Towards an Efficient Storage Workload Modeling with Time-Series Clustering Parallelization

Edwin F. Boza*, Gladys E. Carrillo†, Cristina L. Abad‡, Sixto Garcia§
Escuela Superior Politecnica del Litoral, ESPOL
*eboza@fiec.espol.edu.ec, †gladys.carrillo@cti.espol.edu.ec, ‡cabad@fiec.espol.edu.ec, §sgarcia@cti.espol.edu.ec

***Abstract*—Evaluation of distributed storage systems requires workloads modeled from real systems. Modeling a workload from a trace file, can be a resource expensive task for todays' datasets; and resulting models can be large enough to negatively affect the operation of the evaluation tool. Clustering algorithms are useful to reduce the models, but their time-complexity limits their application to some domains, like time-series. In this work we propose a model to use parallelization and probabilistic sampling, to reduce the space and time requirements for clustering processes, while keeping a moderate level of accuracy.**

I. Introduction

Distributed storage systems are a key part of current cloud and Big Data applications. Google File System, Cassandra, DynamoDB, Ceph are a few examples of alternatives currently available in the market, each of them with particularities that allow their use in specific scenarios.

Evaluation of storage systems, in order to compare products or evaluate improvements on a specific product, requires the use of workloads modeled from real systems [1]. To perform the modeling, a workload trace is required as an input; and the modeling process consists in abstracting essential characteristics of the objects observed in the trace, to allow the synthetic production of a trace with similar characteristics and dimensions.

In the case of storage systems, the workload trace contains the sequence of accesses to the stored objects, for a certain period of time. Traditional modeling techniques suggest the creation of a statistical summary applied to all the observed workload [2], or to identify the patterns of access to each object and extract a model which allows reproducing this pattern [3]. The former approach will vanish distinct per-object behavior, while the later requires an excessive amount of resources (memory and processing) and can affect the operation of the evaluation tool. To address this problem, automated learning algorithms can be used to group similar objects into a reduced number of classes. These new models will be used for the automatic generation of access streams that can be used at the time of performance evaluation.

Several approaches can be used for clustering similar objects based on their access patterns. For example, in a previous work [4], a set of statistical summary measures were used as descriptors of the interarrival sequences of each object (Fig. 1). With this set of measures, a classification process was performed using the K-means algorithm [5].

A new approach consists in considering the interarrival times of each object, as a time series, and find groups or clusters that are meaningful and useful. However, while several techniques and tools for grouping time series exist, and they are even grouped in packages for popular programming language [6], their performance is seriously affected with large datasets.

In this work, we propose the use of the Apache Spark [7] framework to speedup the clustering process of time-series, by parallelizing the construction of the corresponding distance matrix. Additionally, we explore the use of a probabilistic sampling [8] method to reduce the space and time requirements for the clustering process.

We show that the use of a naive parallelization of the time-series clustering method leads to a considerable reduction of the required time, however it still is not suitable to be used with a trace with millions of objects. Therefore, we show that using the probabilistic sampling method, in combination with the aforementioned parallelization allows to accomplish the clustering task in affordable time, while keeping an accuracy of up to 65% points.

II. Motivation and background

A. Apache Spark

Apache Spark is an open source framework aimed for building fast applications for large-scale data processing. Apache Spark addresses several of the inherent limitations of Hadoop and MapReduce related to the linear dataflow structure on distributed programs. Spark extends the MapReduce model to make it faster and enable more analysis scenarios, such as interactive queries and real-time stream processing. This is possible because Spark uses a cluster of in-memory computing and includes also a new storage primitive: resilient distributed datasets(RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost [7]. It lets the users cache precomputed data in memory and/or disk across queries. Spark is developed in Scala and can provide API support for Java, Scala and Python, which makes it easy to build parallel applications.

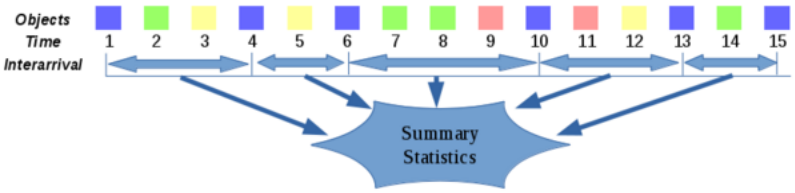

Figure 1. After obtaining the interaction times for each object, a set of statistical summary measures are calculated and used to feed a classification algorithm (K-means).

B. Workload modeling

To perform the modeling, a workload trace is required as an input. A workload trace is a collection of records of the accesses and operations that have been performed on a production system. Traces are generally obtained by monitoring mechanisms of a system in production.

The modeling process consists in abstracting essential characteristics of objects in the available trace; those characteristics must enable the synthetic reproduction of the workload described by the trace.

C. Probabilistic sampling methods

Other approaches of workload modeling for storage systems propose the use of probabilistic methods to reduce the space and time requirements needed to obtain the model. Wires et al. [9] propose a novel data structure, the counter stack, to approximate Miss Ratio Curves (MRCs) to produce workload models with smaller computational and memory overheads. The results of their experiments with large workloads show a 70% reduction in memory and storage usage and the resulting MRCs with an error rate between 0.002 and 0.02.

Waldspurger et al. [10] introduce SHARDS (Spatially Hashed Approximate Reuse Distance Sampling), an algorithm to calculate approximate Miss Ratio Curves using a hash based spatial sampling technique. The evaluation of the algorithm showed that MRCs can be constructed online using less than 10MB of memory per workload. Also, the error between approximate and exact MRCs is very low, between 0.007 and 0.02.

D. Time-series clustering

Time series are used in different areas of study such as finance, meteorology, agriculture, etc. And there are several techniques and tools for grouping them through unsupervised learning. For example, the TSclust package, of the programming language R [11].

The process of grouping time series consists of two separate stages or subprocesses. First, a distance matrix must be calculated using a measure of dissimilarity. And second, using the distances matrix, we proceed to apply a clustering algorithm.

The TSclust package, while useful since it implements a variety of dissimilarity measures and clustering algorithms for time series, has certain limitations when it comes to processing large or particular datasets.

For example, the current implementation of the distance matrix calculation, although possibly optimized, requires time series of the same size. This requirement is not met with access to storage systems, since due to the popularity of objects, some of them have a large number of accesses, in contrast to a majority group that has few or only access along the trace

This situation forces the manual calculation of the distance matrix, because there are measures of dissimilarity that operate on series of different size. However, this process is extremely slow. In an exploratory test, the manual distance matrix calculation, for a subset of 1,000 objects, took more than 1 day to complete using a desktop PC.

III. Model

Our proposed model comprises 4 steps, as described in Algorithm 1:

Algorithm 1 Steps for the proposed model

1: Extract the 10k more accessed objects.
2: Get a representative sample of the more accessed objects.
3: Build the distance matrix for the time-series corresponding to the interarrival times of sampled objects
4: Run the clustering process with the distance matrix obtained in phase 3

We start by extracting the subset of the more accessed objects, because they will represent the largest part of the workload. Also, as we show in the section IV-E, accuracy will decrease when clustering is applied to time-series of very different sizes.

In the next step, a randomly-selected representative sample of the more accessed objects should be obtained in order to reduce the required resources, while keeping a high level of accuracy in the clustering process.

For the third phase, the expected distance matrix is a symmetric matrix containing the measure of dissimilarity of time series between each pair of objects. The distance matrix calculation is an “embarrassingly parallel” process, therefore it is only necessary to calculate the distance between unique

pairs of objects. To speedup this process, we proposed a parallelized Algorithm 2 to be used with Apache Spark.

Algorithm 2 Parallel calculation of distance matrix

```
1: procedure CALCULATEDISTANCEMATRIX
2:     Read object list as RDD
3:     Combine objects to get a paired lists as RDD.
4:     Calculate distance between time series of each pair
   of objects in parallel.
5:     Complete distance matrix with the calculated values.
6: end procedure
```

In the last step, for the clustering process, we propose the use of the K-medoids clustering algorithm [12]. This algorithm is less sensitive to noise and outliers than the more commonly used K-means; and for the case of time-series, K-medoids is more suitable because it always select objects from the sample as centroids for the clusters, instead of *calculate/create* a new centroid, which is not a reliable action with time series.

IV. Experimental evaluation

A. Dataset

The dataset used in the experiments consists of a trace of accesses to a storage system (HDFS) of Yahoo Inc. company. The trace contains $54{,}600{,}645$ accesses to $4{,}988{,}059$ unique objects. A preprocessing step was performed to obtain the interarrival times for each object.

B. Environment

Two servers where used to test the parallel version of the distance matrix calculation. Table I shows the characteristics of the servers. Server 1 has three times more CPU cores than Server 2; however, the later has three times more RAM than the former.

Table I
Characteristics of the servers used in the experiments

	RAM	CPU Model	Number of cores
Server 1	32GB	Intel Xeon 2.30GHz	72
Server 2	96GB	Intel Xeon 2.50GHz	24

The model was implemented in Spark 2.1.0 using Python API version 2.7. The measure of dissimilarity used for the distance matrix calculation is a Python implementation of FastDTW [13]. This improved version of Dynamic Time Warping (DTW) [14] allows to measure similarity between two sequences like time series with an O(N) time and memory complexity. Clustering was performed using the Python implementation of K-medoids implemented by Bauckhage [12].

C. Speedup in distance matrix calculation

The goal of our first experiment was to show the speedup obtained from the parallelization of the distance matrix calculation process. Table II lists the times required to calculate the distance matrix for a random set of 1,000 objects, using the FastDTW distance. Server 1, which has more CPU cores, obtain the best results for both, sequential and parallelized, methods. It also shows a *speedup* of 24.45, which is largest than the speedup reached by the Server 2 with the parallelized method.

Table II
Execution time for distance matrix calculation for 1000 objects

	Sequential	Spark	Speedup
Server 1	28.82 horas	1.05 horas	24.45
Server 2	64.91 horas	4.18 horas	15.52

Even though the Server 1 has three times more CPU cores, the speedup reached by this server is roughly 1.77 times the speedup obtained for the Server 2. This can be explained by the higher amount of memory available in the second server. However, it is clear that CPU cores will be the main factor in the acceleration.

D. Limitation of the naive parallelization

We consider our parallelization method *naive*, because we have only focused on the optimization of the matrix calculation, which is a "perfectly parallel problem". However, the functions to calculate distances or similarity measures for time-series are not parallelized and have higher big O time complexities.

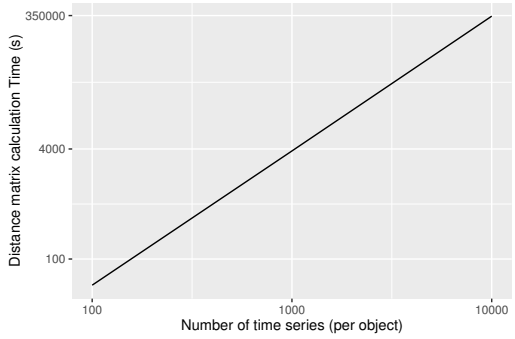

Figure 2. Execution time to calculate distance matrix using Spark

Figure 2, shows the evolution of the execution time for the parallelized calculation of a distance matrix, with an increasing number of time series. Log scales indicates an exponential growth in the observed execution time, as the number of time series increases. The execution time for a sample of 100 objects was 41.64 seconds, for 1,000 objects was 3,743.95 seconds and for 10,000 objects was 343,140 seconds.

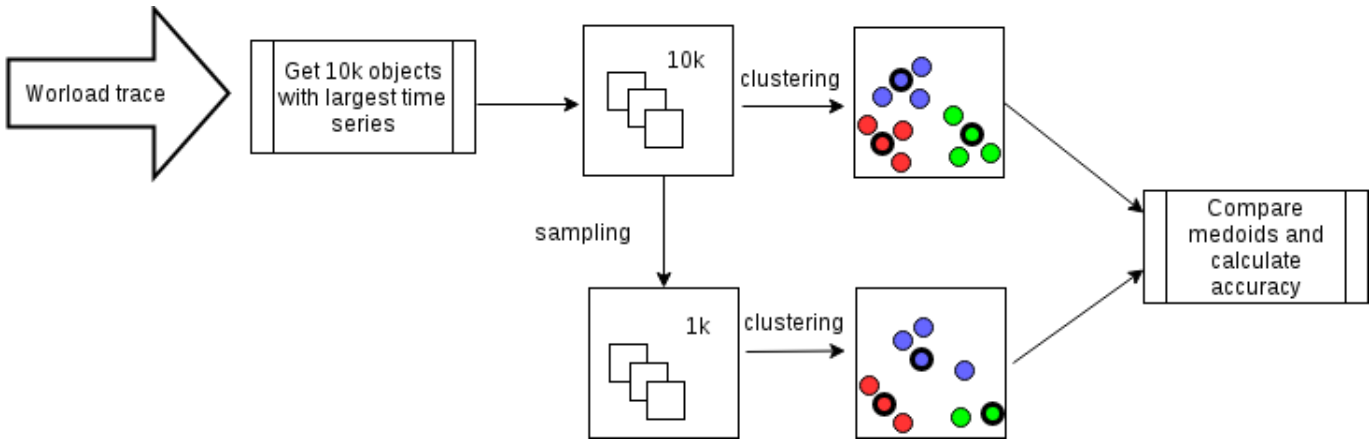

Figure 3. Experimental design for accuracy testing of the proposed model

This behavior compromises the use of this naive parallelization approach as the unique method to speed up the clustering of time series, and supports our decision to explore other techniques to complete the clustering in an affordable time.

E. Accuracy of the probabilistic model

To evaluate the accuracy of our proposed model (Section III), we designed and performed the experiment showed in Figure 3.

In the first step, a subset of 10,000 objects where selected, corresponding to the most accessed objects. Most-popular objects, corresponding to largest time-series, are selected because clustering accuracy decreases when the size range of time-series is too widespread, as shown in Figure 4

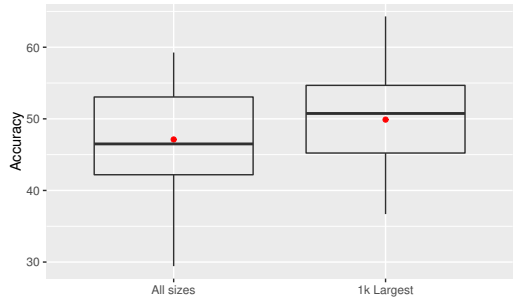

Figure 4. Comparison of clustering accuracy for 1k objects of wide-range sized time-series vs largest time-series. Accuracy increases when sizes of time series are similar.

This new dataset was used to obtain the distance matrix based on the design explained in section III. Using the distances matrix, a clustering process was performed with K-medoids algorithm to get 10 clusters.

To evaluate the probabilistic sampling method we took 1000 random objects from the dataset of the 10k largest objects; and computed the distance matrix of the new sample. This new matrix was used for a new clustering process to obtain 10 clusters.

In the final step, the accuracy is measured by comparing the efficiency of the medoids obtained in the clustering of the sampled subset, to reproduce the same clustering obtained in the clustering of the full 10k largest time-series. The Figure 5, shows the accuracy results obtained with our experimental method; the median accuracy is 57.1%, and the maximun accuracy is 65.34%.

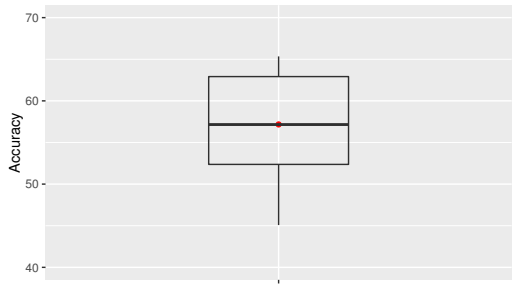

Figure 5. Clustering accuracy: Median = 57.1%; Max = 65.34%

V. Related work

The high-throughput molecular data makes clustering tasks complex due to poor performance of the correlation matrix calculation. In the face of this problem Wang et al. [15] have proposed a new approach for calculating correlation applying an effective algorithm based on MapReduce to optimize storage and correlation calculation. The approach was implemented with RHIPE, a Hadoop MapReduce based R package that transforms R functions into MapReduce jobs. The algorithm divide the input vectors, which contain intensity values for a single subject, into data blocks that are dispatched to different Mappers. Then each data block

is copied to a Reducer that calculates correlation values. As result, this algorithm performs 30 times faster than the normal R implementation.

In the maritime field, Liu [16] proposed a parallel version of a clustering algorithm for maritime traffic patterns extraction from Automatic Identification System Marine Traffic data. The parallel algorithm was designed and implemented using Apache Spark. The parallel algorithm was designed to convert the problem to a graph-based one to apply also the GraphX API. Although the proposed algorithm didn't have a satisfying performance result, it serves as a starting point to look for other methods that allow to reduce the complexity of the algorithm.

VI. Conclusions

In this work, we proposed a combined approach to speedup the clustering process of time-series.

We have shown that the use of the Apache Spark [7] framework, to parallelize the construction of a distance matrix, leads to a speed-up as large as 24, compared with the corresponding sequential algorithm.

Additionally, our experiments demonstrate that it is viable to use a probabilistic sampling method to reduce the space and time requirements for the clustering process, while keeping an accuracy of up to 65% points.

References

[1] A. Traeger, E. Zadok, N. Joukov, and C. Wright, "A nine year study of file system and storage benchmarking," *ACM Transactions on Storage (TOS)*, vol. 4, no. 2, 2008.

[2] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[3] P. P. Ware, T. W. Page Jr, and B. L. Nelson, "Automatic modeling of file system workloads using two-level arrival processes," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 3, pp. 305–330, 1998.

[4] C. Abad, M. Yuan, C. Cai, Y. Lu, N. Roberts, and R. Campbell, "Generating request streams on Big Data using clustered renewal processes," *Performance Evaluation*, vol. 70, no. 10, 2013.

[5] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[6] P. Montero, J. A. Vilar *et al.*, "Tsclust: An r package for time series clustering," *Journal of*, 2014.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[8] B. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 5: populations and samples," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 5, pp. 17–20, 2002.

[9] J. Wires, S. Ingram, Z. Drudi, N. J. Harvey, A. Warfield, and C. Data, "Characterizing storage workloads with counter stacks." in *OSDI*, 2014, pp. 335–349.

[10] C. A. Waldspurger, N. Park, A. T. Garthwaite, and I. Ahmad, "Efficient mrc construction with shards." 2015.

[11] H. Fritz, L. A. Garcıa-Escudero, and A. Mayo-Iscar, "tclust: An r package for a trimming approach to cluster analysis," *Journal of Statistical Software*, vol. 47, no. 12, pp. 1–26, 2012.

[12] C. Bauckhage, "Numpy/scipy recipes for data science: k-medoids clustering," *researchgate. net, Feb*, 2015.

[13] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[14] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[15] S. Wang, I. Pandis, D. Johnson, I. Emam, F. Guitton, A. Oehmichen, and Y. Guo, "Optimising parallel r correlation matrix calculations on gene expression data using mapreduce," 2014.

[16] B. Liu, "Parallel maritime traffic clustering based on apache spark."