore_tabs?version=1.1.0&_format=for_explore_search_web&items_per_grid=60000&fetch_filters=true&supports_for_you_v3=true&timezone_offset=-420&auto_ib=true&tab_id=home_tab&location=895

21%2C+NV%2C+United+States&allow_override%5B%5D=&key=d306zoyjsyarp7ifhu67rjxn52tv0t20&currency=USD&locale=en

The most important HTTP argument to look out for is "&location=" followed by the zip code "89521" followed by the divisor "%2C+", the state "NV", another divisor "%2C+"m, and country of origin "United+States".

To run *GetListings* in parallel within AWS Lambda, another function titled "InvokeAWS" (invoke.js) was created. It takes the group's AWS accessKeyId, secretAccessKey, as well as region ('us-east-1') to specify which AWS Lambda function to be called and from which region, using the AWS Lambda account's credentials specified by the secretAccessKey. This function must be run locally, and will pass the list of zipCodes (JSON) to the AWS Lambda Event for use in the API HTTP GET URL string.

When calling "all" argument, or invoking numerous instances of *GetListings* in AWS Lambda, each zip code and respective state is used to asynchronously retrieve Airbnb listings. These listings come back in a large block of raw JSON data.


```
{"explore_tabs":
[{"all_tab_metadata":null,"empty_state_metadata":null,"e
xperience_tab_metadata":null,"home_tab_metadata":
{"search":{"business_travel_reward_data":
{},"business_travel_ready_data":{"filter_criteria":
{"amenities_to_filter_out":[11,17],"hosting_amenities":
[76,4,40,41,35,36,44,45,46,47],"listing_types":
[1,3,43,38,4,22,37,41,40,42,2,35,47,36,11,48,49,50,51,54
,55],"room_types":["Entire
home/apt"]},"show_btr_upsell":false},"is_business_travel
_verified":false,"is_last_minute_eligible":false,"last_m
inute_show_dist_sort":false,"mobile_session_id":"0l1CCWP
P","native_currency":"USD","price_type":"nightly","price
_range_max_native":1000,"price_range_min_native":10,"sea
rch_id":"f15b8d71-360e-4f34-904b-
0949b0b5e13e"},"pagination":
{"next_offset":45,"result_count":45},"facets":
{"availability":[{"key":"Instant Book","value":"Instant
Book","count":11},{"key":"Business Travel
Ready","value":"Business Travel
```


Fig 2: *Raw JSON outputted by the Airbnb API HTTP Get call, containing a tree of listings along with their respective information.*

After the API calls succeed, Airbnb listings are printed to console including some important information such as Listing ID, Listing Name, Number of Beds, Price Per Night, and GPS location (latitude and longitude, respectively). This data can also be stored into a database, or printed out to a JSON file if necessary.

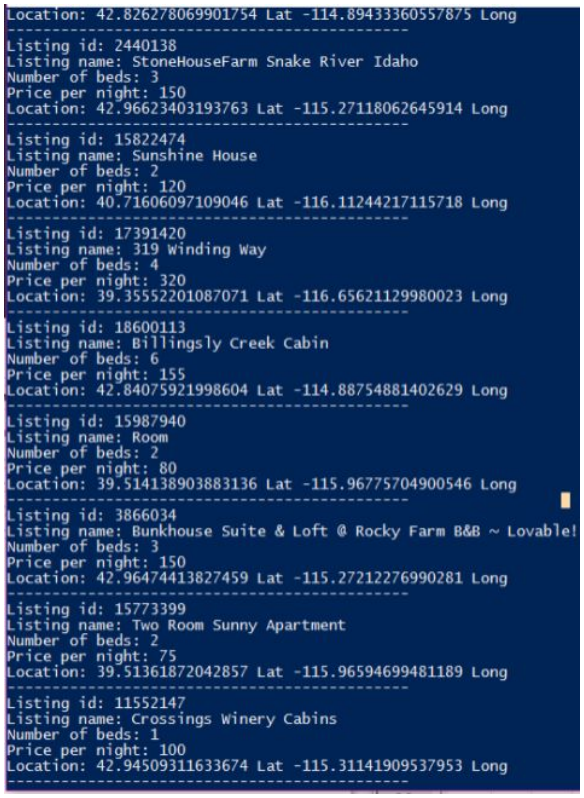

Fig 3: *Parsed JSON of Listings output from a location in Elko, Nevada at event.zip = 89801, event.state = "NV"*

Compare to Zillow and Score Listings

A second Lambda function was written, both to further test it in a Big Data project and to create a meaningful interaction between online datasets. The second function, named *ScoreListing*, looks at recently sold homes in Zillow's databases [7] in order to score individual AirBnB listings. Each individual AirBnB listing is compared against 100 to 400 nearby Zillow listings to generate a *value score*.

The *value score* is an interesting concept. It's a score computed from comparing Zillow homes in a region around some latitude and

longitude. In the application developed here, the score was used to rank AirBnB listings on which is the "cheapest" or the "most expensive". This score could be used for move novel concepts, however, including:

- Determining an ideal price to buy or sell a home at.
- Determining the ideal price to host an AirBnB listing at, given where it is located.
- Determining which part of town might be good to invest in, or perhaps might be suitable for an AirBnB rental.

Evaluation

Lambda color analyser

The results of the image analyzer were very promising, and they perfectly illustrated how Lambda scales big data computations. While the amount of computation time on the local machine increased linearly, the computation on lambda seemed to stay fairly consistent. There were twenty tests taken starting at 50 images and ending with 1000 images. Each test would increase the number of images computed by 50 new images.

Here are the results:

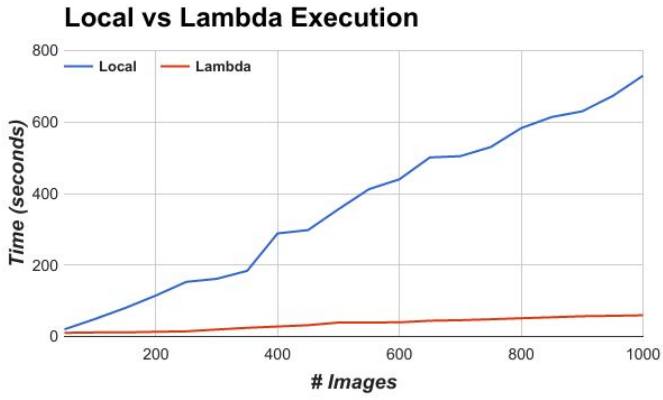

Fig 4: *This graph compares the speed of local vs Lambda image analysis time in seconds.*

Table 1: Local vs Lambda Execution

# Images	Local	Lambda
50	20.015	10.307
100	48.94	11.403
150	79.923	11.583
200	114.631	13.173
250	152.9	14.499
300	161.461	19.563
350	183.699	24.307
400	288.438	27.985
450	297.797	31.678
500	356.259	39.092
550	412.14	39.338
600	439.911	39.771
650	501.052	44.149
700	504.59	45.714
750	530.171	48.282
800	583.027	51.179
850	614.023	53.8
900	630.007	56.759
950	672.65	57.911
1000	729.743	59.464

Notice that the execution time for Lambda did increase every time, but this was likely due to local execution invoking the Lambda functions or receiving the response from AWS rather than Lambda itself. Once a Lambda function is executed it will run independently from other Lambda functions, which means they're faster because they run in parallel to each other. The data provided shows, that Lambda really shines when there are large data sets involved. When the data set was small, execution time between the local machine and Lambda, were pretty close, and if the number was smaller, it would have been faster to run on the local machine rather than running on Lambda.

The images used in this test were random 1080p wallpapers found on imgur. One thousand images were taken at random and ordered randomly for this test.

This simple example demonstrated how Lambda can take large complex problems, that would take a long time to execute on a single machine, and then execute it using Lambda but much faster. Instead of the task taking a linearly increasing amount of time, the task can be scaled to the number of Lambda functions that would be used linearly. This could theoretically reduce the time it takes to complete a task by large margins of time.

Get Listings from Airbnb

Here is a Worst-case time analysis used for measuring potential cost of running API calls for all zip-codes in the country.

Assuming 1,000 listings per zip code
Using Approx. 43000 total zip codes

Benchmark analysis: 1000 Listings in New York City, Zip Code 10003

AWS Billed time = 4.300 Seconds
Max Bandwidth = 36MB-Seconds

Amazon cost for running AWS Lambda: $0.00000625125 per 128 MB-s

Maximum possible cost for retrieving listings across the country: (0.00000625125/128) * 36 * 4.3 * 43000 = $0.33

The vast majority of US zip codes are low to medium-density in the amount of listings, and will not contain anywhere near 1000 listings.

As such, local and AWS Lambda execution times are on average much quicker. Here is an Average-case time analysis.

Local execution time: 6.356ms

AWS Lambda execution time: 0.24ms

Compiled (.csv) list of zip codes and states contains 41,712 entries.

Locally testing 41,712 Airbnb API calls is difficult to impossible, as it floods the network and quickly kills the internet connection until the application console is forcefully killed and the network router is restarted. After numerous attempts, and many self inflicted DOS attacks, the local results had to be extrapolated as accurately as possible to receive an estimate as to how much more effective AWS Lambda is at performing *GetListings* to its full extent.

Extrapolation: 41,712 * 6.356ms = 265.1215 seconds = 4.42 minutes to execute sequentially.

Asynchronously (total): From around 6017.462ms, up to around 14741.996ms per run.

AWS Lambda test: Avg. 4ms each,
Asynchronously (total): Minimum of 132ms, Average of 3658ms, Maximum of around 5000ms (cap)

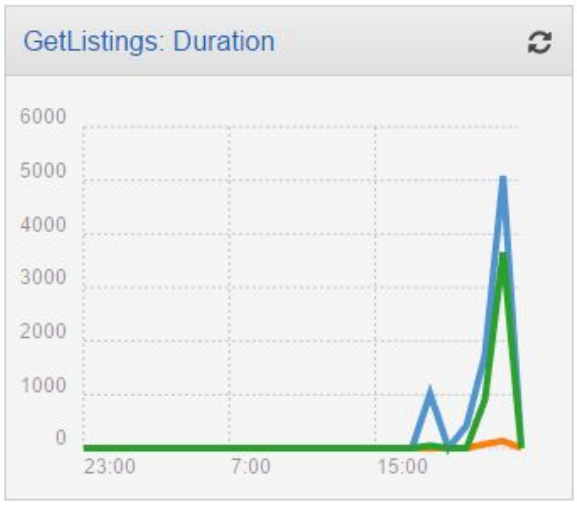

Fig 5: *This graph compares the minimum (orange), average (green), and maximum (blue) time spent running all API Get calls in parallel asynchronously (milliseconds). Retrieved from AWS Lambda Metrics dashboard.*

Compare to Zillow and Score Listings

Grabbing AirBnB listings in a region turned out to be simple. All that was needed was a zip code and the state it was in. The function was able to produce all listings in that given zip code. This task was mostly trivial.

The task of grabbing Zillow listings was somewhat non-trivial however. Latitude and longitude coordinates had to be obtained, and then translated into a different notation in order for Zillow to accept them. The general algorithm is shown below converting a longitude into the format required by Zillow:

```
input ← 139.543943
input *= 100000
input = Integer(input)
```

The program was set up by default to pull all recently-sold with a radius of 25 miles around the provided latitude and longitude. With those listings, the program was able to generate an ideal score. A limitation of the score generation was that it did not omit erroneous listings that may have been outliers. Lots that are sold outside of suburbs, for instance, may sell for many times the price of other homes in that area. Other forms of analysis that may have been more useful include comparing homes in a larger radius that match some criteria given details known about a home at the provided coordinates.

The function to provide these scores was originally planned to be one in a chain that could be called from a website. One idea proposed was to get all AirBnB listings by *GetListings* and then rank them by the score generated from *ScoreListing*. Both of these functions could have been made more selective or exposed wider ranges for their datasets. Given the constraints of time, the website was not worked on for this project. It is an expressed interest of the authors to build a website that could provide this service in the future.

Detailed contribution by each member

Brian Smith

- Designed and implemented image analyser program for AWS Lambda.
- Designed and implemented local test for the image analyser.
- Wrote the invoke example for AWS Lambda
- Tested local image analyser on local computer.
- Tested AWS Lambda image analyser on Lambda.
- Analysed results from both the local machine and from Lambda and made results table.
- Wrote the report's implementation for the image analyser.
- Wrote the report's evaluation for the image analyser.

Vladislav Savranschi

- Researched the Airbnb API and the ability to call HTTP Get on localized listings.
- Performed worst-case cost analysis of using AWS Lambda for around 43,000 (ALL) US zip codes, assuming each zip code contains 1,000 listings.
- Co-Designed and implemented local test for the Airbnb GetListings and InvokeAWS function.
- Tested and retrieved results for local GetListings API/function calls.
- Tested and retrieved results for AWS GetListings function invokes.
- Extrapolated and compared results from both the local machine and from Lambda.
- Wrote the report's implementation for the Airbnb GetListings and its associated InvokeAWS functions.
- Wrote the report's evaluation for the Airbnb GetListings and its associated InvokeAWS function.

Max Wiegant

- Researched the both of the AirBnB and Zillow websites and APIs.
- Researched NodeJS, and writing HTTP requests within it.
- Researched Lambda and NodeJS templates to be ran by Lambda.
- Designed and implemented both the Airbnb *GetListings and ScoreListings* functions.
- Tested and retrieved results for local *GetListings* API/function calls.
- Tested and retrieved results for AWS *ScoreListings* function invokes.
- Hooked up the *GetListings* and *ScoreListings* functions in Lambda.
- Debugged all steps along the way

Discussion/shortcomings

Some shortcomings of AWS Lambda, is that the location of your code is separated into two different areas rather than one place in your code. Lambda requires you to upload your code in a zip file to the website when you want to make a lambda function. It's difficult to manage your code and debug this way because it's hard to tell what's going wrong when your code isn't all in the same place.

Another problem with AWS Lambda, is that it's very difficult to debug things going on within the Lambda function, because you can't console log or do any kind of error checking. In order to do error checking, you have to send that data back through the callback, which becomes very cumbersome because it stops execution of the program when you do the callback.

AWS and AWS Lambda dashboard can be difficult to navigate, as there is a lot of information scattered throughout the UI. It takes time getting accustomed to it, which may impede project progress. When it comes to billing, Lambda is also very difficult to figure out, because their billing scheme consists of both time of execution and memory used. It's hard to determine what you're going to pay until after you've executed the program. With EC2 instances, information is very clear about what you are being charged, and very easy to calculate.

The idea to create a website for calling AWS Lambda was scrapped due to security concerns, together with the fact that the lambda functions can easily be accessed and multiple instances parallelized using the local *InvokeAWS* function. The issue is that the accessKey and secretAccessKey would have to be put up on the net, which can be abused by anyone with access to the website - as any AWS Lambda calls directly bill the account,

and the credit card of the STS200 group. These costs cannot be refunded by Amazon, and so, any further testing of these functions would have to be done using one's personal account.

Conclusion and Future Work

With the tests provided in this paper, Lambda has proven itself as an excellent solution to big data problems. Lambda was able to scale and meet the demands of all tests, and proved itself a worthy contender to mapreduce and other existing frameworks. By running tasks in parallel, Lambda acts as a separate machine, running with its own resources acting as a scalable extension of your environment. Since its a pay as you go service, Lambda won't break the bank for short term or long term tasks, and the execution time last just long enough to complete the task.

As more and more companies look for solutions to big data problems early on, before they really need it, Lambda fits the demand. No matter what size the job is, Lambda can execute, analyse, and report at a low and efficient price. More and more people are starting to see the benefits of Lambda and it will be necessary tool in the way people handle big data in the future.

References

[1] "Tag: AWS Lambda." AWS Big Data Blog AWS Lambda Tag. N.p., 12 May 2017. Web. 16 May 2017. https://aws.amazon.com/blogs/big-data/tag/aws-lambda

[2] "Serverless Map/Reduce." To The Stars. N.p., 03 Nov. 1970. Web. 16 May 2017. http://tothestars.io/blog/2016/11/2/serverless-mapreduce

[3] Akfish. "Akfish/node-vibrant." *GitHub*. N.p., 10 Mar. 2017. Web. 16 May 2017. https://github.com/akfish/node-vibrant

[4] Aws. "Aws/aws-sdk-js." *GitHub*. N.p., 15 May 2017. Web. 16 May 2017. https://github.com/aws/aws-sdk-js

[5] "AWS Lambda Limits." *AWS Lambda Limits - AWS Lambda*. N.p., n.d. Web. 16 May 2017. http://docs.aws.amazon.com/lambda/latest/dg/limits.html

[6] N. Rahnemoon (2016) The Official Unofficial Airbnb API Documentation, (Website). Retrieved from:

http://airbnbapi.org/

[7] Zillow, Real Estate and Mortgage Data for Your Site, (Website). Retrieved from:

https://www.zillow.com/howto/api/APIOverview.htm