Parallel Hierarchical Affinity Propagation with MapReduce

Dillon Mark Rose[*], Jean Michel Rouly[†], Rana Haber[‡], Nenad Mijatovic[§], Adrian M. Peter[¶]

[*]Computer Science Department, Florida Institute of Technology, Melbourne, Florida; Email: drose2010@my.fit.edu
[†]Computer Science Department, George Mason University, Fairfax, Virginia; Email: jrouly@gmu.edu
[‡]Mathematical Sciences Department, Florida Institute of Technology, Melbourne, Florida; Email: rhaber2010@my.fit.edu
[§]Electrical Engineering Department, Florida Institute of Technology, Melbourne, Florida; Email: nmijatov2005@my.fit.edu
[¶]Engineering Systems Department, Florida Institute of Technology, Melbourne, Florida; Email: apeter@fit.edu

Abstract—The accelerated evolution and explosion of the Internet and social media is generating voluminous quantities of data (on zettabyte scales). Paramount amongst the desires to manipulate and extract actionable intelligence from vast big data volumes is the need for scalable, performance-conscious analytics algorithms. To directly address this need, we propose a novel MapReduce implementation of the exemplar-based clustering algorithm known as Affinity Propagation. Our parallelization strategy extends to the multilevel Hierarchical Affinity Propagation algorithm and enables tiered aggregation of unstructured data with minimal free parameters, in principle requiring only a similarity measure between data points. We detail the linear run-time complexity of our approach, overcoming the limiting quadratic complexity of the original algorithm. Experimental validation of our clustering methodology on a variety of synthetic and real data sets (*e.g.* images and point data) demonstrates our competitiveness against other state-of-the-art MapReduce clustering techniques.

Index Terms—MapReduce, Cluster, Affinity Propagation, Hierarchical Affinity Propagation, Hadoop

I. INTRODUCTION

In 2010, big data was growing at 2.5 quintillion [1] bytes per day. This overwhelming volume, velocity, and variety of data can be attributed to the ubiquitously spread sensors, perpetual streams of user-generated content on the web, and increased usage of social media platforms—Twitter alone produces 12 terabytes of tweets every day. The sustained financial health of the world's leading corporations is intimately tied to their ability to sift, correlate, and ascertain actionable intelligence from big data in a timely manner. These immense computational requirements have created a heavy demand for advanced analytics methodologies which leverage the latest in distributed, fault-tolerant parallel computing architectures. Among a variety of choices, MapReduce has emerged as one of the leading parallelization strategies, with its adoption rapidly increasing due to the availability of robust open source distributions such as Apache Hadoop [2]. In the present work, we develop a novel MapReduce implementation of a fairly recent clustering approach [3], and demonstrate its favorable performance for big data analytics.[1]

Clustering techniques are at the heart of many analytics solutions. They provide an unsupervised solution to aggregate similar data patterns, which is key to discovering meaningful insights and latent trends. This becomes even more necessary, but exponentially more difficult, for the big data scenario. Many clustering solutions rely on user input specifying the number of cluster centers (e.g. K-Means clustering [4] or Gaussian Mixture Models [5]), and biasedly group the data into these desired number of categories. Frey et al. [6] introduced an exemplar-based clustering approach called Affinity Propagation (AP). As an exemplar-based clustering approach, the technique does not seek to find a mean for each cluster center, instead certain representative data points are selected as the exemplars of the clustered subgroups. The technique is built on a message passing framework where data points "talk" to each other to determine the most likely exemplar and automatically determine the clusters, *i.e.* there is no need to specify the number of clusters a priori. The sole input is the pairwise similarities between all data points under consideration for clustering—making it ideally suited for a variety of data types (categorical, numerical, textual, etc.). A recent extension of the AP clustering algorithm is Hierarchical Affinity Propagation (HAP) [3], which groups and stacks data in a tiered manner. HAP only requires the number of hierarchy levels as input and the communication between data points occurs both within a single layer and up and down the hierarchy. To date, AP and HAP have been mainly relegated to smaller, manageable quantities of data due to the prohibitive quadratic run time complexity. Our investigations will demonstrate an effective parallelization strategy for HAP using the MapReduce framework, for the first time enabling applications of these powerful techniques on big data problems.

First introduced by Google [7], the MapReduce framework is a programming paradigm designed to facilitate the distribution of computations on a cluster of computers. The ability to distribute processes in a manner that takes the computations to the data is key when mitigating the computational cost of working with extremely large data sets. The parallel programming model depends on a mapper phase that uses key-value identifiers for the distribution of data and subsequent independent executions to generate intermediate results. These are then gathered by a reducing phase to produce the final output key-value pairing. This simple, yet widely applicable

[1]Implementation available at http://research2.fit.edu/ice/?q=software

IEEE computer society

parallelization philosophy has empowered many to take machine learning algorithms previously demonstrated only on "toy data" and scale them to enterprise-level processing [8]. In this same vein, we adopted the most popular open source implementation of the MapReduce programming model, Apache Hadoop, to develop the *first ever* parallelized extension of HAP, which we refer to as MapReduce Hierarchical Affinity Propagation (MR-HAP). This allows efficient fault-tolerant clustering of big data, and more importantly, improves the run time complexity to potentially linear time (given enough machines).

A. Relevant Work

To handle the explosion of available data, there is now a vast amount of research in computational frameworks to efficiently manage and analyze these massive information quantities. Here we focus on the MapReduce framework [1], [9], [10], [11] for faster and more efficient data mining, covering the most relevant to our approach.

Among the state of the art MapReduce clustering algorithms is Hierarchical Virtual K-means (HVKM), which was implemented by Nair et al. [12]. HVKM uses cloud computing to handle large data sets, while supporting top to bottom hierarchies or a bottom to top approach. Since it derives its roots from K-means, HVKM requires one to specify the number of clusters. Our MR-HAP implementation does not require presetting the number of required clusters; it instead organically and objectively discovers the data partitions.

In Wu et al. [13], the authors propose to parallelize AP on the MapReduce framework to cluster large scale E-learning resources. The parallelization happens on the individual message level of the AP algorithm. We perform a similar parallelization but significantly go beyond and allow for hierarchical clustering, which enables a deeper understanding of the data's semantic relationships. In addition, our development is designed to work on a variety of data sources; thus, our experiments will showcase results on multiple data modalities, including images and numerical data, as shown in § IV.

The rest of this paper is organized as follows. In the next section, §II, we detail the non-parallel HAP algorithm. §III discusses the MapReduce paradigm and the implementation details for these algorithms. The experimental validations provided in §IV demonstrate our favorable performance against another clustering algorithm which is readily available in the open source project Apache Mahout [14]. Finally, we conclude with a summary of our efforts and future recommendations.

II. HIERARCHICAL AFFINITY PROPAGATION

AP is a clustering algorithm introduced by Frey et al.[6] motivated by the simple fact that given pairwise similarities between all input data, one would like to partition the set to maximize the similarity between every data point and its cluster's exemplar. Recall that an exemplar is an actual data point that has been selected as the cluster center. As we will briefly discuss, these ideas can be represented as an algorithm in a message passing framework. In the landscape of clustering

methodologies, which includes such staples as K-means [4], K-medoids [15], and Gaussian Mixture Models [5], predominantly all methods require the user to input the desired number of cluster centers. AP avoids this artificial segmentation by allowing the data points to communicate amongst themselves and organically give rise to a partitioning of the data. In many applications an exemplar-based clustering technique gives each cluster a more representative and meaningful prototype for the center, versus a fabricated mean.

HAP, introduced by [3], extends AP to allow tiered clustering of the data. The algorithm starts by assuming that all data points are potential exemplars. Each data point is viewed as a node in a network connected to other nodes by arcs such that the weight of the arcs s_{ij} describes how similar the data point with index i is to the data point with index j. HAP takes as input this similarity matrix where the entries are the negative real valued weights of the arcs. Having the similarity matrix as the main input versus the data patterns themselves provides an additional layer of abstraction—one that allows seamless application of the same clustering algorithm regardless of the data modality (*e.g.*text, images, general features, etc.). The similarity can be designed to be a true metric on the feature space of interest or a more general non-metric [6]. The negative of the squared Euclidean distance if often used as a metric for the similarities. The diagonal values of the similarity matrix, s_{jj}, are referred to as the "preferences" which specify how much a data point j wants to be an exemplar. Since the similarity matrix entries are all negative values, $-\infty < s_{ij} \leq 0$, $s_{jj} = 0$ implies data point j has high preference of being an exemplar and $s_{jj} \approx -\infty$ implies it has very low preference. In some cases, as in [6], [3], [16], the preference values are set using some prior knowledge; for example, uniformly setting them to the average of the maximum and minimum values of s_{ij}, or by setting them to random negative constants. Through empirical verification, we experienced better performance with randomizing the preferences and adopt this approach for most of our experiments. Once the similarity matrix is provided to the HAP algorithm, the network of nodes (data points) recursively transmits two kinds of intra-level messages between each node until a good set of exemplars is chosen. The first message is known as the "responsibility" message and the second as the "availability" message. The responsibility messages, ρ_{ij}^l, are sent at level l from data point i to data point j portraying how suitable node i thinks node j is to be its exemplar. Similarly, availability messages, α_{ij}^l, are sent at level l from data point j to i, indicating how available j is to be an exemplar for data point i. The responsibility and availability update equations are given in Eq. 1 and Eq. 2, respectively.

$$\rho_{ij}^{l>1} \leftarrow s_{ij}^l + \min[\tau_i^l, -\max_{ks.t.k \neq j}\left\{\alpha_{ik}^l + s_{ik}^l\right\}] \quad (1)$$

$$\alpha_{ij}^{l<L} \leftarrow \min\left\{0, c_j^l + \phi_j^l + \rho_{jj}^l + \sum_{ks.t.k \notin \{i,j\}}\max\{0, \rho_{kj}^l\}\right\}(2)$$

$$\alpha_{jj}^{l<L} \leftarrow c_j^l + \phi_j^l + \sum_{ks.t.k \neq j}\max\{0, \rho_{kj}^l\} \quad (3)$$

where L is the number of levels defined by the user and $l \in \{1, \cdots, L\}$. Eq. 3 is the self-availability equation which reflects the accumulated positive evidence that j can be an exemplar. The self-responsibility messages are updated the same way as the responsibility messages. To avoid numerical oscillation, the responsibility and availability messages are dampened by $\lambda \in (0,1)$ at every level l.

HAP also introduces two inter-level messages. These messages are denoted by τ in Eq. 4, which receives messages from the lower level and ϕ in Eq. 5, which receives messages from the upper level. At every level, the cluster preference c_i^l is updated using Eq. 6.

$$\tau_j^{l+1} \quad = \quad c_j^l + \rho_{jj}^l + \sum_{k s.t. k \neq j} \max(0, \rho_{kj}^l) \qquad (4)$$

$$\phi_i^{l-1} \quad = \quad \max_k (\alpha_{ik}^l + s_{ik}^l) \qquad (5)$$

$$c_i^l \quad \leftarrow \quad \max_j (\alpha_{ij}^l + \rho_{ij}^l) \qquad (6)$$

A variety of strategies can be employed to update the similarity matrix s_{ij}^l to vary level-wise. We have achieved good results by simply taking into consideration the cluster relationship of the previous level:

$$s_{ij}^{l+1} = s_{ij}^l + \kappa \max_{j s.t. j \neq i} [\alpha_{ij}^l + \rho_{ij}^l] \qquad (7)$$

where κ is a constant value within [0,1]. This updates the relation between data points in level $l+1$ by negatively increasing the similarity between points that belong to different clusters in level l and enforces the similarity between points that fall under the same cluster in level l.

After all messages have been sent and received, the cluster assignments are chosen, at every level, based on the maximum sum of the availability and responsibility messages as in Eq. 8. These cluster assignments can be used to extract the list of exemplars.

$$e_i^l \quad \leftarrow \quad \arg\max_j \{\alpha_{ij}^l + \rho_{ij}^l\} \qquad (8)$$

These net message exchanges seek to maximize the cost of correctly labeling a point as an exemplar and gathering its representative members (a cluster). In Algorithm 1, we detail the pseudo-code implementation of HAP. Given this description of HAP, we now proceed to discuss MapReduce and our novel parallelization strategy.

III. MapReduce Hierarchical Affinity Propagation

The MapReduce programming model [7] is an abstract programming paradigm independent of any language that allows the processing workload of the implemented algorithm to be balanced over separate nodes within a computer cluster. Our overarching MapReduce approach for HAP was motivated by viewing the major update equations for HAP (see Algorithm 1) as tensorial mathematical constructs [17]. One can simply view these tensorial constructs as two or three dimensional matrices. The HAP algorithm can be parallelized because all the updates to the various tensors require only a subset of the information provided. Therefore, the updates can be split up

---

Algorithm 1 Hierarchical Affinity Propagation
1: **Input:** Similarity (S), Levels (L), Iterations, and λ
2: **Initialize:** $\alpha = 0$, $\rho = 0$, $\tau = \infty$, $\phi = 0$, $c = 0$, $e = 0$
3: **for** $iter = 1 \rightarrow$ Iterations **do**
4:     **for** $l = 1 \rightarrow$ Levels **do**
5:         Update ρ_{ij}^l (eq. 1) & Dampen ρ^l
6:         Update α_{ij}^l (eq. 2 & 3) & Dampen α^l
7:         Update τ_j^l, ϕ_j^l & c_j^l (eq. 4, 5 & 6)
8:         **Optional** Update s_{ij}^l (eq. 7)
9:     **end for**
10: **end for**
11: **for** $l = 1 \rightarrow$ Levels **do**
12:     Update e_j^l (8)
13: **end for**

---

into different jobs and each job will receive the subset of data needed to evaluate the update.

To achieve a balance between computational partitioning and efficient formatting for data representation on the Hadoop Distributed Filesystem (HDFS), all the data is constructed as three dimensional tensors. In support of the fault tolerance aspect of MapReduce, it is important to retain a copy at all times of the S, α, ρ, c, τ, and ϕ tensors. (Recall S, α, ρ, and c refer to the Similarity, Availability, Responsibility, and Cluster Preferences, respectively.) To this end, even those tensors not required by a job must be passed directly through to the next job. For the S, α, and ρ tensors, the dimensions represent the nodes, the exemplars, and the levels. Since there are N nodes, N possible exemplars, and L levels, these tensors contain LN^2 values. For the c, τ, and ϕ tensors, the first two dimensions represent the index and level and the depth dimension has length one. Since there are N indices and L levels, these tensors contain LN values. In the sequel, for the S, α, and ρ tensors, the node dimension will be iterated by i, the exemplar dimension will be iterated by j, and the level dimension iterated by l. As for the c, τ, and ϕ tensors, the index dimensions will be iterated by both i and j.

With these underlying structures, data must be deconstructed and represented as (key,value) pairs for use in the MapReduce framework. There are two formats for storing the information: node-based and exemplar-based formatting. In the *node-based format*, the keys are string tuples, (i, l, ξ), where i represents the node, l represents the level, and ξ represents the tensor $(\alpha, \rho, ...)$. The values, represented by ν, are the vectors for the ith node of the matrix on the lth level of the tensor. In the *exemplar-based format*, the keys are string tuples, (j, l, ξ), where j represents the exemplar, l represents the level, and ξ represents the tensor. The values, represented by ν, are the vectors for the jth exemplar of the matrix on the lth level of the tensor. With the data thus represented, MapReduce jobs must be constructed to manipulate the information using the given HAP equations.

In our parallelization scheme, MR-HAP is broken down into three separate MapReduce jobs. The first job handles updating τ, c, and ρ. The second job handles updating ϕ and α. These
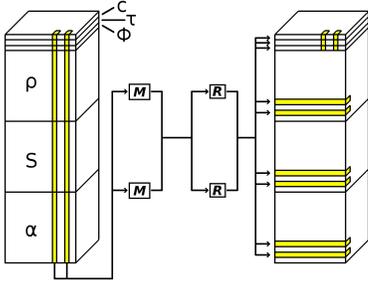
Fig. 1. Parallelization Scheme

first two jobs loop for a set number of iterations. At the end of the iterations, the final job extracts the cluster assignments on each level. Due to dependencies set out in the equations, the ρ update must occur first. Therefore, τ and c are not updated during the first iteration. In all other iterations they occur before the Responsibility update. At the start of each iteration, the data will be in exemplar-based format. After the first job, the data will have switched to node-based format. The second job converts the data back to exemplar-based format to begin a new iteration or to be used as input to the final job. See Fig. 1 for a visual representation of the parallelization scheme. The figure represents what happens to the data during either of the first two jobs. The tensors have been stacked to show how the indices line up. The yellow strips on the left represent information being passed to one mapper, one strip per mapper. The focus of each mapper is on providing the reducers with the necessary information. The subsequent focus of each reducer is on performing the tensor updates as defined in the HAP equations. As the data comes out of the job, the switch between exemplar-based and node-based formats can be easily seen. The output is now ready for use by the next job, which will follow a similar flow. The following sections will provide in-depth explanations of each MapReduce job.

1) Updating τ, c, and ρ: This job takes as input the exemplar-based representation of the data and outputs the node-based representation of the data with updated values. In the first iteration, τ and c are not updated due to previously mentioned dependencies. In this MapReduce job, the mapper deconstructs the exemplar-based vectors into node-based values for the reducer to reconstruct node-based vectors. Each mapper receives a key describing a unique (j, l, ξ) combination and a value with the corresponding vector. The indices of the vector represent the nodes; thus, the mapper iterates over the vector with i. Each reducer receives a key describing a unique (i, l) combination and a list of values which will be used to reconstruct the 6 node-based vectors, the 2 node-based vectors from the level below and the 2 special diagonal vectors. The indices of the constructed vector represent the exemplars so the reducer iterates over the vector with j.

2) Updating α and ϕ: This job takes as input the node-based representation of the data and outputs the exemplar-based representation of the data with updated values. In this MapReduce job, the mapper deconstructs the node-based

vectors into exemplar-based values for the reducer to reconstruct exemplar-based vectors. Each mapper receives a key describing a unique (i, l, ξ) combination and a value with the corresponding vector. The mapper iterates over the vector with j. Each reducer receives a key describing a unique (j, l) combination and a list of values which will be used to reconstruct the 6 exemplar-based vectors and the 2 node-based vectors from the level above. The indices of the constructed vector represent the nodes so the reducer iterates over the vector with i.

3) Extracting Cluster Assignments: This job takes as input the exemplar-based representation of the data and outputs the cluster assignments. In this MapReduce job, the mapper deconstructs the exemplar-based vectors into node-based values for the reducer to reconstruct node-based vectors. Each mapper receives a key describing a unique (j, l, ξ) combination and a value with the corresponding vector. The mapper iterates over the vector with i. Since this is the last step, only the required information has to pass to the reducer and the other information can be neglected. Each reducer receives a key describing a unique (i, l) combination and a list of values which will be used to reconstruct the 2 node-based vectors and the 2 special diagonal vectors. The reducer iterates over the vector with j.

A. Runtime Complexity

A standard sequential HAP implementation must necessarily have a runtime complexity of $O(kLN^2)$ where k represents the number of algorithmic iterations, run either as a hard limit or until convergence is reached, L represents the number of output levels requested, and N represents the cardinality of the input data set, such that the size of S is $(L \times N \times N)$. The runtime complexity is a direct result of iterating over all three dimensions of the tensors for each iteration. By implementing the algorithms in the MapReduce framework, we are able to achieve superior runtime complexity. Under MapReduce, the MR-HAP runtime complexity reduces to a linear relationship with the data, assuming the total number of Virtual Machines (VMs) on the cluster, M, scales to LN, *i.e.* $O(\frac{kLN^2}{M}) = O(kN)$ as $M \to LN$. In MR-HAP, M can only scale up to a maximum of LN because M is limited to the number of tasks that can be evaluated at the same time. In this case, it is limited to the minimum of the $6LN$ mapper tasks and the LN reducer tasks, where the constant factor six represents the number of tensor identifications introduced into the algorithm, namely $\alpha, \rho, S, \tau, \phi$, and c.

IV. Experimental Results

To demonstrate the effectiveness and adaptability of the proposed approach, we executed validation experiments on several data sets with a variety of modalities, *e.g.* imagery and synthesized numerical point data. Where applicable, we compared our performance to a popular MapReduce hierarchical clustering algorithm currently available in the Mahout library. At its core, their hierarchical clustering is based on a level-wise K-means clustering approach; thus, we refer to it
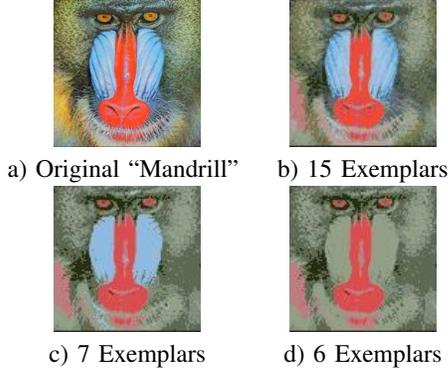
a) Original "Mandrill"   b) 15 Exemplars

c) 7 Exemplars   d) 6 Exemplars

Fig. 2. Hierarchical clustering of "Mandrill" 103x103. See text for discussion.

a) Original "Buttons"   b) 154 Exemplars

c) 25 Exemplars   d) 11 Exemplars

Fig. 3. Hierarchical clustering of "Buttons" 120x100. See text for discussion.

as Hierarchical K-Means (HK-Means). With K-means as the foundation, HK-Means requires the number of cluster centers as input. Since our method does not explicitly impose this requirement, we adopted the initialization method of running Canopy clustering, also available in Mahout, to discover the "natural" number of centers. We then use these cluster centers to seed HK-Means. In order to truly gain an objective understanding of MR-HAP performance versus HK-Means, we use the *purity extrinsic cluster quality metric* to assess their respective aggregation capabilities [18].

A. Image Segmentation

Hierarchical Affinity Propagation performs very well in image segmentation tasks as shown in Fig. 2 & Fig. 3. The "Mandrill" image, Fig. 2, is of size 103×103, which provided 10,609 pixels (data points) to cluster. Similarly, the "Buttons" image, Fig. 3, is of size 120×100, resulting in a data set of 12,000 pixels. The similarity input was computed using the negative Euclidean distance between all pixels treating RGB intensities as vectors. The diagonal, or preference entries, were selected as random numbers within $[-10^6, 0]$. As for the other parameters, we set the iterations to 30 and the dampening factor to $\lambda = 0.5$. To generate the clustered images, we re-color all pixels within a cluster with the color of the selected exemplar. The number of hierarchy levels for the "Mandrill" data set was set to $L = 3$. The top right image is the lowest level where the pixels were grouped into 15 clusters. The bottom left image is the second level where the pixels were grouped into 7 clusters. Finally, the bottom right image is the highest level where the pixels were grouped into 6 clusters. From these images we can still see the mandrill's shape and most of its colors, but at the highest level it appears fuzzier. This is because the members of the same clusters were given the color of the exemplar.

For the "Buttons" image, the number of levels was set to $L = 3$. The top right image is the lowest level where the pixels were grouped into 154 clusters. The bottom left image is the second level where the pixels were grouped into 25 clusters. Finally, the bottom right image is the highest level where the pixels were grouped into 11 clusters. The highest level of the hierarchy appears fuzzier than the original image due to similar colors clustering underneath a single exemplar.
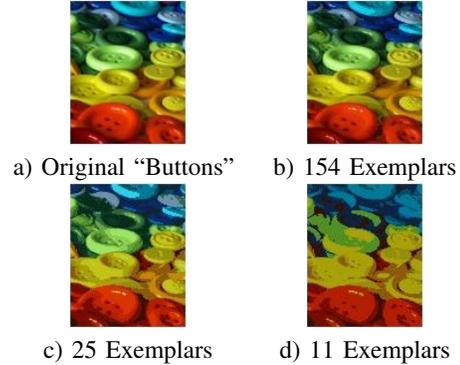
B. Scalability and Comparison to HK-Means

In order to test the scalability of the MR-HAP algorithm with respect to speed, we use the data set "Aggregation" [19], which is a shape set composed of 788 two-dimensional points. The purpose of these tests was to observe trends in algorithm runtime as cluster computing power increased, as well as to determine the benefits of running in a distributed environment as compared to an undistributed environment (a single-machine Hadoop cluster). Hadoop clusters were provided using Amazon Elastic MapReduce (EMR) to dynamically create clusters of standard Amazon Elastic Compute Cloud (EC2) instances. Cluster computing power was scaled both by increasing the number of VMs within a cluster and by provisioning more powerful VMs. The two VM instance types used are: (1) the m1.small, which has 1.7 GB of memory and is considered to have 1 EC2 Compute Unit (ECU) with 160 GB of instance storage and a 32-bit architecture, and (2) the m1.xlarge, which has 15 GB of memory, 8 ECU, 1,690 GB of instance storage, and a 64-bit architecture. The single-machine Hadoop cluster utilized to simulate an undistributed environment has 8 GB of memory, 8 ECU, 40 GB of machine storage, and a 64-bit architecture.

For comparison to another state-of-the-art MapReduce clustering methodology, our MR-HAP algorithm was benchmarked against HK-Means. Due to its inherently parallel design, MR-HAP immediately begins to benefit from being placed in a distributed environment. Represented by a solid blue line in Fig. 4, MR-HAP runtime decreases by 64%, from 320 minutes to 115 minutes, when cluster computational power is increased by just 4 additional ECU. MR-HAP eventually reaches the threshold of a linear relationship with the size of the input data at a runtime of around 20 minutes, which is a 94% decrease from the single ECU cluster. Furthermore, at its best, MR-HAP performs 66% faster in a distributed environment than the undistributed environment which is represented by the blue dotted line in Fig. 4. In contrast, the Mahout HK-Means algorithm used in this experimentation, indicated by the solid green line in Fig. 4, is not parallelized to the extent of MR-HAP. Each single iteration of K-Means is structured under Mahout to distribute over a Hadoop cluster, but the hierarchical "Top Down" structure requires iterative
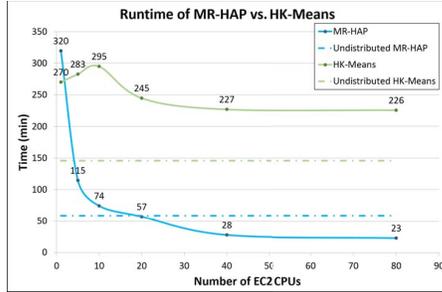
Fig. 4. Time vs. Number of EC2 CPUs. Our MR-HAP better utilizes available compute resources to significantly improve runtime.

Fig. 5. Purity levels of MR-HAP vs. HK-Means. MR-HAP posts results highly competitive with HK-Means.

executions of K-Means for each level. This lack of an overall parallelization scheme results in reduced performance at scale than MR-HAP. HK-Means runtime initially increases by 8.5% when ECU is increased from 1 to 10 due to Hadoop cluster overhead, including network latency and I/O time. However, at 10 ECU, HK-Means overcomes this overhead and begins to benefit from the MapReduce parallelization scheme. This results in an eventual 16% runtime decrease between 1 and 80 ECU, at which point HK-Means eventually reaches a linear relationship with the data at a runtime around 225 minutes. Unlike MR-HAP, HK-Means never surpasses its undistributed runtime threshold of 146 minutes indicated by the green dotted line in Fig. 4. Finally, at its best, HK-Means runs 90% slower than MR-HAP, requiring 226 minutes of execution compared to MR-HAP's 23 minutes. With significantly faster runtimes, MR-HAP still posts purity levels competitive with HK-Means, shown in Fig. 5. This combination of speed and high performance is ideal for processing big data in a large-scale cloud computing environment.

V. CONCLUSION

The need for efficient and high performing data analysis frameworks remain paramount to the big data community. The AP clustering algorithm is rapidly becoming a favorite amongst data scientists due to its high quality grouping capabilities, while requiring minimal user specified parameters. Recently, a multilayer structured version of the AP algorithm, HAP, was introduced to automatically extract tiered aggregations inherent in many data sets. HAP is modeled as a message-based network that allows communication between nodes and between levels in the hierarchy, and mitigates many of the biases that arise in techniques that require one to input the number of clusters. In the present work, we have developed the first ever extension, MR-HAP, to address the big data problem—demonstrating an efficient parallel implementation using MapReduce that directly improves the runtime complexity from quadratic to linear. The novel tensor-based partitioning scheme allows for parallel message updates and utilizes a consistent data representation that is leveraged by map and reduce tasks. Our approach seamlessly allows us to cluster a variety of data modalities, which we experimentally showcased on data sets ranging from synthetic numerical points to imagery. Our analysis and computational perfor-
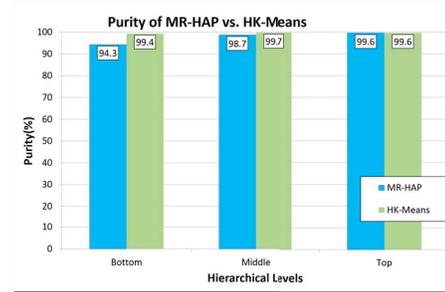
mance is competitive with the state-of-the-art in MapReduce clustering techniques.

REFERENCES

[1] S. Humbetov, "Data-intensive computing with MapReduce and Hadoop," in *AICT*, 2012, pp. 1–5.
[2] T. A. S. Foundation, "Hadoop 1.1.2 documentation," The Apache Software Foundation, 03 2013. [Online]. Available: http://hadoop.apache.org/docs/r1.1.2
[3] I. E. Givoni, C. Chung, and B. J. Frey, "Hierarchical affinity propagation," *CoRR*, vol. abs/1202.3722, 2012.
[4] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Applied Statistics*, vol. 28, pp. 100–108, 1978.
[5] G. McLachlan and D. Peel, *Finite Mixture Model*. John Wiley & Sons, Inc, 2000.
[6] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, 2007.
[7] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *OSDI*. USENIX Association, 2004, pp. 10–10.
[8] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, "MapReduce for Machine Learning on multicore," in *NIPS*, 2007, pp. 281–288.
[9] C. Zewen and Z. Yao, "Parallel text clustering based on mapreduce," in *CGC*, 2012, pp. 226–229.
[10] H. Wang, Y. Shen, L. Wang, K. Zhufeng, W. Wang, and C. Cheng, "Large-scale multimedia data mining using MapReduce framework," in *IEEE CloudCom*, 2012, pp. 287–292.
[11] R. M. Esteves, T. J. Hacker, and C. Rong, "Cluster analysis for the cloud: Parallel competitive fitness and parallel k-means++ for large dataset analysis," in *IEEE CloudCom*, 2012, pp. 177–184.
[12] T. Nair and K. Madhuri, "Data mining using hierarchical virtual k-means approach integrating data fragments in cloud computing environment," in *IEEE CCIS*, 2011, pp. 230–234.
[13] F. Wu, W. Wang, H. Zhang, and Y. Zhuang, *Handbook of Research on Hybrid Learning Model: Advanced Tools, Technologies, and Applications*. IGI Global, 2010, ch. The Clustering of Large Scale E-Learning Resources, pp. 94–104.
[14] T. A. S. Foundation, "Apache Mahout: Scalable machine learning and data mining," The Apache Software Foundation, 2013. [Online]. Available: http://mahout.apache.org
[15] L. Kaufman and P. Rousseeuw, "Clustering by means of medoids," *Reports of the Faculty of Mathematics and Informatics, Delft University Technology*, vol. 87-3, 1987.
[16] J. Xiao, J. Wang, P. Tan, and L. Quan, "Joint affinity propagation for multiple view segmentation," in *IEEE Computer Vision*, 2007, pp. 1–7.
[17] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data," *Pattern Recognition*, vol. 44, no. 7, pp. 1540–1551, Jul. 2011.
[18] N. Sahoo, J. Callan, R. Krishnan, G. Duncan, and R. Padman, "Incremental hierarchical clustering of text documents," in *CIKM*. ACM, 2006, pp. 357–366.
[19] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation," *ACM TKDD*, vol. 1, Mar. 2007.