2012 IEEE 4th International Conference on Cloud Computing Technology and Science

Cluster Analysis for the Cloud:

Parallel Competitive Fitness and Parallel K-means++ for Large Dataset Analysis

Rui Máximo Esteves
Department of Electrical and Computer Engineering
University of Stavanger, Norway
rui.esteves@uis.no

Thomas Hacker
Department of Computer and Information Technology
Purdue University, West Lafayette, USA
tjhacker@purdue.edu

Chunming Rong
Department of Electrical and Computer Engineering
University of Stavanger, Norway
chunming.rong@uis.no

Abstract— **The amount of resources needed to provision Virtual Machines (VM) in a cloud computing systems to support virtual HPC clusters can be predicted from the analysis of historic use data. In previous work, Hacker *et al.* found that cluster analysis is a useful tool to understand the underlying spatio-temporal dependencies present in system fault and use logs. However, the cluster analysis used for reducing spatio-temporal dependences should be fast and accurate to understand the underlying stochastic properties of these systems. K-means is a fast cluster analysis method, in which accuracy depends on the use of initialization algorithms that are usually serial and slow. In this paper we present two new parallel strategies for fast seeding K-means cluster analysis. Both strategies were tested on a real problem where the aim was to reduce spatial and temporal dependencies of failures on large supercomputer systems. The performance of both strategies were compared with five existing serial implementations: K-means implementations of 1) Lloyd (L); 2) McQueen (M); and 3) Hartigan – Wong (HW), all of them using Forgy seeding; 4) K-means++; and 5) Neural Gas clustering (NG), a more recent and sophisticated method. Our results show that our new Parallel Competitive Fitness approach reduces the Within Sum of Squares (WSQQ) measure, thus increasing cluster quality of the three K-means implementations: L; M; HW, and is 200 times faster than the existing serial K-means++. The existing serial and our new Parallel K-means++ have the lowest WSQQ. Our new Parallel K-means++ is twice as fast as the existing serial K-means++ method, and is 4 times faster than the NG method. Moreover, our new methods did not generate empty clusters, while NG did. As a result of our new techniques, predicting the amount of resources needed to provision VMs processing historic system fault and use data can now be done faster and with more accuracy.**

Keywords-component; VM resources prediction, Clustering, K-means, parallel K-means++, Neural Gas

I. INTRODUCTION

The reliability of HPC clusters and cloud computing systems depends directly on the reliability of the constituent components that make up the system. When a component fails, applications that rely on the correct functioning of the entire computer system fail. In response to this problem, the use of virtualization technologies that facilitate the use of proactive and reactive fault tolerance approaches is a useful and interesting alternative. In previous work [1], we described a methodology to predict the amount of resources needed to provision Virtual Machines (VM) in a cloud computing systems to support virtual HPC clusters. One aspect of the problem of virtual HPC cluster provisioning is the need to understand and predict the reliability and availability of physical node resources. To do this, we need to process real log data from systems in the field. However, the spatial and temporal dependencies present in the observation data prevented us from directly using the log data to develop a reliability model. To understand the underlying stochastic characteristics of failures on these real systems we preformed cluster analysis. For clustering analysis we tried a fast K-means method with random initialization and the results did not present satisfactory quality. We tried a more sophisticated and accurate method for cluster analysis, the neural gas, and we found that it was very slow, and could not complete within a reasonable time frame needed for the analysis. To solve this problem, we developed the approaches described in the paper.

A very fast method commonly used in cluster analysis is the K-means [2]. However, the accuracy and running time of K-means clustering depends heavily on the position of the initial cluster centers [2], [3]. K-means provides a solution that can be trapped at a local minimum and there is no guarantee that it corresponds to the best possible solution [4]. Since the speed and accuracy of the clustering are greatly influenced by the choice of the initial centers, the success of the K-means analysis is influenced by a good initialization.

In this paper, we propose two new strategies for seeding K-means that can exploit parallel processing. The first strategy, *Parallel Competitive Fitness (PCF)*, uses the concept of competitive fitness to optimize the cluster analysis quality. The second, *Parallel K-means++ (PKM)*, is a parallel implementation of the existing serial K-means++ method.

PCF, our new first strategy, runs parallel instances of K-means. It seeks to quickly decrease the probability of converging to local minimums during cluster analysis. The main idea of this strategy is to run several instances of K-means in parallel with random initial centers. The result of each instance is then scored using a fitness measure. We run competitive trials of K-means analyses, hill-climbing to find a better result. The hill-climbing effort is terminated when no

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

better solution can be found after a fixed number *im* of consecutive unsuccessful competitive trials. We call this fixed number *im* the *heuristic improvement factor*. Increasing the *heuristic improvement factor* improves the chances of finding a globally better solution at the cost of increased running time. Within each K-means run, the maximum number of iterations for each individual K-means run is restricted to a fixed value (*max_iter*). Using PCF, we can use a smaller value of *max_iter* for each K-means trial, decreasing the running time of individual K-means runs. The benefits of our approach are realized through the use of competitive trials to discover the best K-means run.

The insight for the Parallel Competitive Fitness strategy comes from our empirical experience of clustering a dataset containing thousands of samples. For our analysis, we repeated K-means cluster analysis several times using a generous maximum number of iterations (*max_iter*). We observed that using random seeding, the algorithm could converge in just few minutes or could take more than one day to converge. The reason for such disparity is the proximity of the initial centers to an optimum: a seed converges with fewer iterations when is closer to an optimum.

From our analysis, we found that the final quality of cluster analysis depended on the position of the initial centers provided to the cluster algorithm, not on the number of iterations of the algorithm. A drawback of our strategy is that when using a small *max_iter*, some K-means instances may not converge. However, the score of the non convergence instances are poor and they lose the competition. Since we run several K-means in parallel, the final result of cluster analysis will not be influenced by the losing instances. If the winning K-means does not converge, our strategy should be repeated with a larger *im* or with a larger *max_iter*. For better understanding the advantages of a low *max_iter*, consider the following two scenarios of the same cluster analysis problem. In scenario A, we run 100 K-means with *max_iter* = 100. In scenario B, we run 1,000 K-means with *max_iter* = 10. Suppose that the probability of a seed converges in less or 10 iterations is the same as the probability of converges in more than 10 iterations (0.5). The probability of converge in more than 100 iterations is 0.05. According to this probability distribution, we expect 95 K-means to converge in scenario A and 500 in scenario B. The scenario B has the advantage of early discarding slow convergence seeds. Thus more time becomes available for clustering seeds that converges fast. As this example shows, reducing *max_iter* and raising the number of K-means instances can increase the chance of finding a solution with a better local minimum per running time.

To improve initial seed selection, other techniques have been developed by several authors [4–13]. The K-means++ method from 2007 by Arthur and Vassilvitskii [7] provides a significant increase in K-means quality by choosing the initial cluster centers away from each other. In K-means++, initial centers from the dataset are chosen consecutively with probability proportional to the distance to the nearest center. However, K-means++ faces the drawback of running serial only, which slows the cluster analysis [14], [15]. Our new second strategy is a parallel implementation of the existing serial K-means++. Despite not being embarrassingly parallel, we found that K-means++ has the potential for parallelizing the most time-consuming steps of the algorithm. Our implementation considerably reduces the running time, while maintaining the same quality as the existing serial algorithm.

We compared the performance of our two new strategies with the existing non-parallel different implementations of the K-means method from Lloyd [16], McQueen [17] and Hartigan – Wong [18] all of them with Forgy seeding [19]. We found that compared with the approaches from these authors, our two new approaches, reduced the within sum of squares, thus improved the cluster quality. We compared our new parallel K-means++ with the existing Arthur and Vassilvitskii serial version. We found that our new K-means++ is twice as fast as the existing serial approach, yet maintains cluster quality. Our new Parallel K-Means++ has the advantage of not requiring additional parameters than K-means and maintaining the same straightforward analysis as the serial. Our parallel version shares all the disadvantages of the K-means++ except the long running time.

All the algorithms were used to analyze a real dataset of failures collected from a large supercomputing system. We aim to solve a similar problem as the one presented in [20] by Hacker *et al.* These authors suggest the usage of Neural Gas clustering instead of K-means for this type of problem. Their argumentation claims higher quality of Neural Gas towards K-means at the cost of longer computational time. Hacker *et al.* concludes that a major downside of their model to predict failures in large supercomputer systems is the time consumed by Neural Gas. Experiments presented in this paper show that our new parallel K-means++ produces a cluster with better quality and is four times faster than the Serial Neural Gas method.

II. THEORY AND BACKGROUND

Cluster analysis

Kaufman and Rousseeuw [21] define cluster analysis as the classification of similar objects into groups, where the number of groups, as well as their forms is unknown. The "form of a group" refers to the parameters of cluster; that is, to its cluster-specific means, variances, and covariance that also have a geometrical interpretation. A survey of applications can be found in Niknam [22].

The aim of clustering is to partition data points into clusters that are compact and well separated from each other. A common way to evaluate the quality of cluster analysis is to calculate the sum of the intra distances between the samples and their respective centroids. When using Euclidean distance, the quality can be measured with the total Within Sum of Squares (WSSQ). A lower value of WSSQ means better quality. More about cluster quality measures are presented in [23], [24].

Empty clusters can be obtained if no points are allocated to a cluster at the end of cluster analysis, resulting in centers that do not represent any data. Empty clusters are an undesired outcome when using cluster analysis for compression [2], since

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

the cluster prototypes will not represent any sample.

K-means method

K-means is a partitional clustering method that has been implemented in slightly different algorithms. Stuart Lloyd first proposed the standard algorithm in 1957, though it was not published until 1982 [16]. James MacQueen presented in 1967 [17] a K-means algorithm that recalculates the centroids after the assignment of each point to a cluster. This was an improvement over Lloyd's, which only recalculates after all the points have been assigned on each passage. Hartigan - Wong provided another famous K-means improvement in 1979 [18], which is a more complex algorithm. Telgarsky and Vattani [25] present a theoretical justification for its better convergence. The algorithms described in the paper by Lloyd, MacQueen, Hartigan and Wong are non-parallel.

The K-means method requires three parameters: the number of clusters (k), the distance metric, and the cluster initialization [2]. Steinley [3] presents a list of methods that can be used to aid the user with the number of clusters choice. Xing *et al*. [26] describe an algorithm that learns the distance metric given similar or dissimilar pairs from the dataset. Our work deals with the cluster initialization problem that is the choice of the cluster centers starting values. The choice of the number of clusters and the distance metric are problems out of scope for this paper.

The cost function of K-means converges to an optimum without a guarantee that it is the global one. As Maitra *et al*. [4] affirmed, there may often be a large number of local optima, with the specific number depending on the total number of data points (n), k and the original layout of the points. Thus, the choice of the starting values (or seeds) for the algorithm is critical, since different seeds can produce different local optima, leading to varying partitions. Therefore, good initialization is crucial for finding globally optimal partitioning. A survey of K-means initialization methods that are non-parallel can be found in [4–9], [27].

The Forgy serial initialization method was proposed in 1965 [19]. The approach selects k centers in a single iteration randomly. This approach takes advantage of the fact that a point in a higher density region is more likely to be selected. However, there is no guarantee that this method will not choose two seeds near the center of the same cluster, or a seed that is an outlier point. According to Redmon and Heneghan [5], repeated runs of this method are the standard method of initializing the K-means algorithm. Peña *et al.* [6] have empirically concluded that this initialization method does not give the best results, however it is fast and generates an initial partition independent of the instance order. As Redmon and Heneghan confirmed, the primary negative aspect of using repeated runs, as an initialization method is that the additional time needed to obtain a feasible solution. This time can be significantly reduced with our parallel initialization K-means algorithm with a small *max_iter* value.

K-means++ method

Arthur and Vassilvitskii [7] described serial K-means++, an algorithm to seed the initial cluster centers that improves clustering. The K-means++ selects the first centroid at random. The subsequent centroids are then selected according to the minimum probable distance that separates the centroids. The K-means++ uses k iterations and selects one point in each iteration according to a non-uniform distribution (that is constantly updated after each new center is selected) [14]. According to Brunsch and Röglin [27], K-means++ has $O(\log k)$ running time. Kumar and Wasan [28], Celebi [29] and Maitra *et al.* [4] described studies where K-means++ was compared with other initialization methods. There are few studies that investigate the parallelization of K-means++. Ailon *et al*. [10] and Ackermann *et al*. [11] introduced a streaming algorithm based on K-means++. Bahami *et al.* [14] presents a scalable K-means++. The major downside of Bahami *et al.* approach is the requirement of an extra input parameter l that is non intuitive and there are no described guidelines. As the authors admit, the analysis of the provable guarantees is highly non-trivial and requires new insights comparing to the analysis of K-means++. Our new parallel version of the K-Means++ has the advantage of not requiring additional parameters and maintaining the same straightforward analysis as the serial.

Neural Gas method

Martinetz and Schulten first presented the Neural Gas method in 1991 [30]. It is a competitive learning technique using the SoftMax learning rule based on Kohonen's self-organizing map. In this technique, multiple centroids get updated whenever the algorithm visits the data instance. The amount of update depends on the distance between the data instance and the cluster centroid. This method has a deterministic annealing mechanism built into the learning and optimization process to avoid many locally optimal solutions. Zhang *et al*. [31] compared this method with K-means and concluded that it has the benefit of arriving at more accurate clustering results at the expense of additional computational time to converge to a result. We did not find a study that compared the performance of Neural Gas method with K-means++. Details about the Neural Gas can be found in [30].

R and the DoMC package

R [32] by default is a serial program that uses only a single computational core. To extend *R* to use multiple cores, the *doMC* package can be used. The *doMC* provides a mechanism to execute loops in parallel using syntax similar to *R*. The user writes *R* code inside this special loop. The *doMC* automatically forks the loop into process that can be run in separated cores and processors.

R has a serial implementation of the K-means in the *stats* package. It calls compiled functions to *C* and to *FORTRAN* that are highly optimized. We found that the doMC package was an efficient means to run parallel instances of the existing optimized serial K-means algorithm.

III. ALGORITHMS

In this section we describe the existing serial K-means++ algorithm along with our two new strategies.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

Algorithm 1 shows the existing serial K-means++. Instead of presenting the original formulation by Arthur and Vassilvitskii [7], we describe a pseudo-code implementation to show the possibilities of parallelization. Step 2.1 in Algorithm 1 is a good candidate for parallelization. In our experiments this step consumed 97% of the total execution time, which indicates that parallelization can successfully reduce the overall running time of this algorithm. If we assume that the unparallizable fraction of the time consumed by the algorithm is s = 3%, the maximum possible speedup using Amdahl's law [33] is then a factor of 33.

Our first new strategy shown in Algorithm 2 uses the concept of competitive fitness to improve the initial seeding of the centroids for cluster analysis. The algorithm runs p parallel instances of K-means with Forgy's initialization method that is described in Section 2. The results of the K-means instances are evaluated using a fitness score that can be defined by the user. In this algorithm, the parallel instances compete with the fittest found during the course of the search. If a fitter instance is found it wins the competition. The winner becomes the fittest and the algorithm iterates again. After *im* iterations without improvement the algorithm stops.

The degree of parallelization p is the number of processor cores that will be used for the cluster analysis. F is the fitness score function used to measure the quality of the clustering of a K-means instance, e.g.: a quality measure such as WSSQ. The improvement factor *im* is a heuristic that defines the stopping condition of the algorithm. *im* represents the number of iterations that the algorithm should run without finding any fitter instance. A higher value increases the chance of fitness and the running time.

Input: A set of data points X; the number of clusters, k and the distance measure, *dm*
Output: X points grouped into k clusters and respective centroids
Let n be the number of points in X
1. Take one center c_1 chosen uniformly at random from X
2. For i in 1.. k-1 do:
  2.1. For j in 1.. n do:
    2.1.1. $d_{i,j} = dm(c_i, x_j)$
    2.1.2. $mds_j = [\min_{1 \le y \le i}(d_{y,j})]^2$
  2.2. End for
  2.3. Take c_{i+1}, choosing $x_j \in X$ with Probability
$$P(x_j) = \frac{mds_j}{\sum_{q=1}^{n} mds_q}$$
3. End for
4. K-means with C initial centers

Algorithm 1. Existing serial K-means++

Our new second strategy is shown in Algorithm 3. It is a modified version of the serial K-means++ algorithm described in Section II that seeks to reduce the seeding time. The algorithm starts by allocating the first centroid by randomly sampling the set of data points X. Then X is partitioned into p subparts X_l. The parallelization occurs in step 3.1. The *For* cycle processes in each l core a sub partition X_l. Step 3.1.1 calculates distances $d_{i,j}$ between the centroids that are yet allocated and the points in each X_l. Step 3.1.2 selects for each point the closest distance to the allocated centroids and squares it. Step 3.3 calculates the probability for each point to be selected and chooses the next centroid. This step is serial since $P(x_j)$ requires the mds_j from all cores. Step 3 stops when all centroids are sampled. Step 5 calls K-means algorithm with the centroids allocated in step 3.

Input: A set of data points X; the number of clusters, k; the degree of parallelization, p; the fitness score function, F and the improvement factor *im*.
Output: the result of the fittest K-means, *CL_fit*
1. *CL_fit* = K-means with random initial centers; *count* = 0
2. Calculate *score_fit*=F (*CL_fit*)
3. While *count* < *im* do:
  3.1. ||[1] (1..p) *CL* = K-means with random initial centers
  3.2. || (1..p) Calculate *score_cand* = F(*CL*)
  3.3. If *score_cand* is fitter *score_fit* Then:
    3.3.1. *CL_fit* =*CL*
    3.3.2. *count* = 0
  3.4. Else *count* = *count* + 1
  3.5. End If
4. End While

Algorithm 2. Our new Parallel Competitive Fitness K-means seeding

Input: A set of data points X; the number of clusters, k; the degree of parallelization, p and the distance measure, *dm*
Output: X points grouped into k clusters and respective centroids
Let n be the number of points in X and m the number of points in each X_l sub partition of X
1. Take one center c_1 chosen uniformly at random from X
2. Partition X into $\bigcup_{l=1}^{p} X_l$
3. For i in 1.. k-1 do:
  3.1. For j in 1.. m do: (in parallel to $X_1, .., X_p$)
    3.1.1. $d_{i,j} = dm(c_i, x_j)$
    3.1.2. $mds_j = [\min_{1 \le y \le i}(d_{y,j})]^2$
  3.2. End for
  3.3. Take c_{i+1}, choosing $x_j \in X$ with Probability
$$P(x_j) = \frac{mds_j}{\sum_{q=1}^{n} mds_q}$$
4. End for
5. K-means with C initial centers

Algorithm 3. Our new Parallel K-means++

IV. Experimental Setup

In this section, we demonstrate the significant improvements in processing time and cluster quality possible through the use of our two new strategies described in Section III. We show this by testing the following hypotheses:

H1- The Hartigan - Wong (H.W.) serial implementation of the K-means method is faster and has lower WSSQ than

[1] Symbol || means: running in parallel.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

serial implementations of Lloyd and McQueen.

H2- Our new Parallel Competitive Fitness K-means seeding reduces the WSSQ, improving cluster analysis quality compared with a single instance of a serial K-means with a higher *max_iter*,

H3- Our new Parallel K-means++ reduces the execution time while simultaneously maintains the cluster analysis quality when compared with the existing serial K-means++.

H4- Our new Parallel K-means++ has lower WSSQ and is simultaneously faster than Neural Gas.

We formulated the first hypothesis to test if H.W. is the best K-means serial implementation to be used in *H2*. To test our hypothesis we conducted the following experiments:

A. Clustering using Lloyd, McQueen, and Hartigan - Wong K-means implementations using Forgy as initialization method. The parameter *max_iter* was 150.
B. Clustering using our Parallel Competitive Fitness with the best K-means implementation found in experiment A. The improvement factor *im* was set to be 100. This means that the algorithm stops after running at least 100 times without WSSQ improvement. Each instance of K-means had *max_iter* of 10.
C. Clustering using our Parallel version of the K-means++ algorithm.
D. Clustering using the existing serial version of the K-means++.
E. Clustering using a Serial Neural Gas. It was tested with the number of iterations from 10 up to 100.

Real-World Application

Our problem is a typical use of cluster analysis for data compression. We aim to reduce the spatial temporal dependency by compressing the data into cluster prototypes and use the prototypes as data representatives to further processing. The problem is described by Hacker *et al.* in [20].

We needed to cluster 5 datasets. Each one has 10,000 points and 2 dimensions. We expected to have more than 2,000 clusters and the need to repeat the clustering process with several different numbers of centers. This is a computationally intensive task that has to be accomplished quickly and using modest resources.

The experiments

For all the experiments, we measured the execution times and WSSQ. We normalized the data and used Euclidean distance. The experiments A and B were repeated 20,000. The experiments C and D were repeated 100. Due to the deterministic nature of Neural Gas each experiment was repeated only 5 times. The experiments were repeated for 2,000, 5,000 and 10,000 centroids for all datasets.

We ran our experiments on a PC with 16GB 1333MHZ and 1 CPU AMD Phenom™ II X6, 6 cores 3300 MHz. We used *R* for the cluster analysis. The *R* version was 2.11.1 (2010-05-31). We used the *R* packages: *doMC* for the parallelization; *cclust* for the serial Neural Gas method; *pracma* for the serial K-means++ and *stats* for the serial implementations of K-means. For our parallel analyses, we limited the number of parallel tasks to six, matching the number of cores on our system. If we increased the degree of parallelization to more cores, we expect that our new approaches would take less time.

V. Experimental Results

We only included results from one dataset and with a *k* value of 2,000 centroids due to space limitations. The results for the other 4 datasets and for the different values of *k* values (5,000 and 10,000 centroids) not included in this section were similar.

Experiment A

The left graph in Figure 1 shows that the H.W. implementation has lower WSSQ than Lloyd and McQueen. However, 5% of the H.W. experiments did not converge within *max_iter* = 150. The Lloyd and McQueen implementations converged in all the experiments. We observed that Lloyd generated empty clusters. The WSSQ dispersion of H.W. is the narrowest of the 3. Still, the interval between the maximum and minimum value is 0.23, indicating that the cluster analysis can produce results with significant different quality.

The right graph in Figure 1 shows the execution time of the 3 K-means implementations. The H.W. implementation has wider execution time dispersion. The minimum and maximum values shows that H.W. can be the fastest and the slowest K-means implementation. The implementation with the lower median is the McQueen.

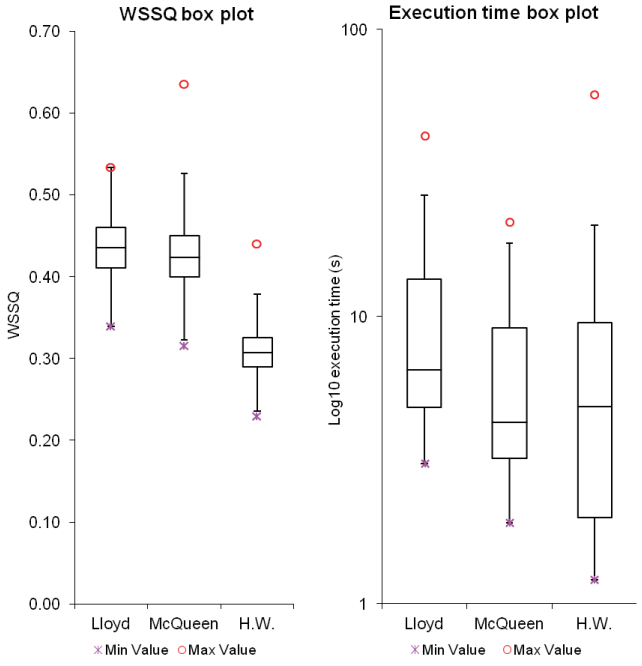

Figure 1. (Left) WSSQ of three serial K-means implementations, lower WSSQ means more accuracy. (Right) Cluster analysis run time in seconds.

The histogram in Figure 2 shows the probability density of the number of iterations necessary for H.W. to converge. In our cluster analysis, the probability density is concentrated in lower iterations, thus suggesting a benefit from a small *max_iter*.

From the three algorithms assessed in *H1*, Hartigan - Wong proved to be the one that has better WSSQ. Hence H.W. is the

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

K-means implementation choice for our two new approaches.

Experiment B

Figure 3 compares the results obtained from single instances of H.W. and from our new Parallel Competitive Fitness (PCF). Both approaches use Forgy initialization method and the same K-means implementation (H.W.). To test our *H2* we reduced the *max_iter* from 150 to 10 in the PCF strategy and an improvement factor *im* of 100.

The left side of Figure 3 shows that our new PCF narrows the total WSSQ dispersion comparing to the existing H.W.. Our approach decreased the maximum value of WSSQ by a ratio of 1.67, meaning that the worst case is significantly better than the worst case of the existing H.W. The median of our approach decreased the median of the existing H.W. by a ratio of 1.23, indicating that the PCF increases the chances of a better WSSQ. Figure 3 right side compares the execution times of single instances of H.W. with our new PCF. Since PCF run at least *im* instances of H.W. we expect the execution time to increase. The median value of PCF corresponds to a situation where the PCF converged in 169 instances and consumes 14.21 times more time than the serial H.W. With a degree of parallelization of 6, the PCF run 6 instances simultaneously and each instance is faster since the *max_iter* has been decreased.

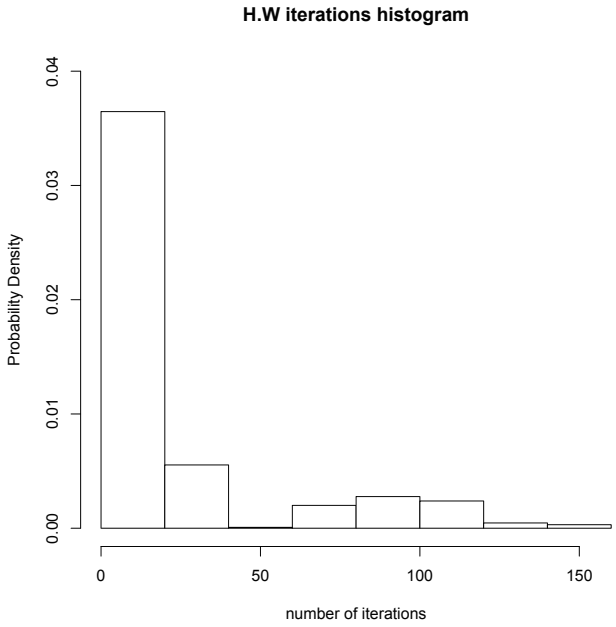

Figure 2. H.W. iterations histogram with *max_iter*= 150

Experiment C

The left side of Figure 4 shows that our new Parallel K-means++ dramatically improves the cluster analysis quality compared with our new PCF and, subsequently with existing serial Lloyd, McQueen and H.W. The dispersion of WSSQ is significantly lower when compared to the PCF. It is remarkable that we always obtained better results with Parallel K-means++ in all our experiments. The worst result of Parallel K-means++ has a WSSQ of 0.0014, while the best result of the PCF has a WSSQ of 0.22.

We improved the cluster quality at the cost of significantly increasing execution time compared with existing methods (Figure 4 right side). Despite the fact that K-means itself converges with fewer iterations, the whole method becomes slower due to the amount of time in the seeds selection.

Experiment D

The cluster analysis of the existing serial K-means++ has the same quality of our new parallel K-means++, thus the WSSQ is the same has presented in Experience C.

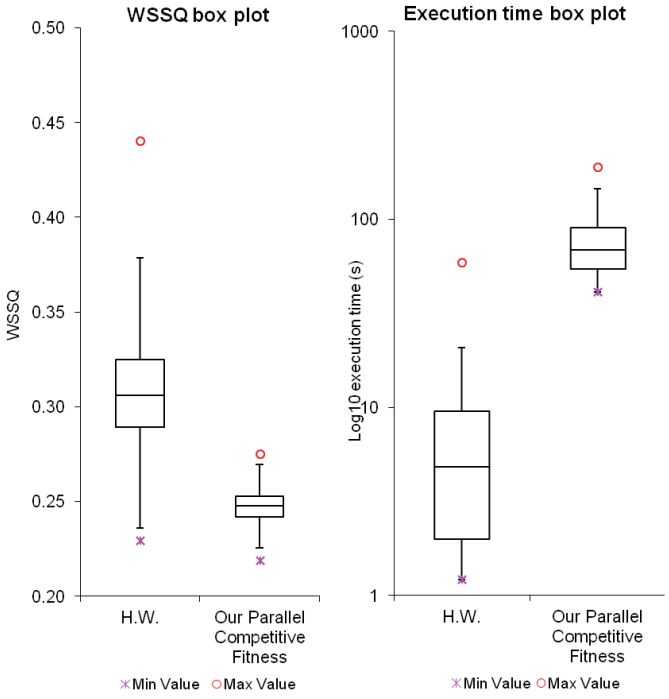

Figure 3. (Left) Total WSSQ of H.W. with *max_iter*=150 and our Parallel Competitive Fitness with *max_iter*=10; *im*=100; *p*=6. Lower WSSQ means more accuracy. Values between 0 and 1. (Right) Execution time in seconds.

Figure 5 shows the execution time of the total algorithm and the execution time of the steps that can be parallelized is almost the same. This means that by parallelizing the algorithm the most relevant time consuming steps can be reduced. With our implementation using R and the DoMC package the processing time was reduce more than a half.

Experiment E

Compared with our new Parallel K-means++, the Neural Gas (NG) method performed very poorly. Table I shows that the execution time of NG is almost 4 times slower than our new parallel K-means++. The WSSQ of NG is 0.028, which is considerably worse than the value of 0.001 obtained by our Parallel K-means++. Compared with our new PCF the NG improves the WSSQ. However, both our new approaches did not generate empty clusters, while NG generates 420 empty clusters out of 2,000. Empty cluster is an undesired outcome of cluster analysis when used for data compression. Related to execution time, NG takes 280 times more than our PCF.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

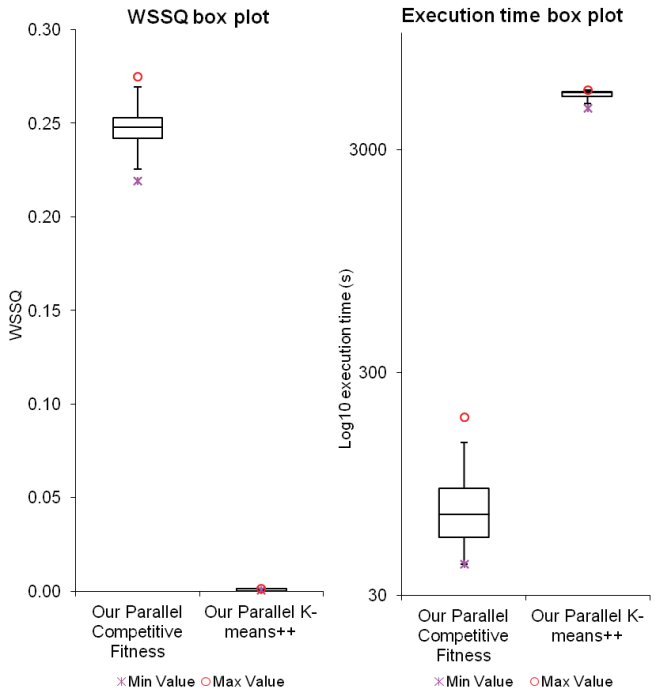

Figure 4. (Left) Total WSSQ of our Parallel Competitive Fitness and our Parallel K-means++. Lower WSSQ means more accuracy. Values between 0 and 1. (Right) Execution time in seconds.

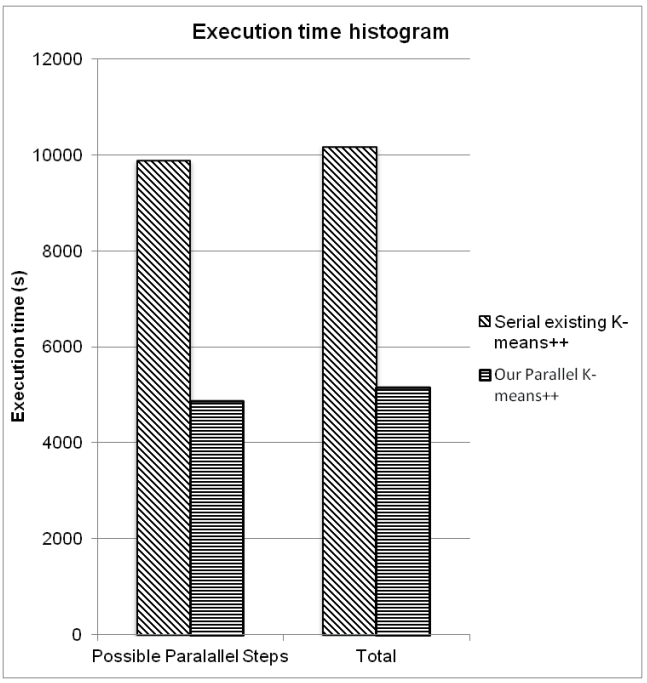

Figure 5. Comparison between the serial and Parallel K-means++

TABLE I. PERFORMANCE OF NEURAL GAS.TIME IN SECOND. WSSQ BETWEEN 0 AND 1.

	Neural Gas	Parallel CF K-means	Parallel K-means ++
Time	21,266	76	5,290
WSSQ	0.028	0.250	0.001
Convergence (%)	100.00	100.00	100.00
Empty Clusters	420	0	0

VI. DISCUSSION OF RESULTS

In this section, we discuss the impact of the results on our Hypothesis. The first hypothesis was proved to be true in what WSSQ concerns. Hartigan - Wong is the serial implementation with lower WSSQ in average compared with Lloyd and McQueen. However, the first hypothesis was not proved to be true in what speed concerns. The results proved that McQueen is the fastest of the three implementations in average. However, H.W. has the minimum and maximum execution times to converge, which indicates that the time for H.W. to converge depends more on the quality of the initial seeds.

The second hypothesis was proved to be true. Our new Parallel Competitive Fitness reduced the WSSQ compared with a single instance of a serial K-means with a higher *max_iter*.

The third hypothesis was proved to be true. Our new Parallel K-means++ reduces the execution time while simultaneously maintains the cluster analysis quality when compared with the existing serial K-means++. Our new parallel implementation and the existing serial K-means++ have lower WSSQ compared with the existing serial implementations of H.W., Lloyd, McQueen suing Forgy initialization method. Our new Parallel K-means++ is twice as fast as the serial method

The fourth hypothesis was proved to be true. K-means++ has lower WSSQ than Neural Gas method. Our new Parallel K-means++ was four times faster than the Neural Gas method.

VII. CONCLUSIONS

K-means is a fast clustering method. However, initial seeding heavily influences the cluster quality. Arthur and Vassilvitskii proposed a seeding improvement that dramatically reduces the clustering speed.

In this paper, we presented two new strategies to parallelize the seeding of K-means. Our new Parallel Competitive Fitness runs parallel instances of K-means until a heuristic performance factor *im* is met. We conclude that it is a fast way to improve the cluster quality and to increase the probability of converging to a better cluster.

Our other new strategy is Parallel K-means++. It is much slower than our Parallel Competitive Fitness, but it produces considerably better results. The execution time of our Parallel K-means++ is 1.97 times faster than the current serial method.

Using our parallel version of K-means++, it is possible to perform cluster analysis faster and with more accuracy than more complex techniques like Neural Gas. With our new approaches, the user can perform cluster analysis with more quality and in shorter time. This is particularly important in problems in which the dataset is frequently updated and the cluster analysis has to be performed quickly, such as in the

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

analysis of clustered failures on large supercomputing systems. With our findings, the major downside of finding a proper seeding in useful time has been improved. Thus, predicting the amount of resources needed to provision Virtual Machines in a cloud computing systems to support virtual HPC clusters can now be done faster and with more accuracy.

The strategies were implemented using R and the DoMC package. This package is easy to use but limits the parallelizing to one single machine. Yet, we expect more gains in performance using a distributed technology. Another present limitation of our implementation is that the totality of the dataset has to be in memory. It would be interesting to study in the future a map reduce implementation.

REFERENCES

[1] T. Hacker and K. Mahadik, "Flexible Resource Allocation for Reliable Virtual Cluster Computing," in *Supercomputing 2011 (SC11), Proceedings of Supercomputing 2011 (SC11)* , November 12-18, 2011 Seattle, WA.

[2] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[3] D. Steinley, "K-means clustering: A half-century synthesis," *British Journal of Mathematical and Statistical Psychology*, vol. 59, no. 1, pp. 1–34, 2006.

[4] A. D. Peterson, A. P. Ghosh, and R. Maitra, "A systematic evaluation of different methods for initializing the K -means clustering algorithm," *Knowledge Creation Diffusion Utilization*, pp. 1–11, 2010.

[5] S. J. Redmond and C. Heneghan, "A method for initialising the K-means clustering algorithm using kd-trees," *Pattern Recognition Letters*, vol. 28, no. 8, pp. 965–973, Jun. 2007.

[6] J. . Peña, J. . Lozano, and P. Larra√±aga, "An empirical comparison of four initialization methods for the K-Means algorithm," *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1027–1040, Oct. 1999.

[7] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.

[8] Qinchao Liu, Bangzuo Zhang, Haichao Sun, Yu Guan, and Lei Zhao, "A Novel K-Means Clustering Algorithm Based on Positive Examples and Careful Seeding," in *Computational and Information Sciences (ICCIS), 2010 International Conference on*, 2010, pp. 767–770.

[9] T. Onoda, M. Sakai, and S. Yamada, "Careful Seeding Based ßßon Independent Component Analysis for k-Means Clustering," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, 2010, vol. 3, pp. 112–115.

[10] N. Ailon, R. Jaiswal, and C. Monteleoni, "Streaming k-means approximation," 2009.

[11] M. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot, "StreamKM++: A Clustering Algorithm for Data Streams," *ALENEX*, pp. 173–187, 2010.

[12] P. S. Bradley and U. M. Fayyad, "Refining Initial Points for K-Means Clustering," in *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 91–99.

[13] B. Mirkin, *Clustering for data mining: A data recovery approach*. London: Chapman and Hall, 2005.

[14] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proc. VLDB Endow.*, vol. 5, no. 7, pp. 622–633, 2012.ßß

[15] T. Li, S. Bai, and J. Ning, "K-means#, an applicable and efficient clustering algorithm," *Energy Procedia*, vol. 13, no. 0, pp. 3189–3196, 2011.

[16] S. Lloyd, "Least Squares Quantization in PCM," *IEEE Trans. Information Theory*, 1982.

[17] J. B. MacQueen, "Some Methods for Classification and Analysis of MultiVariate Observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, vol. 1, pp. 281–297.

[18] J. Hartigan and M. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[19] E. Forgy, "Cluster analysis of multivariate data: efficiency vs. interpretability of classifications," *Biometrics*, vol. 21, pp. 768–769, 1965.

[20] T. J. Hacker, F. Romero, and C. D. Carothers, "An analysis of clustered failures on large supercomputing systems," *J. Parallel Distrib. Comput.*, vol. 69, no. 7, pp. 652–665, 2009.

[21] L. Kaufman and P. J. Rousseeuw, *Finding groups in data. An introduction to cluster analysis.*, 99th ed. Wiley-Interscience, 1990.

[22] T. Niknam, E. Taherian Fard, N. Pourjafarian, and A. Rousta, "An efficient hybrid algorithm based on modified imperialist competitive algorithm and K-means for data clustering," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 2, pp. 306–317, Mar. 2011.

[23] Q. H. Nguyen and V. J. Rayward, Smith, "Internal quality measures for clustering in metric spaces," *Int. J. Bus. Intell. Data Min.*, vol. 3, no. 1, pp. 4–29, 2008.

[24] B. Raskutti and C. Leckie, "An Evaluation of Criteria for Measuring the Quality of Clusters," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999, pp. 905–910.

[25] M. Telgarsky and A. Vattani, "Hartigan's Method: k-means Clustering without Voronoi.," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 820–827, 2010.

[26] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance Metric Learning, with Application to Clustering with Side-information," in *Advances in Neural Information Processing Systems 15*, 2002, pp. 505–512.

[27] K. K. Pavan, A. V. D. Rao, and G. R. Sridhar, *Single Pass Seed Selection Algorithm for k-Means 1.* .

[28] P. Kumar and S. K. Wasan, "Comparative Analysis of k-mean Based Algorithms," *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 4 April, pp. 314–318, 2010.

[29] M. E. Celebi, "Effective initialization of k-means for color quantization," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, 2009, pp. 1649–1652.

[30] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, "`Neural-ßßgas' network for vector quantization and its application to time-series prediction," *Neural Networks, IEEE Transactions on*, vol. 4, no. 4, pp. 558–569, 1993.

[31] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," *Intelligent Systems, IEEE*, vol. 19, no. 2, pp. 20– 27, 2004.

[32] "The Comprehensive R Archive Network." [Online]. Available: http://cran.r-project.org/. [Accessed: 10-May-2012].

[33] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, Atlantic City, New Jersey, 1967, pp. 483–485.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE