Implementation of Unsupervised k-Means Clustering Algorithm within Amazon Web Services Lambda

Anthony S. Deese, Ph.D.
Department of Electrical and Computer Engineering
The College of New Jersey (TCNJ)
Ewing, NJ USA
deesea@tcnj.edu

***Abstract*—This work demonstrates how an unsupervised learning algorithm based on k-Means Clustering with Kaufman Initialization may be implemented effectively as an Amazon Web Services Lambda Function, within their serverless cloud computing service. It emphasizes the need to employ a lean and modular design philosophy, transfer data efficiently between Lambda and DynamoDB, as well as employ Lambda Functions within mobile applications seamlessly and with negligible latency. This work presents a novel application of serverless cloud computing and provides specific examples that will allow readers to develop similar algorithms. The author provides compares the computation speed and cost of machine learning implementations on traditional PC and mobile hardware (running locally) as well as implementations that employ Lambda.**

Keywords—machine learning, k-Means Clustering, Kaufman, serverless computing, Amazon Web Services Lambda, cloud computing

I. Introduction

A new breed of cloud services referred to as serverless cloud computing (SC) provides a new set of tools for developers of software that run on low-power mobile devices, rely on cloud-stored information, and implement advanced techniques like artificial neural networks. Note that SC is also referred to as function as a service (FaaS), which some consider a more appropriate description its purpose. When serverless cloud computing (SC) functions (also known as lambda functions) are integrated within a capable cloud infrastructure, they are provided with efficient access to services associated with data storage and migration, authentication and secure access, as well as system monitoring. If the network is efficient, entire applications may be designed as a collection of these interrelated SC lambda functions with little consideration of hardware limitations, library dependencies, and server boundaries. There are some works that already address machine learning and its implementation within the cloud [1, 2, 3, 4, 5, 6]. The author's objective is to build upon these works and take their results to the next level.

The evolution of cloud services and their increasing role in everyday life motivated the author to study their application to his research in the field of machine learning and artificial neural networks. SC has become a major force in the development of new commercial products and academic research projects. The first commercial service (AWS Lambda) was only introduced by Amazon in late 2013, as a supplement to their existing Elastic Cloud Computing (EC2) service. Over the last five years, many competitors have emerged including IBM Apache OpenWhisk, Microsoft Azure, and Google Cloud Functions. SC represents a rapidly growing component of the approximately $30 Billion cloud computing industry [2]. Because computing power is provisioned on demand by a SC service provider, costs may be lower than those associated with more traditional services (e.g. EC2). SC will play a major role in the future of cloud computing as well as computer technology in general [2]. It should be noted that SC has potential to play a major role in technologies like the Internet of Things (IoT) that rely on edge computing.

II. Problem Statement

In this paper, the author addresses several concrete problems related to machine learning and cloud computing such as the need(s):

- *For Machine Learning to Analyze and Process Big Data* – Reference [4] discusses how and why machine learning is essential for analyzing and processing big-data. Basic analytical methods used in traditional business intelligence reduces reporting to sums, counts, and simple averages. These types of online analytical processing are merely a systematic implementation of basic operations that rely on human interaction to direct activities and specify desired output. Alternatively, machine learning is data-driven and highly-scalable, making it well suited to deal with the disparate data sources and large variety of variables and amounts of data involved. Machine learning is essential for analyzing and processing big-data because its performance improves as more training data is supplied to it [4].
- *For Cloud Computing to Perform Such Complex Tasks* – Reference [12] discusses how traditional frameworks have struggled to provide adequate computing power for ma-chine learning as employed by many common data-intensive tasks, especially when most users rely on low-power mobile devices. There is a need to employ cloud computing as a potentially more effective alternative [12].
- *To Implement Machine Learning via SC* – Reference [9] discusses how difficult it can be for users to effectively estimate as well as efficiently request and provision cloud computing resources for their machine learning task

when the training datasets and algorithms employed may vary radically from case to case. There is a need to implement machine learning via SC / FaaS because it allows service providers to optimally allocate and release resources as needed, lifting this burden from the user. The use of SC as the platform for machine learning is advantageous for scalability, cost efficiency, and lifetime maintenance effort [9].

- *For Better Implementations of Machine Learning for SC* – Reference [10] discusses the difficulties of translating traditional web applications for upload to and execution by SC services like AWS Lambda. There are many limitations placed on lambda functions regarding their structure, size, and types of methods and libraries that may be employed. This paper illustrates that developing a machine learning algorithm to AWS Lambda is by no means a trivial task, one that has significant value to the research as well as industry com-munities. Paper [11] focuses on one such limitation, highlighting the need to minimize library dependencies and develop lean lambda functions may be quickly instantiated and killed by an SC server to minimize costs.

III. Solutions and Research Objectives

The author of this work presents a novel approach to machine learning for efficient big data processing within a system dominated by mobile devices. The proposed solution achieves this by utilizes custom, lean, and modular lambda functions to perform machine learning. Unlike many previous works, it does not investigate how existing web applications and libraries like Machine Learning in Hadoop may be translated for use within a serverless cloud computing infrastructure, an interesting but ultimately less efficient approach. The authors proposed solution:

- Employs a lean design philosophy such that serverless cloud computing resources are used efficiently.
- Interfaces lambda functions directly with cloud storage to efficiently access large amounts of data.
- Emphasizes code modularity and reliance on micro-services to maximize scalability and reusability.
- Interacts with mobile device applications seamlessly as gateways for user interaction.

This approach is supported by multiple previous works, most strongly by [8].

IV. Technical Background

A. Serverless Cloud Computing

Serverless cloud computing refers to a model in which the service provider handles many tasks to alleviate certain burdens from the software developer(s). This provider is expected to automatically divide and provide resources such that clients are billed for the resources that they use, not pre-reserved units of capacity. SC is one form of utility [13, 14, 3, 15, 16, 11, 17]. Although serverless computing still requires physical servers, the name is used for several reasons. Virtual machine instances are anonymous and generalized. Server management and capacity planning are handled completely by the cloud service provider. Computing power is provisioned on demand as well as deprovisioned when no longer needed. Clients are billed by execution and resource consumption, not by an hourly or hardware-base rate.

There are many benefits of serverless computing as compared to traditional server models:

- Utilizes a basic operating system with a generalized config.
- Client can code in multiple languages, from Python to Java.
- Client can deploy/automate more quickly and effectively.
- Client does not need to manage or own any hardware.
- Provider can quickly create/destroy virtual machines.

Costs can be significantly lower for those clients that employ server resources sporadically.

It goes without saying that, like any new technology, serverless computing does present disadvantages as compared to more traditional computing methods. Examples include laggy startup and the potential for lags and drops in communication. It is important for any researcher to acknowledge these disadvantages and address them in her/his work.

B. Commercial Machine Learning Services

One should note that Amazon Web Services does provide users with its own Machine Learning (ML) Service. However, it is not suited for this research in several ways:

- *For Supervised Learning Only* – AWS ML supports supervised learning only, and therefore cannot be used to solve clustering problems.
- *It's Not Optimized for App Integration* – AWS ML is not intended to be a service that is called by mobile applications, especially when it comes to the learning process itself. The service requires that users upload the training data as an Excel spreadsheet to an AWS server, while most mobile applications would store the data to the cloud via DynamoDB.
- *Can Be Expensive, Especially for Real-Time Prediction* – AWS ML is expensive, especially when used to perform real-time prediction over batch-types. AWS charges $1.00 per 10,000 real-time predictions; this cost can get out of hand relatively quickly.

This service may be a viable option for the author's future research, but is not employed here.

C. Special Considerations for Serverless

Cloud computing is a powerful tool for software developers, especially when considering its effect on project scalability. Cloud computing allows applications to draw resources from hundreds of machines to serve a larger number of users. SC presents an interesting variation on this idea, improving the potential elasticity of these applications and their ability to respond efficiently to a user base that grows and shrinks rapidly. Because the service provider allocates computing resources on demand, the applications they serve can quickly respond to changes in data input size and user traffic [10]. This elasticity is most advantageous, regarding cost, when applications use resources sporadically. The author believes that many machine learning-based applications fit this profile, with training data being collected over long periods of time and being analyzed periodically for relatively short periods [10].

For this type of machine learning algorithm, the ability of a cloud service provider to quickly provision a new virtual machine and execute code is essential [10]. This speed is often hindered by the need to perform lengthy software installations and load resources from dependent libraries. This overhead may drastically increase resource usage (and cost) by requiring the import of library packages over the network, decompression of these packages via CPU, and writing the decompressed files to disk [10]. A *lean application* is one that employs efficient code within the lambda function itself as well as efficient and minimal reliance on large external libraries. This is one reason the authors of this work opted to employ custom and efficient machine learning algorithms embedded within the lambda function source code itself [10].

V. Software Development

A. Development Environment

The author developed lambda functions within the Eclipse Integrated Development Environment (IDE) using Java. AWS allows development of lambda functions in multiple other languages. The author chose this environment because AWS recently released a toolkit for Eclipse. This toolkit provides templates for Lambda projects, uploads code to AWS servers, and facilitates access to AWS cloud storage services.

For this work, the author employed many services within the AWS suite. User access is handled by the AWS Cognito and Identity and Access Management (IAM) services. Note that IAM allows a user to access and monitor only those lambda function instances that she or he created. This maintains the privacy of user training data and machine learning results. Within IAM, the author configured this user pool to include policies (or permissions) that allow members to access and create new instances of the lambda function and obtain results, but not make changes to the algorithm itself.

The author designed his lambda function to accept basic AWS session credentials composed of an access key ID, secret access key, and session token. The user must regenerate these credentials every few hours (on average). This type of credentials prevents a user from gaining permanent access to the cloud computing resources in AWS Lambda.

B. Training Data Import and Usage

The author employed training data that he acquired from his previous work in wireless sensor development. It contains measured values for temperature, background light and motion, as well as current drawn by electric loads within five rooms of an approximately $2000ft^2$ home. Specifically, the training data contains more than $5{,}000$ samples collected over a period of three months; each sample has 25 fields or values.

Initially, the author's training data was stored within his PC as a comma separated value (CSV) file. This work, obviously, required that the data be imported to the AWS DynamoDB cloud data storage service. This task was completed in five steps. First, an appropriate policy must be attached to the target pool within AWS IAM, giving proper users the ability to transfer data via pipeline. The pipeline is a web service that automates movement of information between AWS services. There is no need to create a custom policy, as a suitable option is available under the AWS Service Role tab within IAM. The policy is named Amazon EC2 Role for Data Pipeline. The second step is to convert the CSV file to a JavaScript Object Notation (JSON) file. Each row is defined by its own JSON object with the keys (Id, dataType01, dataType02…) and data types (n = number, s = string) for each value clearly defined. This conversion may be easily completed within Excel.

The third step in the import process is to create a new table within DynamoDB. Here it is important to ensure that the primary key or partition key (as defined during the creation of a new table) matches the name and type of at least one training attribute. For example, a JSON object may employ a primary key with definition "PrimaryId" and type of "number." If the appropriate keys do not match, then Lambda will be unable to access elements from the table.

The final two steps in the process are to upload the training data to AWS Simple Storage Service (S3) and execute the AWS Pipeline. Note that JSON file uploaded must be accompanied by a manifest file named manifest.txt stored within the same folder. This manifest file defines the how the data may be accessed and URL within S3 of the target data. Once the data is ready, the author created and executed the AWS Data Pipeline. This process is relatively straightforward because this service provides a template under "DynamoDB Templates" named "Import DynamoDB Backup Data from S3". The developer is still required, however, to define the location of the training data packet, target table in DynamoDB table, and any logs to be stored. Note that next to "Input S3 Folder" that the developer should include the name of the folder that contains both the JSON file and manifest.txt, not the location of the JSON file itself. The manifest instructs the pipeline which source data to employ.

C. Lambda Function Structure

As discussed above, one essential aspect of this research is the development of lambda functions that are lean, or use a minimal number of external library dependencies. The author's lambda function includes ten library dependencies, nine of which are essential to interact with the AWS Cognito, IAM, DynamoDB, and Lambda services. One is the general java.io.IOException dependency. The machine learning lambda function presented in this paper does not use any machine learning-related external libraries. The total package size is $4.9MB$, as reported by the AWS online console.

Instructions are provided to AWS Lambda as a class composed of a main handler function along with any supporting methods or object types. The term lambda function is broad and often used to describe this class or the handler function itself. The handler function is starts the desired execution and may be passed two inputs, for AWS Lambda implementations at least. One input is the application context, which describes how the lambda function interacts with AWS and defines values like maximum execution time. The second input is defined by the user; it may be assigned type JSON, allowing multiple values to be passed to the handler function as part of a single input object. For this research, the author passed multiple inputs to AWS Lambda as a concatenated string that may be parsed by the handler function into its components. He chose this approach after encountering problems with AndroidOS and how it passed objects to AWS Lambda.

The author employed an application structure where session credentials (composed of access key ID, secret access key, and session token) are passed to the handler function as strings. These credentials are used within the handler function to create an *AmazonDynamoDBClient* and access the DynamoDB using methods from the AWS Java SDK.

VI. Clustering For Unsupervised Machine Learning

The author employs an unsupervised machine learning algorithm for data clustering based on the k-Means Classifier (*kMC*) method [20]. This algorithm divides the training data set containing N_S samples into N_{Cat} clusters. Each sample is assumed to be an array containing N_L element values. The algorithm may be broken down into several steps:

- *Initialization* – The first step in the clustering process is to generate N_{Cat} arrays (each should contain N_L elements) to define the initial averages for all clusters. This work employs the Kaufman Initialization method and is further discussed below [21,22]. For the calculations below, the initial average for cluster γ is represented by $z_{Avg}^{(0)}[\gamma]$. Element i of this array is represented by $z_{Avg}^{(0)}[\gamma, i]$.
- *New Iteration* (m) – Start a new iteration of the kMC Method by defining the target m, beginning at $m = 0$. The sample index j is calculated from iteration m using the modulo operator ($j = m \textbf{ mod } N_S$).
- *Calculate Distances* – Quantify the distance (aka. Euclidean Distance) between sample j and each of the average value arrays for the current iteration m as shown in (1). In this work, the Euclidean Distance between sample j and $z_{Avg}[\gamma]$ is be represented by $d[\gamma, j]$.
- *Find Minimum Distance* – Use a search to determine the cluster index corresponding to the minimum value of $d[\gamma, j]$ for sample j and iteration m as shown in (2). In this work, this optimal value of γ is represented by γ_{Min}.
- *Reassign Sample to Best Cluster* – Once the best cluster is known, reassign sample j to cluster $C_{\gamma Min}$ such that $j \in C_{\gamma Min}$. The *kMC* Method assumes that the closest cluster, as defined by the spanning Euclidean Distance, is best and that to which sample j should be assigned.
- *Update Cluster Averages* – Once the cluster assignment for sample j is redefined, the averages for all clusters must be updated as shown in (3). In this work, collection of all samples assigned to cluster γ for iteration m is represented by $C_\gamma^{(m)}$. The number of sample assigned to this cluster is represented by $\boldsymbol{n}(C)$.
- *Increment to Next Iteration* $(m+1)$ – The algorithm next increments iteration $m = m + 1$ and updates sample j.
- *Check for Completion* – These iterations stop upon convergence, once there is no change in the members of any clusters between two cycles of the training data.
- *Begin New Iteration (Again Potentially)* – If convergence is not achieved, the algorithm will return to bullet #2 above for the incremented iteration / sample and updated definitions of $z_{Avg}^{(m+1)}[\gamma]$ and $C_\gamma^{(m+1)}$ for all clusters (γ).

This process is shown mathematically in (1) through (3).

$$\textit{for iteration } m\text{: } \underline{\underline{d}}[\gamma, j] = \sqrt[2]{\sum_{i=1}^{N_L} \left(\underline{\underline{z}}[j,i] - \underline{\underline{z}}_{Avg}^{(m)}[\gamma, i] \right)^2} \quad (1)$$

$$\textit{for sample } j\text{: } \underline{\underline{d}}[\gamma_{Min}, j] = \min_{for\ all\ \gamma} \left\{ \underline{\underline{d}}[\gamma, j] \right\} \quad (2)$$

$$\textit{for all clusters}\text{: } \underline{\underline{z}}_{Avg}^{(m+1)}[\gamma, i] = \frac{1}{\boldsymbol{n}\left(C_\gamma^{(m)}\right)} \sum_{\substack{j=1 \\ j \in C_\gamma}}^{N_S} \underline{\underline{z}}[i, j] \quad (3)$$

For initialization, the author employs the Kaufman Method to set the initial values of $z_{Avg}^{(0)}[\gamma]$ for all clusters [21,22]. The objective of this initialization method is to define a set of initial averages spread all throughout the solution space and emphasize the potential for differentiation between these clusters. One intuitive alternative method is to simply assign the initial cluster averages randomly. However, if they are very close to one another, it may be difficult for the kMC method to differentiate between clusters properly. In short, the Kaufman Initialization Method cycles through the training data and sets initial cluster averages iteratively based on the distance between samples an all existing initial cluster averages [21,22].

VII. AndroidOS Mobile Application

Another objective of this research is to develop a mobile application to call the AWS Lambda functions. The author did develop a simple AndroidOS mobile application for this purpose. It allows users to select the training data set, machine learning algorithm, and minimum number of iterations to be used. It also provides basic information to users regarding results. Full results are stored to and must be viewed within the DynamoDB database. The mobile application performs several basic tasks to fulfill the objective above. It creates a *Cognito-CachingCredentialsProvider* to acquire credentials from Amazon Web Services. It then creates a *Lambda-ProxyInterface*. This interface allows the mobile app to interface with AWS Lambda and defines the structure of the input to be passed to it. Next, the application creates a *Lambda-InvokerFactory*. This object builds the aforementioned *LambdaProxyInterface* and passes relevant input to AWS Lambda. It is essential that any interaction between the mobile application and AWS Lambda be packaged within a separate thread or asynchronous task. This prevents communications delays or dropped responses from freezing the main thread of the mobile application.

VIII. Testing and Analysis

A. Test Parameters

The author compares results from three different computing platforms: 1) AWS Lambda running within the Cloud, 2) AndroidOS running locally on a mobile device, and 3) Mathworks' Matlab running on a PC. The mobile device is a fourth-generation Motorola Moto G with an 8-core Snapdragon 617 processor running at 1.5GHz and 2GB RAM. The PC is a Dell XPS 15 9560 with an Intel i7-7700HQ processor running at 2.2GHz and 16GB RAM.

In this paper, the author does address the use of batch-mode read and write requests within AWS and their effect on computation time as compared to individual requests. Individual write operations are performed by: 1) instantiating a *Table*-type object to be accessed, 2) creating an *Item*-type object to be written, 3) saving all attributes to this *Item* one-by-one, as well as 4) executing a *PutItemRequest.* Individual reads are performed by: 1) instantiating a *Table*-type object to be accessed, 2) executing an individual *GetItemRequest* to retrieve the target object whose ID matches the desired primary key, as well as 3) extracting all attributes from that *Item* and transferring them to a compatible Java object. Batch mode reads are slightly more complicated. They are performed by: 1) creating a *Table*-type object to be accessed, 2) creating a *TableKeysAndAttributes*-type object to define the group of individual requests, 3) adding the desired primary keys to this object, 4) executing a *BatchGetItem* that employs *TableKeysAndAttributes* as input as well as yields a *BatchGetItemOutcome*-type output, as well as 5) extracting individual *Items* and embedded attributes from this outcome using iterative loops. The maximum size of a *BatchGetItem* request is 100 items. Batch mode write operations are performed by: 1) creating a *Table*-type object to be accessed, 2) creating a *BatchWriteItemRequest*-type object to define the group of individual requests, 3) creating a list of individual *PutRequets* to define the target rows, 4) executing a *BatchWriteItem* that employs the *BatchWriteItemRequest* as input as well as yields a *BatchWriteItemResult*-type output, and 5) checking this object for any unprocessed items or errors. The maximum size of a *BatchWriteItemRequest* is 25 items. Unless otherwise stated, AWS Lambda is assumed to perform all reads and writes to DynamoDB in batch mode.

The author examines the performance of an unsupervised kMC-based machine learning algorithm used to assign each sample within the training data set to one of five clusters, corresponding to the five rooms for which measurements were collected. Convergence is achieved if more than 85% of these assignments remain constant between two consecutive cycles through the training data set. The algorithm is required, by default, to process each training data sample at least twice. Note that Section V.B of this work provides further details on the author-generated training data. This section of the paper will present the following results:

- Quality of the unsupervised learning algorithms applied to both author-generated training data and control data set provided by Chen et. al [24].
- Computation time vs. computation method (Mathworks' Matlab, AndroidOS locally, and AWS Lambda) for unsupervised learning algorithm applied to author-generated training data set.
- Computation time vs. read and write types (batch vs. individual) for unsupervised learning algorithm applied to the author-generated training data set.

Mathworks' Matlab provides the control for all test results.

B. Training Results and Quality

Because all three of the computing methods (Mathworks' Matlab, AndroidOS, and AWS Lambda) use the same training data sets and implementations of machine learning, they generate identical results. For the author's training data set, the kMC method assigns approximately 68.5% of samples to the ideal or correct cluster, with 31.5% assigned to a cluster that does not match their location within the home.

To provide another reference, the author also applied his machine learning algorithms to a training data set provided by the University of California at Irvine (UCI) Machine Learning Repository, specifically data describing environmental factors like temperature and humidity for five cities in China between 2010 and 2015 [24]. The author filtered this data: 1) to remove samples with missing attributes as well as 2) include only data samples between 9:00am and 4:00pm during summer months. This filter was required to assist the unsupervised learning algorithm; it had significant problems with the variations between seasons as well as day and night. Ideally, more than five clusters would be employed (e.g. Guangzhou Daytime Summer, Guangzhou Nighttime Summer, Guangzhou Daytime Fall). The unsupervised learning algorithm attempted to categorize the training samples by city (from one of five cities). Test results showed that it did so with an accuracy of approximately 73.1%.

C. Computation Time vs. Computation Method

The second set of results presented in this paper compare computation times for all three of the computation methods presented above (Mathworks' Matlab, AndroidOS, and AWS Lambda) versus number of available training data samples. Figure 1 displays total computation time versus sample size and computation method for the unsupervised learning algorithm described above, using the author's training data set. This graph illustrates that the speed of AWS Lambda begins to exceed AndroidOS for training data sizes between 4000 and 8000 samples. For large data sets, AWS Lambda is faster than AndroidOS running this algorithm locally. It should be noted that AndroidOS running on the Moto G smartphone had significant difficulty processing training data sets larger than 6000, frequently crashing beyond this threshold. The author only included results for which the algorithm converged successfully in the graphs below. In all cases, Mathworks' Matlab running on a PC is expectedly the fastest and most efficient implementation.

D. *Computation Time vs. Read and Write Types*

The results presented in Figure 1 employ batch mode for all read and write operations with maximum batch sizes. Figure 2 illustrates exactly how the use of individual read and write operations affects computation time for unsupervised learning running on AWS Lambda. Note that this value considers the time required to load training data samples from as well as save results to DynamoDB.

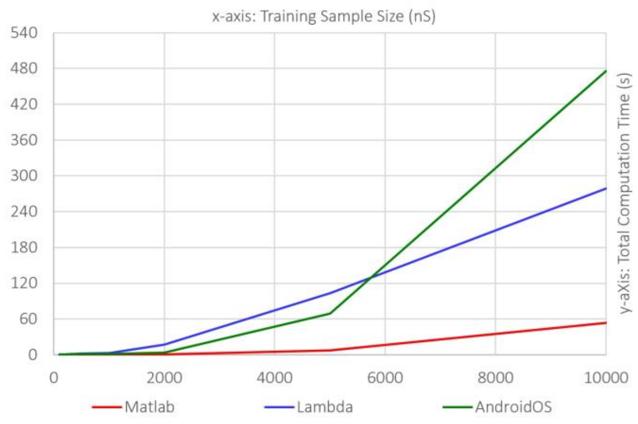

Figure 1: Computation Time vs. Sample Size and Computation Method for Unsupervised Machine Learning

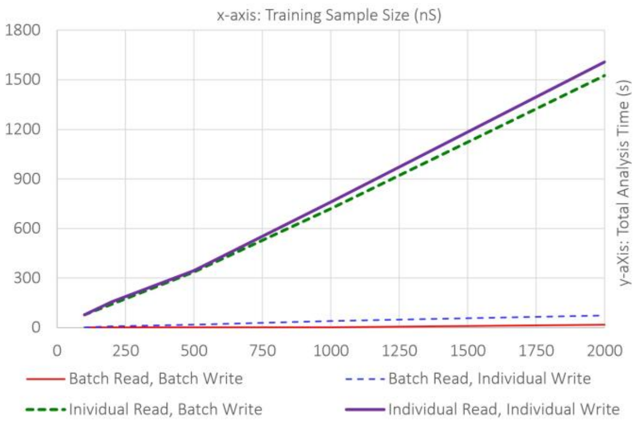

Figure 2: Computation Time vs. Training Data Set Size and Read/Write Type for Unsupervised Learning Running on AWS Lambda

Interestingly, the penalty (aka. increased computation time) associated with use of individual read operations is minimal as compared to that associated with individual write operations. In other words, the developer sees a significant speed increase from use of batch write operations but relatively small benefit from use of batch reads.

IX. Conclusion

A. *Summary*

This paper examines a novel approach to machine learning for efficient big data processing within composed primarily of mobile devices. Focus is placed on development of custom lambda functions that are lean, modular, and effective for machine learning. The results section illustrates that machine learning may be effectively implemented within a serverless cloud computing infrastructure, as highlighted by Figure 1 and Figure 2.

AWS Lambda, for this type of application, is more cost-efficient that use of AWS EC2 with Reserved Hosting. The author spent less than $20 in Lambda and DynamoDB-related fees as part of this research. This number may be compared to the minimum cost for purchase of reserved instances (t2.micro type) within AWS EC2: $118.00 for a one-year term with limit of 0.5 normalized units per hour and access to 2.0GB RAM. The training data set with 5000 samples itself requires approximately 0.75GB of storage. As such, this conservative plan would most probably run the machine learning algorithms at a reduced speed, considering the significant additional RAM required to perform the machine learning algorithms themselves and store intermediate data structures. There are no limits on memory use within Lambda. The key is that this work assumes an application, like IoT, where learning algorithms are performed infrequently, sporadically, and for short high-intensity bursts. This is where AWS Lambda is most cost-efficient. For extended periods of machine learning, as needed by other applications, EC2 may then be the more cost-efficient option. This is an issue any designer must consider.

B. *Future Work*

In the author's future work, he will investigate how unsupervised learning and clustering may be used as a preprocessor for supervised learning of an artificial neural network. One question readers may ask is why the training samples will be clustered before being supplied to the neural network. This idea comes from the author's personal experience working with sensors for IoT applications. As part of a previous project, the author collected significant physical data describing how the use of a device within a home related to use of other devices as well as environmental measurements (e.g. motion and temperature). He observed certain devices and measurements naturally clustered. This may be attributed to many factors including device locations, user habits, the need to use devices when humans are present, time of day, and naturally linked devices such as a television and DVR. Therefore, a single neural network is not the best representation of device use in a home or other commercial building. The use of multiple trained neural networks, each trained specifically with data from its associated cluster, generates more accurate predictions within less computing time and power. The “cluster averages” indicate which relationships between input and output are dominant, allowing synapse connections to be placed optimally. The result are multiple neural networks that have fewer synapse connections and are easier to train. The author’s preliminary work shows that this method has great promise for training data related to Internet of Things (IoT) and similar applications.

Another more significant advantage is the potential for parallelization. By dividing one neural network and set of training data into multiple clusters or sub-sets, parallel processing may be applied to the problem. Preliminary results demonstrate that, by applying parallelization, the proposed method may beat traditional approaches in terms of both accuracy as well as computation time.

X. References

[1] J. A. Bhat, V. George and B. Malik, "Cloud Computing with Machine Learning Could Help Us in the Early Diagnosis of Breast Cancer," in 2015 Second International Conference on Advances in Computing and Communication Engineering, 2015.

[2] P. Castro, V. Ishakian, V. Muthusamy and A. Slominsky, "Serverless Programming: Function as a Service," in 2017 IEEE 37th International Conference on Distributed Computing Systems, 2017.

[3] C. Mtita, M. Laurent and P. Daragon, "Serverless Lightweight Mutual Authentication Protocol for Small Mobile Computing Devices," in 2015 7th International Conference on New Technologies, Mobility and Security (NTMS), 2015.

[4] F. Sun, G.-B. Huang, J. Wu, S. Song and D. C. Wunsch, "Efficient and Rapid Machine Learning Algorithms for Big Data and Dynamic Varying Systems," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no. 10, p. 20, 2017.

[5] G. McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," in 2017 IEEE 3th International Conference on Distributed Computing Systems Workshop, 2017.

[6] J. Yuan and S. Yu, "Privacy Preserving Back-Propagation Neural Network Learning Made Practical with Cloud Computing," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 1, p. 9, 2014.

[7] G. McGrath, B. Judson, P. Brenner, J. Short and S. Ennis, "Cloud Event Programming Paradigms," in 2016 IEEE 9th International Conference on Cloud Computing, 2016.

[8] L. Columbus, "Roundup of Internet of Things Forecasts and Market Estimates 2016," Forbes Magazine, New York, NY USA, 2016.

[9] R. Pakdel and J. Herbert, "Adaptive Cost Efficient Framework for Cloud-Based Machine Learning," in 2017 IEEE 41st Annual Computer Software and Applications Conference, 2017.

[10] E. Oakes, L. Yang, K. Houck, T. Harter, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Pipsqueak: Lean Lambdas with Large Libraries," in 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops, 2017.

[11] G. McGrath, J. Short, S. Ennis, B. Judson and P. Brenner, "Cloud Event Programming Paradigms: Applications and Analysis," in 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), 2016.

[12] K. Li, C. Gibson, D. Ho, Q. Zhou, J. Kim, O. Buhisi, D. E. Brown and M. Gerber, "Assessment of Machine Learning Algorithms in Cloud Computing Frameworks," in 2013 IEEE Systems and Information Design Symposium, Charlottesville, VA USA, 2013.

[13] A. Eivy, "Be Wary of the Economics of Serverless Cloud Computing," IEEE Cloud Computing, vol. 4, no. 2, pp. 6 - 12, 2017.

[14] M. Srivatsa and L. Ling, "Mitigating Denial-of-Service Attacks on the Chord Overlay Network: A Location Hiding Approach," IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 4, pp. 512 - 527, 2009.

[15] C. Yeoh, E. Kendall and A. Khan, "An Agent Based Approach to Discovery and Formation of the Serverless Core Network," in 2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication, 2005.

[16] I. Baldini, P. Castro, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah and P. Suter, "Cloud-Native, Event-Based Programming for Mobile Applications," in 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2016.

[17] A. Amato, B. DiMartino, M. Scialdone and S. Venticinque, "A Negotiation Solution for Smart Grid Using a Fully Decentralized, P2P Approach," in 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, 2015.

[18] D. Vanderweide, Invited Talk on Advantages of Serverless Cloud Computing at 2017 NYC Big Cloud Expo, New York, NY, 2017.

[19] "Amazon Web Services Shared Responsibility Model," Amazon, 21 September 2017. [Online]. Available: https://aws.amazon.com/compliance/shared-responsibility-model/. [Accessed 21 September 2017].

[20] C. W. Chen, J. Luo and K. J. Parker, "Image Segmentation via Adaptive K-Mean Clustering and Knowledge-Based Morphological Operations with Biomedical Applications," IEEE Transactions on Image Processing, vol. 7, no. 12, p. 10, 1998.

[21] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, New York, NY: Wiley Books, 1990.

[22] F. Cao, J. Liang and G. Jiang, "An Initialization Method for the K-Means Algorithm Using Neighborhood Model," Elsevier Journal of Computers and Mathematics with Applications, vol. 58, no. 3, p. 9, 2009.

[23] R. Rojas, Neural Networks: A Systematic Approach, New York, NY USA: Springer-Verlag Publishing, 1996.

[24] S. X. Chen, "University of California at Irvine Machine Learning Repository," 21 October 2017. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities. [Accessed 21 October 2017].

[25] X. Bu, J. Rao and C.-Z. Xu, "Coordinated Self-Configuration of Virtual Machines and Appliances Using a Model-Free Learning Approach," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 4, p. 8, 2013.