УДК: 004.421.2:519.16

DATA EXCHANGE MODEL FOR MAPREDUCE SYSTEM IMPLEMENTED ON UPC LANGUAGE

UPC ТІЛІНДЕ ЖҮЗЕГЕ АСЫРЫЛҒАН MAPREDUCE ЖҮЙЕСІНІН ДЕРЕКТЕРДІ АУЫСУ ҮЛГІСІ

МОДЕЛЬ ОБМЕНА ДАННЫМИ ДЛЯ MAPREDUCE СИСТЕМЫ РЕАЛИЗОВАННОЙ НА ЯЗЫКЕ UPC

SHOMANOV A.S.

AMIRGALIYEV E.N.

MANSUROVA M.E.

URMASHEV B.A.

Mapreduce is a model for parallel data processing that became famous due to ability to process data in parallel in a scalable and efficient way. Mapreduce implementations such as Apache Hadoop, Apache Spark, Mapreduce-MPI, Phoenix are widely used in a domain of scientific and industrial applications. Companies such as Google, Facebook, Amazon rely on Mapreduce for their key applications and services.
The main difficulty in implementing Mapreduce lies in algorithms that govern how shuffling is performed. Shuffle is a procedure that group similar keys obtained from different map processes and feed the result to reduce stage, where further processing is done. In this paper we propose a new shuffle method in Mapreduce system implemented on UPC language. The method is based on an idea of using shared address space implementation of hashmap data structure.

Mapreduce жоғары масштабты және тиімділігі арқылы танымал болған параллельді деректерді өңдеу үлгісі болып табылады. Mapreduce параллельді деректерді өңдеу үлгісі жоғары масштабты және тиімділік арқылы танымал болған. Apache Hadoop, Apache Spark, Mapreduce-MPI, Phoenix жүзеге асырылған Mapreduce жүйелер ғылыми және де коммерциялық қосымшаларда кеңінен қолданылады. Google, Facebook, Amazon компаниялардың басты сервистер және қосымшалар өзінің жұмысында Mapreduce қолданады.
Mapreduce жүзеге асыру негізгі қиындығы shuffle операциясын орындау алгоритмдерін құру болып табылады. Shuffle бұл әртүрлі map процесстерден алынған ұқсас кілттерді топтастыру және reduce кезеңіне келесі өңдеу үшін жіберу операция болып табылады. Бұл мақалада UPC параллельді тілінде жүзеге асырылған Mapreduce жүйесінің жаңа shuffle әдісі ұсынылады. Shuffle әдісі бөлінген ғаламдық адрестік кеңістігінде жүзеге асырылған hashmap деректердің құрылымы негізінде жүзеге асырылған.

Mapreduce – это модель параллельной обработки данных, которая стала популярной за счет способности параллельной обработки данных с высокими показателями масштабируемости и эффективности. Такие реализации Mapreduce как ApacheHadoop, ApacheSpark, Mapreduce-MPI, Phoenix широко используются как в научных, так и коммерческих приложениях. Основные сервисы и приложения в таких компаниях как Google, Facebook, Amazon используют для своей работы Mapreduce.
Основная трудность в реализации Mapreduce заключается в алгоритмах, которые определяют, как выполняется операция перетасовки (shuffle). Shuffle – это процедура, которая группирует похожие ключи, полученные из разных процессов map, и передает результат на этап reduce, где происходит дальнейшая обработка. В этой статье предлагается новый метод тасования данных (shuffle) в системе Mapreduce, реализованной на языке UPC. Метод основан на идее использования реализации hashmap структуры данных в разделенном глобальном адресном пространстве.

Keywords: *Mapreduce, UPC, shuffle, parallel computing*

Introduction

Mapreduce is a distributed computing model used for parallel computations and data processing of large amounts of data in a distributed cluster environment. Mapreduce model was introduced by Google in 2004 [1]. The work in Mapreduce is performed in 3 stages:

1. Map
2. Reduce
3. Shuffle

In the Fig. 1 below general workflow of Mapreduce model is presented.

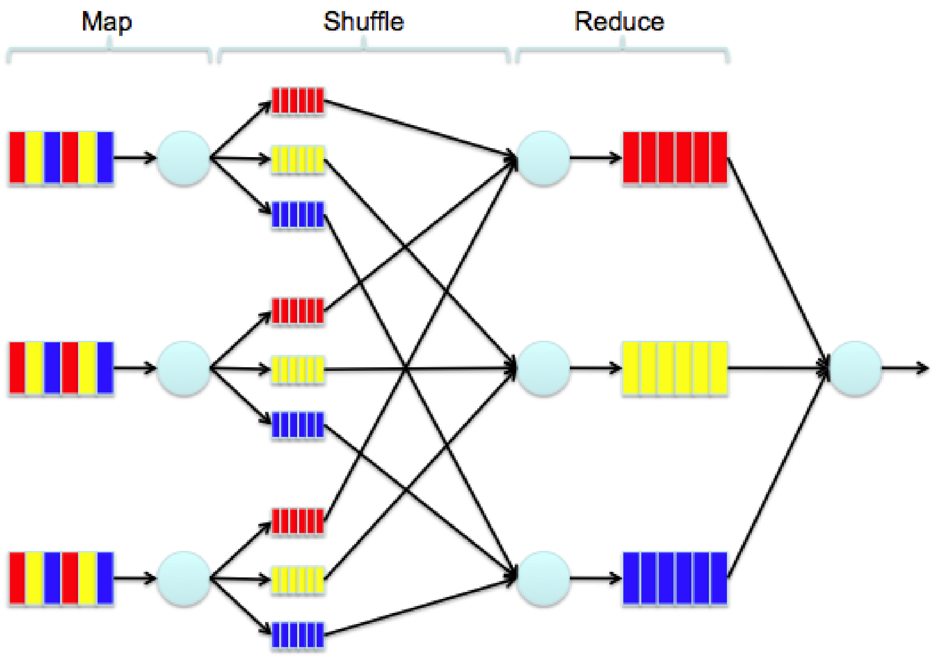

Fig. 1. Mapreduce workflow

Mapreduce works by dividing processing among parallel processes each of which implement map or reduce functions. Different processes operate in parallel on different chunks of data. Map processes are responsible for processing input data and generating pairs of key/value elements. Further in shuffle phase those pairs of elements are required to be delivered and distributed among reduce workers. Each reduce worker are responsible for processing all the values with the same key. Therefore, in a shuffle phase each map worker sends keys to reduce workers that are waiting for receiving them. After all communications are finished each reduce worker can continue with executing their routines. Reduce workers accept as an input a pair of key/{list_of_values}. The functionality of reduce is to obtain some result in a form of key/value pair derived from the input data.

The main principles of Big Data development can be formulated as follows:

- Scalability
- Fault-tolerance
- Locality of data

Mapreduce model can be efficiently implemented such that all of these properties are satisfied. The main examples of such implementations include such frameworks as Apache Hadoop, Apache Spark, Mapreduce-MPI [2-4].

At the moment such Mapreduce systems as Apache Hadoop, Apache Spark, Google Mapreduce are widely used as an effective means of data processing and analysis. Mapreduce is actively used in companies such as Google, Yandex, Facebook as the main tool for speeding up work and increasing the amount of processed data.

The implementation of Mapreduce systems is based on several basic elements that ensure the functioning of the entire system. These elements are data storage mechanisms (distributed file system), mechanisms for organizing data exchange between individual nodes or processes in a distributed computing environment, ensuring fault tolerance of the system, and organizing load balancing between parallel processes.

Some of the Mapreduce frameworks are being developed for specifically targeting different hardware architectures. For example, MapReduce system called Phoenix specifically has been developed for shared-memory systems [5]. Phoenix uses shared memory for the purpose of reducing the overheads of data communication and task startup latencies. Phoenix has been shown to achieve high scalability for different types of multiprocessors architectures. Also Phoenix has built-in tools to perform scheduling of key distribution among threads and have recovery mechanisms for faults in performing map and reduce tasks. The authors of the paper compared Phoenix with P-threads execution model which resulted in comparable performance for majority of applications. Also authors report considerable decrease in code size for majority of applications compared to P-threads approach.

Mapreduce has been explored in traditional HPC platforms before in [6]. As a result authors conclude that Mapreduce can be efficiently implemented using MPI with some limitations.The approach described in the paper works based on application of collective non-blocking operations. In the map stage MPI_Scatter and on reduce stage MPI_Reduce collective functions are used. Master-worker model is employed. Master is responsible for scheduling computation tasks. Subsequently, tasks are executed by worker processes. Limitations on reduce function imposed by MPI library are the following:

1) Reduce function must be associative

2) Number of different keys must be known for each process ahead of reduce function call

3) If some key is missing for some processes the key value must be assigned some identity element value

Using scalable MPI collective functions it is possible to obtain high speed-up. Numerous architectures even support hardware-optimized collective operations.

MapReduce is designed to manage and handle fault issues transparently.

Similar work has been done by a research team from Sandia National Laboratories, USA and described in paper [7]. Their work based on implementing Mapreduce using MPI. MR-MPI framework has the following features:

1) Inter-process communication based on MPI C++ library.

2) Portability and small size of the library

3) In-core or out-of-core processing.

4) Flexible programmability

5) Python, C, C++ interfaces

6) Absence of fault-tolerance and data redundancy

The approach presented in paper for data distribution among reduce workers has certain limitation associated with poor data locality due to random shuffling of key/value pairs. That randomness leads to higher network load due to large number of data movements between different processes. However, the advantage of random distribution is efficient load-balancing among processes.

Our approach to implement shuffle method in Mapreduce system was developed in UPC parallel programming language. UPC language is an extension for C language for high-performance computing on large-scale distributed clusters.

UPC is based on PGAS (Partitioned Global Address Space) parallel programming model [8]. In PGAS model concurrent threads operate on partitioned shared address space. Each thread has an access to its own private memory and partitioned shared address space. Partitioned shared address space is divided among threads into non-overlapping regions. Each region has an affinity to a particular thread.

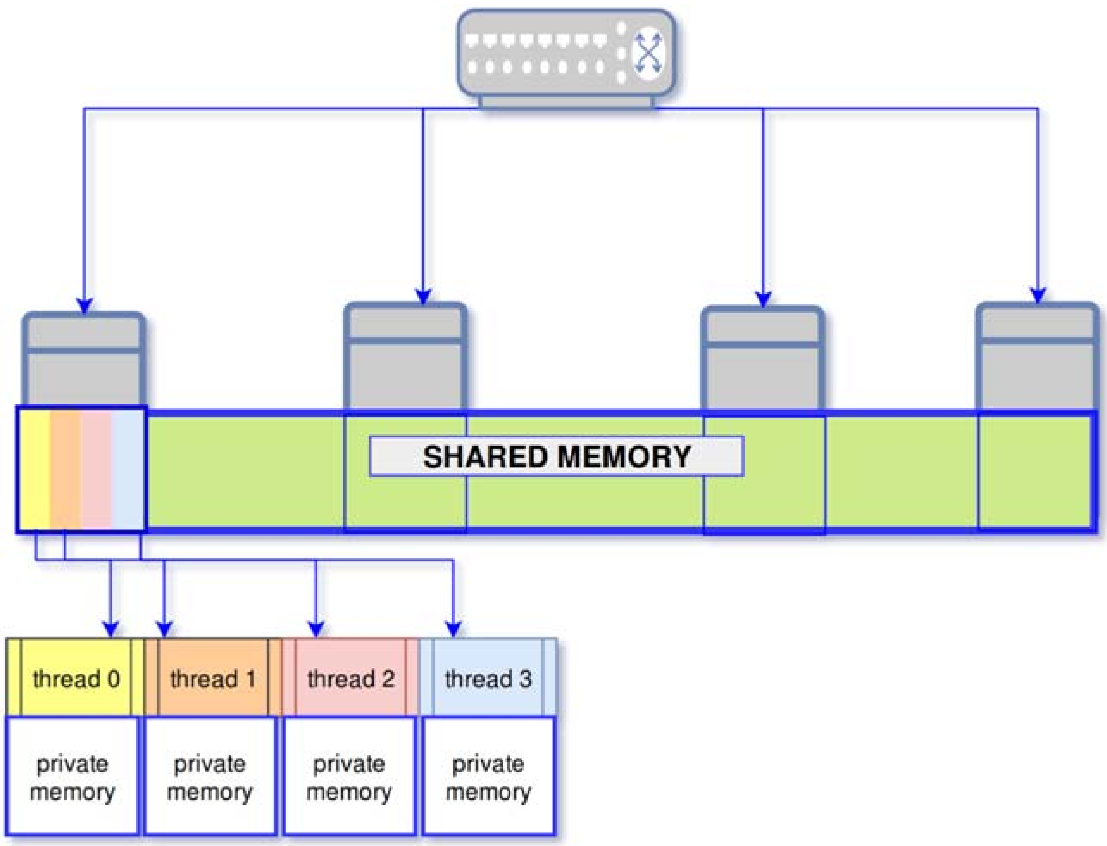

Fig. 2. PGAS memory model

From Fig. 2 it can be seen that shared memory region spans the physical memory of all computing nodes in a cluster and colored areas of shared memory represent memories that has affinity to a particular thread.

In this work we introduce a new approach to perform shuffle operation in Mapreduce system implemented on UPC language based on using hashmap data structure. Hashmap is a data structure that implements an associative array (mapping from keys to values). Hashmap uses a hash function that assigns an index to a key from which value with specified key can be found. Hashmap is very efficient for applications that require storing key/value pairs. Amortized complexity of read/write operations for hashmap data structure is equivalent to $O(1)$.

Choosing a good hash function is indispensable for hashmap insert/extract operation performance. One of the main properties of a good hash function is uniformly distributed hash indexes. Non-uniformity leads to increased number of collisions and as a result increased time of resolving these collisions.

Collision resolution methods are used in order to avoid mapping of two or more keys to a single index value. Separate chaining and open addressing are two different methods for collision resolution.

Nowadays it is essential to introduce new parallel computing solutions that can enhance ability to process the large amounts of data.

Similar works of developing Mapreduce system based on PGAS (partitioned global address space) model were presented in the literature before []. One approach is based on UPC language with idea of using collective functions to perform key/value exchange. Authors argue that their approach possess programmability benefits of parallel code development, customizable load distribution, target an issue of performance bottlenecks leading to better approach of implementing Mapreduce in HPC systems. Although, that approach has shown good performance results the issue of optimized key/value distribution still remains unsolved.

Implementation of the shuffle phase for Mapreduce based on UPC

The usual shuffle method implemented in many Mapreduce systems works by applying hashing to the keys and sending keys to the reducer that corresponds to the computed hash value. After that all keys are sorted by merge-sort algorithm on each reducer. Basically, that means that in usual implementation of shuffle method keys distribution among reducers cannot be controlled and optimized. In our approach we propose a new method for performing shuffle procedure based on using UPC language and hashmap data structure implemented on top of PGAS memory model.

The difficulty with implementing hashmap data structure in UPC language lies in addressing the issue of changing all operations on hashmap to operate in shared address space. The implementation of hashmap data structure is based on using shared pointers and UPC memory manipulation functions such as upc_memget, upc_memput, upc_memcpy. UPC memory manipulation functions allow to access and change values of variables and memory locations which are referenced by shared pointers.

```
typedef shared struct _hashmap_map
{
inttable_size;
int size;
shared []hashmap_element*data;
}
hashmap_map_global;
```

Fig. 3.Hashmapstruct definition.

Definition of hashmap structure is presented in Fig. 3. This definition include size of the hash table *table_size*, current number of elements in hashmap denoted by *size* variable and shared array of hashmap elements stored in *data* variable.

```
typedefstruct _hashmap_element{
shared []char* key;
intin_use;
intind;
shared []shared_vector* data;
int id;
}hashmap_element;
```

Fig. 4.Hashmap_elementstruct definition.

In Fig. 4 definition of hashmap_element structure is presented. This structure represents a single entry stored in shared array *data* of hashmap structure. Each element of the hashmap structure consists of a key and a vector of values that correspond to the given key. That is, when a new element is written to hashmap with an existing key the value will be added to a vector of values corresponding to that key. In such a way we avoid storing multiple entries of the same key in hashmap data structure.

```
            hashmap_map_global hash [THREADS];
            hash[MYTHREAD].data =(shared [] hashmap_element
            *)upc_alloc(INITIAL_SIZE *     upc_elemsizeof(shared
hashmap_element));
```

Fig. 5. Defining and initializing of shared array of hashmap structures

In Fig. 5 *hash* shared array is defined of size equivalent to number of threads, such that each entry of that array of hashmap data type is in one-to-one mapping with each thread. Also second instruction initializes for each thread shared array *data*.

Hashmap defined in such a way is accessible for remote read/write operations and therefore allow exchanging key/value pairs among different entries in shared array of hashmap elements. The described exchange method is foundation for implementing shuffle approach in our Mapreduce on UPC model.

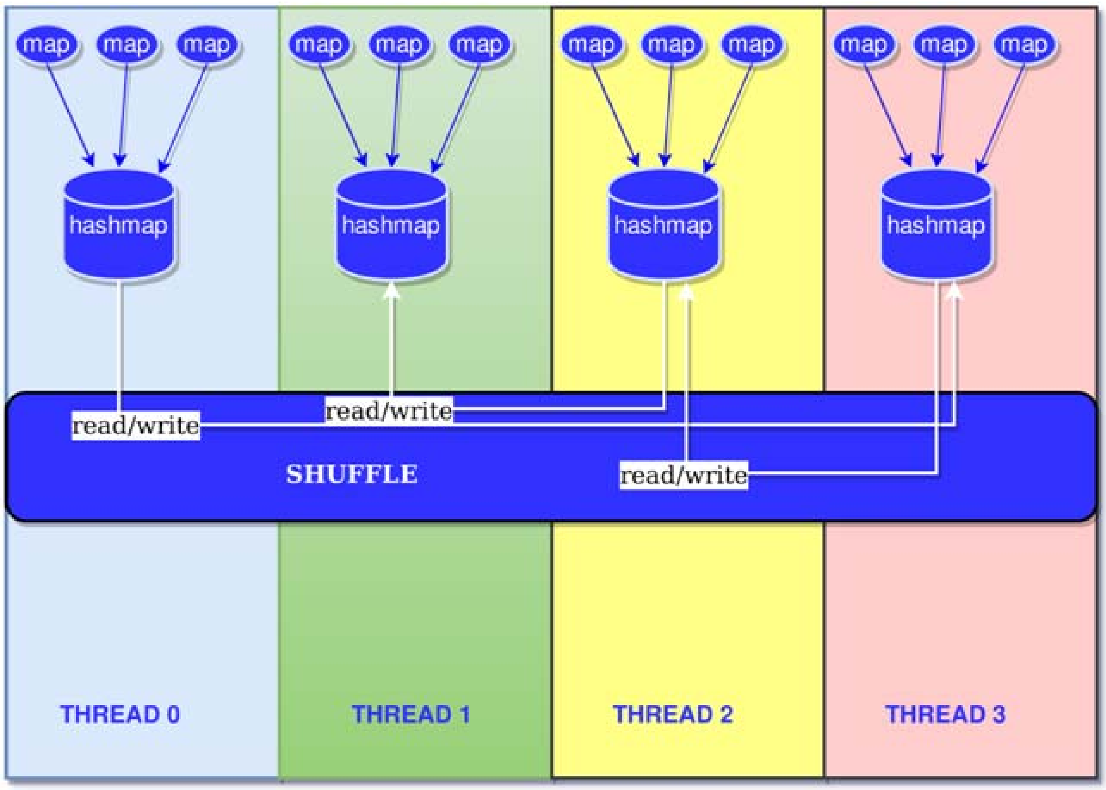

Fig. 6. Data exchange model in Mapreduce

The further steps of Mapreduce shuffle method implementation in our approach involves deciding which threads would process which set of keys. For this task we rely on optimization strategy based on minimizing remote fetching and updating operations and distributing computational workload among threads with aim to uniformly distribute key/value pairs. The general scheme how shuffle method is implemented in our model can be described by the following steps:

1) Assign each key unique integer identifier.

2) Create a new array of hashmap entries that would store in each thread <key,list_of_values> pairs.

3) Perform the task of mapping keys to threads according to key indexing from Step 1.

4) Perform the task of distributing keys among threads according to the obtained mapping by copying values of keys from each old hashmap to new hashmap created in Step 2.

Following these steps, runtime system can perform in parallel reduce function calls on each <key,list_of_values> entry of local thread`s hashmap entry of shared array created in Step 2.

Such an approach presents a new way of organizing Mapreduce shuffle method in a UPC language.

The basic idea of using a globally addressable hashmap structure for implementing Mapreduce technology consists of the following ideas:

1. In the map phase, the intermediate key / value pairs are written to hashmap, which is local to the current thread.

2. At the reduce stage, all values with the same key are copied to the thread that will perform the reduce operation for this key. Thus, since each thread has access to hashmap elements of other threads, it can query elements with the given key from the other hashmap structures.

3. The read / write operations on hashmap are performed in asymptotic complexity of $O(1)$, which can significantly shorten the search time for the desired key.

4. UPC uses the GasNet communication system, which is based on an efficient process of transferring and exchanging data between computational nodes on the basis of the "active messages" protocol during operations with memory.

REFERENCES

1. J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.**doi:** 10.1145/1327452.1327492
2. Zaharia, M., M. Chowdhury, M.J. Franklin, S. Shenker and I. Stoica, 2010. Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Jun 22-25, ACM, USA, pp: 10-10.
3. http://hadoop.apache.org/
4. MapReduce in MPI for Large-Scale Graph Algorithms, S. J. Plimpton and K. D. Devine, Parallel Computing, 37, 610-632 (2011),doi:10.1016/j.parco.2011.02.004.
5. Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., &Kozyrakis, C. (2007). Evaluating MapReduce for multi-core and multiprocessor systems. Paper presented at the *Proceedings - International Symposium on High-Performance Computer Architecture,* 13-24. doi:10.1109/HPCA.2007.346181
6. Hoefler, T., Lumsdaine, A., &Dongarra, J. (2009). *Towards efficient mapreduce using MPI* doi:10.1007/978-3-642-03770-2-30
7. Plimpton, S. J., & Devine, K. D. (2011). MapReduce in MPI for large-scale graph algorithms. *ParallelComputing, 37*(9), 610-632. doi:10.1016/j.parco.2011.02.004
8. W.W. Carlson, J.M. Draper, D.E. Culler, K. Yelick, E. Brooks, K. Warren, Introduction to UPC and language specification, Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, 1999.