12th IEEE International Conference
www.aict.info/2018

AICT 2018
INTERNATIONAL CONFERENCE

APPLICATION
OF INFORMATION
AND COMMUNICATION
TECHNOLOGIES - AICT2018

17-19 October 2018, Almaty, Kazakhstan

CONFERENCE PROCEEDINGS

17-19 October 2018, Almaty, Kazakhstan
www.aict.info/2018

12th IEEE International Conference

www.aict.info/2018

AICT 2018 INTERNATIONAL CONFERENCE

APPLICATION OF INFORMATION AND COMMUNICATION TECHNOLOGIES - AICT2018

17-19 October 2018, Almaty, Kazakhstan

CONFERENCE PROCEEDINGS

IEEE Catalog Number: CFP1856H-PRT
ISBN: 978-1-5386-6467-4

Additional copies of this publication are available from
Curran Associates, Inc.
57 Morehouse Lane
Red Hook, NY 12571 USA
+1 845 758 0400
+1 845 758 2633 (FAX)
email: curran@proceedings.com

© Designed by the AICT2018 Publication Team, 2018.

IEEE Catalog Number: CFP1856H-PRT
ISBN: 978-1-5386-6467-4

Design of K-Means Clustering Algorithm in PGAS based Mapreduce Framework

Shomanov A.S.
National Laboratory Astana
Kabanbay Batyr Avenue, 53, Astana, Kazakhstan
Tel.: 87016947795
adai.shomanov@nu.edu.kz

Mansurova M.E.
Al-Farabi Kazakh National University
Al-Farabi Avenue, 71, Almaty, Kazakhstan
Tel.:87014151960
mansurova.madina@gmail.com

Nugumanova A.B.
Sarsen Amanzholov East Kazakhstan State University,
30th Guards Division str., 34, Ust-Kamenogorsk, Kazakhstan
yalisha@yandex.kz

***Abstract*— Clustering large volumes of data is a complicated and time-consuming task. The main goal of clustering task is to explore a given dataset and find appropriate set of cluster centers that have maximum within-cluster and minimum inter-cluster similarity.**
In our work we present parallel K-means clustering algorithm based on Partitioned Global Address Space implementation of Mapreduce model. The main idea of our approach is to exploit data locality of partitioned global address space model to assign threads to different cluster centers.

Keywords— UPC, K-Means, PGAS

I. Introduction

In recent decades the field of large-scale data processing has seen an exponential growth. For example, such tool as Mapreduce made a large leap forward in terms of enabling processing of petabytes scale data by using cluster of commodity computers. Nowadays, Mapreduce can be regarded as an indispensable tool in analyzing large volumes of data [1]. One of the important categories of data processing is an unsupervised learning. Unsupervised learning is a broad category of tools that allows learning a labeling of objects in the dataset provided that no prior information about their labeling is known. Clustering belongs to the category of unsupervised learning tools. Clustering involves the procedure of grouping objects together according to some similarity measure.

Clustering is widespread technique used in many areas of science and technology. Clustering problem arises in medicine for identifying genome patterns, in image processing for segmentation task, in natural language processing for grouping similar in context or theme documents.

Through use of the appropriate distributed computing techniques one can achieve high level of scalability and performance for the clustering problem.

Mapreduce is a parallel programming model in which parallelism is realized in three main stages: map, shuffle, reduce. The idea of Mapreduce is to distribute data processing among parallel map and reduce processes. The schematic diagram showing how Mapreduce works based on WordCount problem is given in the Fig.1.

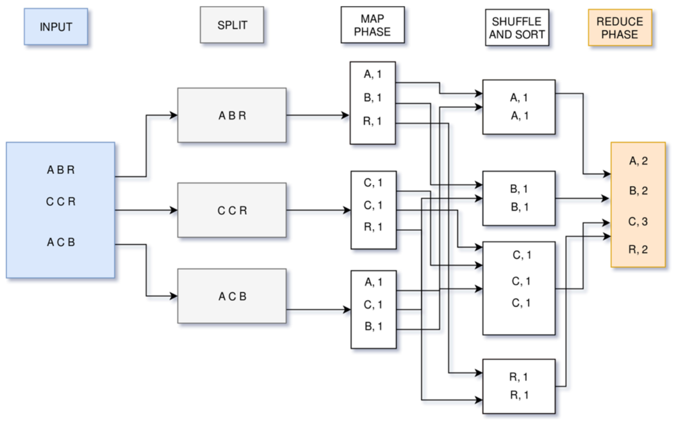

Figure 1. WordCount workflow in Mapreduce

In first stage, map processes each with certain input split simultaneously perform map function defined by the user. Each map process has a key and value parameters. Key is responsible for offset from the beginning of the file and value represent input split that map operates on. Each map process emits key-value pairs. Reduce processes work with output from first stage. In particular, all values that share the same key are grouped together and the task of reduce process is to process a single group of elements with the same key. Advantages of Mapreduce include simple approach to parallelism, close to linear scalability and fault-tolerance.

Standard application of Mapreduce is to analyze data logs, perform clustering of data, search for patterns in a text and etc. K-Means is one of the widely used and highly efficient clustering algorithms [2]. Parallel clustering algorithms such as Parallel K-Means [3], Hybrid K-Means [4] present fairly efficient solution to a clustering task. These approaches are based on Mapreduce model using standard Mapreduce frameworks such as Apache Hadoop or Apache Spark.

K MEANS

K-means algorithm tries to find k clusters given a set S of N data points such that we minimize the following functional:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S} \|x - m_i\|^2 \qquad (1)$$

K-means clustering algorithm is an iterative algorithm that computes a new set of cluster centers at the every iteration of the algorithm according to the formula:

$$m_i = \frac{\sum_{x \in C_i} x}{|C_i|} \qquad (2)$$

Average value of all points closest to C_i is taken as a new cluster center. Algorithm works until convergence is obtained, i.e. cluster centers doesn`t change their values more than specified epsilon constant.

II. UPC BASED MAPREDUCE

Mapreduce relies on distributed file system in order to save intermediate and final results of the computation. However, when dealing with iterative tasks, writing and reading each time from distributed file system is inefficient. That approach incurs a lot of disk operations and degrades overall performance due to iterative nature of clustering algorithms. As a solution to that problem we propose in-memory approach using Mapreduce framework that was built on top of unified parallel C environment [5]. Here we briefly sketch the overall architecture of our system.
Our system depends on a hashmap data structure that allows performing constant time access operations on average. We use UPC parallel programming language that belongs to a class of partitioned global address space languages (PGAS). Hashmap structure is implemented through using shared pointers in a UPC programming language.

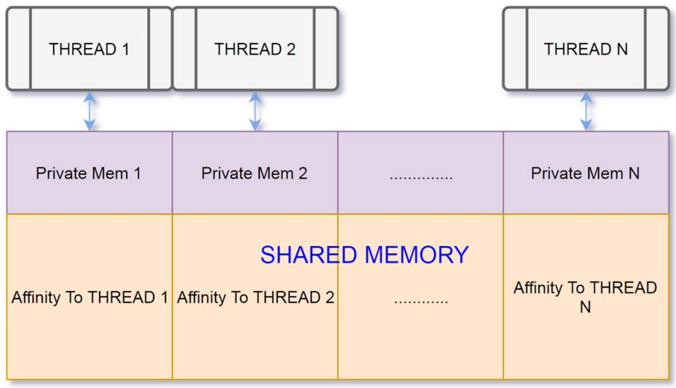

Figure 2. Memory layout of UPC programming language

PGAS languages allow more convenient and flexible remote data access operations by using one-sided communication mechanism. There is a distinction, however, of working with remote data or local one in terms of latencies they incur. Therefore, it is better to minimize fine-grained remote accesses due to order of magnitude higher latencies of remote operations. In our approach we attempt to reduce number of fine-grained operations by making shared array of hashmap entries. Wherein, we can make local accesses through private pointers, reducing overhead of dealing with slow shared pointers. UPC depends on GasNet networking layer as a communication mechanism for node-to-node communication.

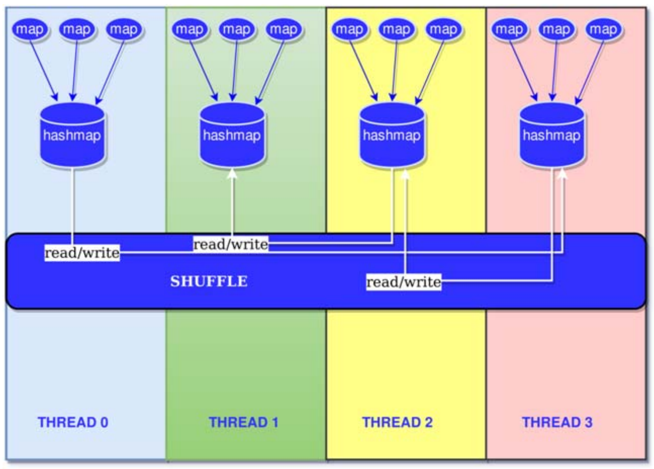

Figure 3. Design of Mapreduce based on UPC

Mapreduce in UPC was implemented based on shared hashmap data structure using upc_memput, upc_memcpy, upc_alloc, upc_memget shared memory functions. Shared hashmap contains the following functions: shared_hashmap_put, shared_hashmap_get, shared_hashmap_resize, shared_hashmap_remove and other. Storing key/value pairs in Mapreduce framework is realized by assigning to each thread a particular hashmap structure that can be accessed by other threads but at the same time independent from them in a context of a single map phase. Hashmaps store key/value pairs local to that particular thread, therefore ensuring locality of put/get operations.

III. KMEANS UPC ALGORITHM

Static data that doesn't change throughout the computation commonly can be seen in iterative algorithms. For example, in case of K-means algorithm data points don't change their values, only the cluster centers are being updated at the every iteration. We need to implement for that purpose a caching utility that maps some data into shared memory and provide interface to access that data. For example, when map processes need to fetch data points that were assigned to them, they can make a call to cache method **get** that will search for that data in cache and return reference to it when found or null otherwise. Therefore, it is now possible to avoid expensive disk operations. The cache can be implemented using shared arrays in UPC language.

```
input  : clusters [$K$], key, value, MapperId
output: $pair$(clusterCenterId, pointVal)
$minDist \leftarrow maxDistanceValue$;
if $data$ $is$ $in$ $memory$ then
    $points \leftarrow$ LoadPointsFromMemory(MapperId);
else
    $points$ = Tokenize (value);
foreach $point \in points$ do
    for $centerId \leftarrow 1$ to $K$ do
        $dist$ = findDistance ($point$,$centerId$);
        if $dist < minDist$ then
            $minDist \leftarrow dist$;
            $centerMin \leftarrow centerId$;
    end
    $kvPair \leftarrow$ MakePair($centerMin, point$);
    Emit ($kvPair$);
end
```

Figure 4. Map algorithm

```
input  : clusters [$K$], key, value
output: $pair$(clusterCenterId, clusterPoint)
$points$ = List (value);
$average \leftarrow 0$;
$sum \leftarrow 0$;
$numberOfPoints \leftarrow$ Len($points$);
foreach $point \in points$ do
    $module$ = computeModule ($point$);
    $sum \leftarrow sum + module$;
end
$average \leftarrow sum/numberOfPoints$;
$kvPair \leftarrow$ MakePair(clusterCenterId, $average$);
Emit ($kvPair$);
```

Figure 5. Reduce algorithm

K-Means algorithm in our approach is similar to the one described in paper [3]. In our approach, however we utilize in-memory data mappings. Each map process first tries to fetch data from in-memory cache. If fetch doesn`t succeed, then reading is performed from the file that was assigned to that process. Map processes, fetch data points by using **LoadPointsFromMemory** function. This function finds data points by **MapperId** unique identifier that is generated by runtime and assigned to the map processes before running them.

Input parameters for map function include: set of cluster points **clusters** (taken randomly and uniformly from set of all points in a dataset), **key** (the value that describes an offset), and list of data points **value**. Map algorithm described in Fig. 4, first creates a list of points and assigns them to **points** either from raw textual information **value**, or fetches them from in-memory cache by using **LoadPointsFromMemory** function**.**

Then, for each data point we need to find the closest cluster center, which is implemented by two nested loops: loop over points assigned to current map process and loop over cluster centers. Additionally, for each point we emit key-value pair of **centerMin** and **point** variables that describe the closest for that point cluster center and point itself respectfully.

The reduce algorithm described in Fig. 5 computes a new cluster centers. The runtime system in a shuffle phase of Mapreduce workflow determines which of the cluster centers will be processed by which thread in UPC Mapreduce system. The decision is taken based on genetic algorithm that tries to minimize number of remote operations and additionally tries to balance the overall workload among parallel UPC threads. Input for reduce algorithm consists of all data points that lie close to the cluster center with unique identifier **clusterCenterId** and the **clusterCenterId** value itself. Reduce function starts with reading data points stored in **value** parameter. Then in a foreach loop the squared mean for data each point is taken and overall sum of all that values are stored in a **sum** variable. After the loop, the quotient of the **sum** and **numberOfPoints** is assigned to an **average** variable.

The next step of the algorithm is to construct a pair from a **clusterCenterId** and an **average** variables and assign that pair to a **kvPair** variable. Then, a **kvPair** variable is passed as a parameter to an **Emit** function. Emit function then performs either a write to a disk of the obtained cluster centers or updates the cluster centers.

CONCLUSION

In this paper we presented K-Means parallel clustering algorithm based on UPC Mapreduce framework. Mapreduce part consists of two stages: map and reduce. Algorithm works by using shared memory in a UPC environment, avoiding reading data points from disk and reducing overhead of running at each iteration a new Mapreduce job.

ACKNOWLEDGMENT

The research was supported by scientific grant № AP05132933: Development of a system for knowledge extraction from heterogeneous data sources to improve the quality of decision-making (2018-2020) and scientific grant "Time variable topological analysis and visualization of tomography data for physical images obtained by high throughput methods".

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters" Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, vol. 6, pp. 10-10, December 2004.

[2] MacQueen, J. Some methods for classification and analysis of multivariate observations, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1., pp. 281—297, 1967.

[3] Zhao W., Ma H., He Q., "Parallel *K*-Means Clustering Based on MapReduce," in Cloud Computing. CloudCom 2009, LNCS vol. 5931, 2009.

[4] Sarma, T.H., Viswanath, P. & Reddy, B.E., "A hybrid approach to speed-up the k-means clustering method" Int. J. Mach. Learn. & Cyber., vol. 4, 2013.

[5] A. Shomanov, D. Akhmed-Zaki, M. Mansurova "PGAS Approach to Implement Mapreduce Framework Based on UPC Language," Proceedings of the Parallel Computing Technologies (PaCT) conference, vol. 10421, pp. 342-250, September 2017.