Poznań University of Technology

Dominance properties for Divisible MapReduce Computations

J.Berlińska, M.Drozdowski

Research Report RA-09/09
2009

Institute of Computing Science, Piotrowo 2, 60-965 Poznań, Poland

Dominance properties for Divisible MapReduce Computations

J.Berlińska[1], M.Drozdowski[2]

1) Faculty of Mathematics and Computer Science,
Adam Mickiewicz University, Umultowska 87, 61-614 Poznań, Poland
2) Institute of Computing Science,
Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland

Abstract

In this paper we analyze MapReduce distributed computations as divisible load scheduling problem. The two operations of mapping and reducing can be understood as two divisible applications with precedence constraints. A divisible load model is proposed, and schedule dominance properties are analyzed. We investigate dominant schedule structures for MapReduce computations. To our best knowledge this is the first time that processing divisible loads with precedence constraints is considered on the grounds of divisible load theory.

Keywords: Parallel processing, MapReduce, scheduling, divisible loads.

1 Introduction

Divisible load theory (DLT) is a model of distributed processing. It assumes that the computations can be divided into pieces of arbitrary sizes and that there are no precedence constraints between these pieces. Thus, the pieces of work can be processed independently on remote computers. The divisible work is generally termed load, and in most of the cases refers to computations on big datasets. There are many examples of divisible load applications, including search for patterns in text and database files [14], processing measurement data [8], image and video processing [17, 18, 19] and linear algebra [10, 15]. Applications on platforms BOINC and distributed.net also fulfill the divisibility and independence of load grains assumptions. Thus, DLT delivers a method of analyzing a broad class of parallel computations.

Divisible load model originated in the late 1980s [1, 8]. It has been applied to represent distributed computations in a network of workstations [1]. A chain of intelligent sensors was analyzed in [8]. Both publications consider a problem of partitioning the computation so that the total processing time is as short as possible. On the one hand distributing the computations reduces processing time by employing additional processors. But on the other

hand, communications cost time. Hence, the problem is what amounts of load should be sent to which processors. The mathematical model proposed in the early publications reduced the scheduling problem to a system of linear equations. Later on, DLT branched in many new directions covering scheduling problems for various types of interconnection networks [8, 9, 13, 6], memory limitations [16, 3], costs of computation [21], and other. The **NP**-hardness of the most general divisible load scheduling problem was proved in [22]. Surveys of DLT can be found e.g. in [2, 4, 12, 20].

In this paper we analyze *MapReduce* distributed computations as divisible load scheduling problem with precedence constraints. MapReduce is a programming model for processing large data sets on big numbers of computers [11, 23]. The idea of MapReduce is outlined in the next section. We propose a divisible load model of MapReduce, and analyze properties of the optimum schedules..

The rest of this paper is organized as follows. In Section 2 we describe MapReduce. We formulate its mathematical model and introduce notation in Section 3. Section 4 is dedicated to computations with a single reducer, and Section 5 to processing with many reducers.

2 Outline of MapReduce

MapReduce can be implemented in many ways, and indeed it has various implementations [23]. Here, we will outline MapReduce as described in [11]. In a nutshell, MapReduce computations consist in processing input data sets by creating a set of intermediate (key,value) pairs, and then reducing them to yet another list of (key,value) pairs. The computations are performed in parallel.

More precisely, MapReduce applications are divided into two steps. In the first step a *Map* function processes the input dataset (e.g. a text/HTML file), and a set of intermediate $(key1, value1)$ pairs is generated. In the second step the intermediate values are sorted by $key1$, and a *Reduce* function merges the intermediate pairs with equal value of $key1$, to produce a list of pairs $(key1, value2)$. Thus, the input dataset is transformed into a list of key/value pairs. Let us consider examples given in [11]. Counting occurrences of words in a big set of documents can be organized in the following way. Map function emits intermediate pair $(word, 1)$ for each $word$ in the input file(s). The intermediate pairs are reduced by summing 1s, and producing pairs $(word, count)$. In the inverted index computation all documents comprising certain $word$ must be identified. The Map function emits pairs $(word, docID)$, where $docID$ is a document identifier (e.g. a URL of a web

page). In the reduce function all $(word, docID)$ pairs are sorted, and pair $(word, list\_docIDs)$ is emitted, where $list\_docIDs$ is a sorted list of $docID$s. There are many types of practical applications which can be expressed in MapReduce model. More detailed and advanced examples are given in [11].

Both map and reduce operations are performed in parallel in a distributed computer system. Processing a MapReduce application starts with splitting the input files into load units (in [11] called splits). Many copies of the program start on a cluster of machines. One of the machines, called master, assigns work to the other computers (workers). There are m map tasks and r reduce tasks to assign. In the further discussion map tasks will be called *mappers*, and the reduce tasks *reducers*. A worker which received a mapper reads the corresponding input load unit and processes the data using the *Map* function. The output of this function is divided into r parts by *partitioning function* and written to r files on the local disk. Each of the r files corresponds to one of the reducers. Usually the partitioning function is something like $hash(key1) \bmod r$. The information about local file locations is sent back to the master, which forwards it to the reduce workers.

When a reduce worker receives this information, it reads the buffered data from the local disks of the map workers. After reading all intermediate data, the reduce worker sorts it by the intermediate keys in order to group together all occurrences of the same intermediate key. Each key and the corresponding set of values are then processed by the *Reduce* function. Its output is appended to a final output file for a given reducer. Thus, the output of MapReduce is available in r output files. The execution of MapReduce is completed when all reducers finish their work.

3 Toward a Model of MapReduce

In this section we formulate mathematical model of MapReduce computations. We will pass from 'microscopic' view of the computation to a coarser 'macroscopic' model used in the following sections. We simplify perception of MapReduce computations to build mathematically and conceptually tractable representation of the complex computing platform and the distributed application. Notation introduced in the paper is summarized in Table 1.

We assume that MapReduce applications are executed by identical processing elements which have CPU, local storage, independent network interface (e.g. NIC and DMA). Terms processing element, computer, processor, worker will be used interchangeably. Let P_i denote a processor i. The structure of the interconnection network is unknown in general. Yet, it is known

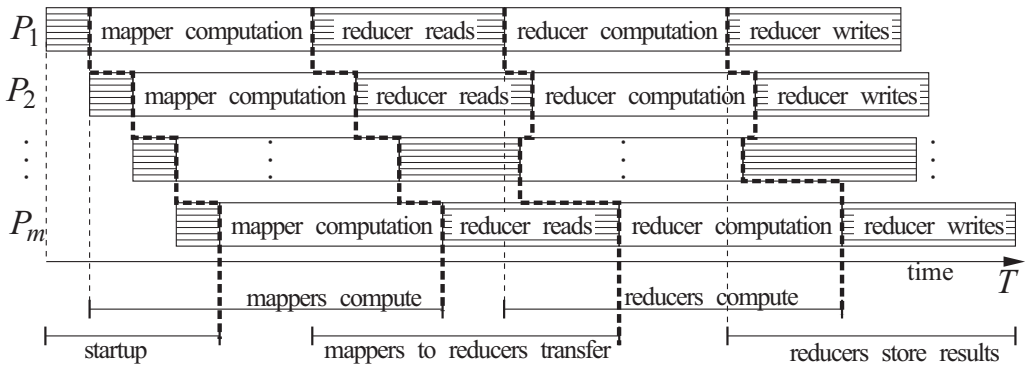

Figure 1: General view of MapReduce schedule structure.

that the bandwidth of the unthrottled communication channels which can be simultaneously used is limited. We will represent this limitation as the number l of communication channels which can be simultaneously in use without reducing the channel communication speed. In other words, if two processors can communicate with speed $1/C$ in the otherwise unused network, then the bandwidth limitation for the concurrent channels in the whole network is l/C. When referring to the above limit on the number of concurrent channels in the network we will be saying about *bisection width* limit. We perceive the mappers and the reducers in a more coarse way than in [11]. In [11] a mapper is an application executing Map function for one load unit. The size lu of the load unit is 16-64MB, and a processor receives approximately 100 load units [11]. Here we will assume that a single mapper is an application executing Map function for all the load (i.e. all load units) assigned to a certain processor. Thus, there is one-to-one correspondence between mappers and processors. Therefore, in the following discussion we can use words processor, and mapper interchangeably when referring to source and destination of data transfers. Let m denote the number of mappers (consequently, also processors executing them). Similarly, we unify all reducer computations assigned to a certain processor to a single (compound) reducer. The number of reducers is denoted by r. The total size of load to be processed is V (e.g. bytes).

A rough schedule structure of MapReduce computation is shown in Fig.1. Detailed schedule structures are analyzed in the following sections. MapReduce computation is divided into several phases which may partially overlap. In the first stage the code for mapper and reducer applications is loaded on processors. For the sake of simplicity of presentation we assume that the mapper and the reducer codes are uploaded together. We assume that most of the processors read the code from the network file system. The code may

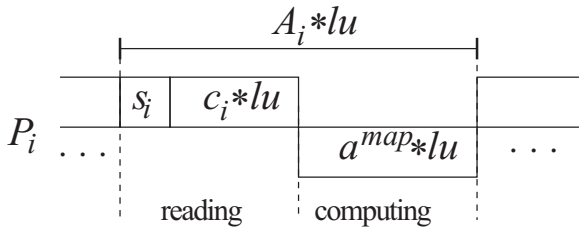

Figure 2: Microscopic view of Map computation for a single load unit of size.

include virtual machines, libraries, the mapper and reducer codes themselves. Thus, computation startup time S may be relatively long. The computation startup time elapses only once because when processing the following load chunks the code already resides on the executing processor. The differences in the startup time between the processors are negligible.

In the second phase mappers read load units from the network file system, process them, and store the results in r local files for r reducers. A 'microscopic' view of processing a single load unit of size lu (e.g. in bytes) by a mapper i is shown in Fig.2. A processor P_i (running mapper i) reads lu bytes of input in time $s_i + c_i * lu$. Though computers may be the same, the load may be read from local or from remote locations. Consequently, s_i, c_i are different for different processors. The fixed time delay s_i includes both communication and computation startup times needed in practice to start computation, and read load for the next load unit. The lu bytes of input are processed in time $a^{map} * lu$. This time comprises both computations and storing the results in local files. Thus, we assume that from the point of view of local computations processors are essentially the same, because local computing rate a^{map} is the same for all processors. The total time of processing a load unit is $s_i + (c_i + a^{map})lu$. Since the load read, processing and storing operations are repeated many times (for an order of hundreds of load units) we simplify representation of these operations to processing with rate A_i. It follows from the above discussion that $A_i * lu = s_i + (c_i + a^{map})lu$, and the operations performed by the mapper may be perceived as if processing the load with average rate $A_i = s_i/lu + c_i + a^{map}$. Though A_i depends on lu, the size of load unit is fixed for MapReduce computation, hence also A_i is constant. In the following discussion we will use this coarse representation of mapper computations as performed with rate A_i. Let α_i denote the total size (expressed e.g. in bytes) of load assigned to processor P_i (i.e. also to mapper i). According to the methodology of DLT we assume that α_i is a rational number. This simplification has two implications. First, the load assignment obtained in our model (e.g. α_is) needs rounding to load units

used in practical MapReduce. We assume that effects of load rounding are negligible. Second, since processing a load unit requires startup s_i, the size of the load unit lu influences via A_i the aggregated speed of computation. Moreover, it will be assumed that the amount of results produced by mappers is proportional to the input size. For α_i bytes of input $\gamma\alpha_i$ bytes of output are produced.

In the third stage (cf. Fig.1) results stored on the mappers are read by the reducers. We assume that partitioning function divides the space of key values into r equal parts. This is achieved by use of hashing in distributing the mapper output as described in Section 2. Consequently, the size of the input for each reducer is roughly equal to $\gamma V/r$ bytes in m chunks of sizes $\gamma\alpha_1/r, \ldots, \gamma\alpha_m/r$. Each chunk comes from a dedicated file on different processor. We assume that reducers read the load from the mappers with equal rate C. There may be some advantages in the communication speed if the mapper and the reducer are executed on the same processor. Yet, each reducer has to read its input from all processors and such advantages cancel out when averaged over the inputs from all mappers. Consequently, we assume that differences in communication rate for the transfers from mappers to reducers are negligible. Each of r reducers reads its input from P_i in time $\gamma\alpha_i C/r$ provided that there is no bandwidth limitation. At most one channel can be opened to a mapper with transfer rate C. The methods of incorporating bandwidth limitations in the communication model are described in the following sections.

In the fourth stage r reducers sort the input data, perform Reduce operations, and finally in the fifth stage store the results in the network file system. Let s^{red} denote reducer computation startup time, and a^{red} (in seconds per byte) the reducer processing rate. Parameter a^{red} represents both computations, transfers to local disks and storing the results in the network file system. All reducers receive input of roughly the same size $\gamma V/r$. As a consequence, all reducers have roughly equal execution time $t_R = s^{red} + \tau(\gamma V/r)$, where $\tau(x)$ is the running time of a reducer vs. size x of the input. We will assume that sorting dominates in reducer execution time, and $\tau(x) = a^{red}(x \log x)$. Here we assumed that writing the reducer results to the network file system in the last stage is contention-free. This may not be true in general. Precautions to avoid reducers writing contention are discussed in Section 5.

We assume that reducers are executed on the same processors as mappers, but a reducer starts computation after completion of a mapper. Thus, the mapper and the reducer computations do not interleave on the same processor. As a consequence, $r \leq m$, yet it is possible to represent also $r > m$ in our model (cf. Section 5). We exclude simultaneous execution of several

mappers, or reducers, on the same computer. Were such coallocation possible, it can be represented in our model as several processors, each running a different mapper or reducer. If there are other background services executed by the processor (e.g. for the network file system), then we assume that the services influence the processor performance in a constant way. In other words, simultaneous computation and communication is possible and performance parameters such as $a^{map}, a^{red}, c_i, S, s_i, s^{red}$ remain constant.

MapReduce implementation includes procedures to tolerate failures. We do not include them in our model. However, since some of the fault tolerance methods are based on retrying failed computations, these features can be represented as processing load greater than V (for mappers) or running additional reducers.

It is possible to optimize schedule length by assigning different amounts of computations to different reducers. This may be achieved by devising appropriate partitioning functions. Then, by better load distribution between the reducers, more of them may work in parallel and finish computations simultaneously. Though it is intuitively advantageous considering minimization of schedule length, a negative effect is that reducers store their results nearly simultaneously, exercising distributed file system performance limitations. To avoid such a situation, we do not allow for schedule length optimization by varying size of the load assigned to the reducers.

Our goal is to partition input load of size V into mapper chunks $\alpha_1, \ldots, \alpha_m$ and schedule mapper to reducer communication so that the whole schedule length T is as short as possible.

4 Processing with a Single Reducer

In this section we analyze MapReduce computations with a single reducer. This special case will be used in the many-reducer case in the next section.

4.1 Schedule Dominance Properties

In this section we will show that it is advantageous to start computations on faster processors first (smaller rate A_i), that results should be returned in the order of activating the processors, and that sharing bandwidth while reading mapper outputs is not advantageous.

We will say that the order of reading the results from the mappers by a reducer is FIFO order if a reducer reads its inputs (mapper outputs) in the order of starting computations on the mapper processor (Fig.3a). The opposite sequence of reading the results from the last activated mapper pro-

Table 1: Summary of notation

α_i	the load size processed by mapper i; in bytes;
a^{map}, c_i, s_i	'microscopic' computing rate, communication rate, communication startup time for processor P_i executing mapper application; in seconds; lower is better;
a^{red}, s^{red}	'microscopic' computing rate, and computation startup time for reducer application, equal for all processors; in seconds; lower is better;
$A_i = \frac{s_i}{lu} + a^{map} + c_i$	'macroscopic' computing rate of processor P_i executing mapper application;
C	communication rate for reading mapper results, equal for all processors;
γ	mapper result multiplicity fraction;
l	bisection width limit, in parallel channels
lu	size of load unit, e.g. in bytes
m	number of mapper processors
P_i	processor (i.e. computer) i;
r	number of reducer processors
S	computation startup time, equal for all processors;
T	schedule length;
$\tau(x)$	reducer computing time function in load size x;
$t_R = s^{red} + \tau(\gamma V/r)$	execution time of a reducer;
V	size of input load at the start of computation; in bytes;

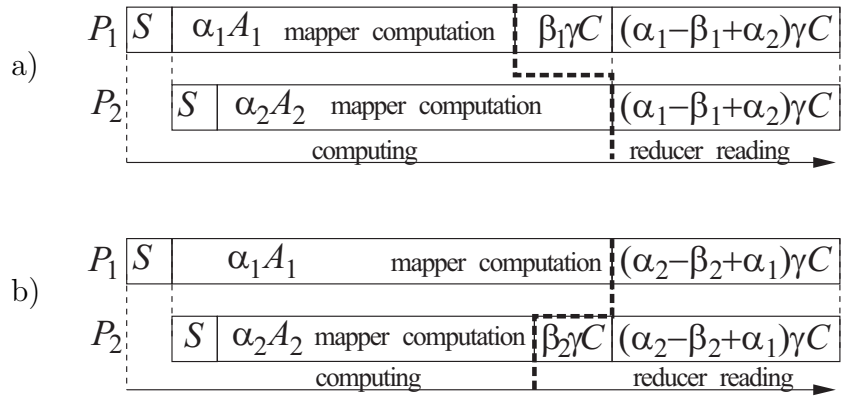

Figure 3: Reducer read orders. a) FIFO schedule structure, b) LIFO schedule structure.

cessor, and finishing with the first mapper activated will be called LIFO order (Fig.3b). The results can be read from the mappers *sequentially*. This means that only after reading the whole file from mapper i can a reducer start reading the file from mapper $i+1$ (in the given FIFO, or LIFO sequence). In the opposite case a reducer may open two communication channels to mappers i and $i+1$ and read the files concurrently. In the latter case the bandwidth $1/C$ of the input to the processor running a reducer is *shared* equally by both channels. In the following lemma we argue that faster processors should start computations first, and that results should be read sequentially in the FIFO order.

Lemma 1 *A schedule activating faster mapper processors first, with sequential FIFO reducer reads is optimum.*

Proof. We will show that the above schedule structure is optimum by comparing the amounts of loads processed by mapper processors in given time T against different schedule organizations. The schedule structure proposed above allows for processing bigger load in T than in other schedules. Therefore, it also allows for processing given load V in the shortest time.

Let us first analyze FIFO structure (see Fig.3a) with bandwidth sharing. A reducer reads from the first mapper $\gamma\alpha_1$ load. Let $0 \leq \gamma\beta_1 \leq \gamma\alpha_1$ be the part of the load read from the first mapper while the second mapper is still computing. The remaining part $\gamma(\alpha_1 - \beta_1)$ is read in parallel with the results from the second mapper. The speed of reading mapper results is determined by the shared bandwidth $\frac{1}{C}$ of the reducer input interface. Thus, we have the following relationships in the computing and communication times:

$$P_1: \qquad S + \alpha_1 A_1 + \beta_1 \gamma C + (\alpha_1 - \beta_1 + \alpha_2)\gamma C = T \tag{1}$$

$$P_2: \quad 2S + \alpha_2 A_2 + (\alpha_1 - \beta_1 + \alpha_2)\gamma C = T \tag{2}$$

from which we obtain

$$P_1: \quad S + \alpha_1(A_1 + \gamma C) + \alpha_2 \gamma C = T \tag{3}$$
$$P_2: \quad 2S + \alpha_2(A_2 + \gamma C) + (\alpha_1 - \beta_1)\gamma C = T. \tag{4}$$

From (3) we obtain

$$\alpha_2 = \left(T - S - \alpha_1 \left(A_1 + \gamma C\right)\right) / \gamma C \tag{5}$$

which substituted in (4) yields

$$\alpha_1 = \frac{TA_2 + S\gamma C - SA_2 - \beta_1 \gamma^2 C^2}{A_1 A_2 + A_1 C\gamma + A_2 C\gamma}. \tag{6}$$

Returning with α_1 to (5) load α_2 is

$$\alpha_2 = \frac{TA_1 - 2SA_1 - S\gamma C + \beta_1 \gamma C A_1 + \beta_1 \gamma^2 C^2}{A_1 A_2 + A_1 C\gamma + A_2 C\gamma}. \tag{7}$$

Together we have

$$\alpha_1 + \alpha_2 = \frac{(T - S)(A_1 + A_2) - SA_1 + \beta_1 \gamma C A_1}{A_1 A_2 + A_1 C\gamma + A_2 C\gamma}. \tag{8}$$

Note that the above load is increasing with β_1. Hence, it is the biggest if $\beta_1 = \alpha_1$. This means, that the bandwidth is not shared while reading results from the second mapper. Therefore, equation system (3)-(4) gets the following form:

$$P_1: \quad S + \alpha_1 A_1 + (\alpha_1 + \alpha_2)\gamma C = T \tag{9}$$
$$P_2: \quad 2S + \alpha_2(A_2 + \gamma C) = T. \tag{10}$$

From (10) we obtain

$$\alpha_2 = \frac{T - 2S}{A_2 + \gamma C}, \tag{11}$$

and by observing that $S + A_2\alpha_2 = \alpha_1(A_1 + \gamma C)$ (cf. Fig.3a) we get

$$\alpha_1 = \frac{TA_2 + S\gamma C - SA_2}{(A_1 + \gamma C)(A_2 + \gamma C)}, \tag{12}$$

The total processed load is

$$\alpha_1 + \alpha_2 = \frac{T(A_1 + A_2) + T\gamma C - 2SA_1 - SA_2 - S\gamma C}{(A_1 + \gamma C)(A_2 + \gamma C)} \tag{13}$$

Let us now analyze the LIFO result reading order (cf. Fig.3b). First let us check if bandwidth sharing while reading mapper results is profitable. Let $0 \leq \gamma\beta_2 \leq \gamma\alpha_2$ be the part of results read by a reducer from P_2 while P_1 is still computing. Analogously to (3), (4) we obtain in LIFO case:

$$P_1: \quad S + \alpha_1(A_1 + \gamma C) + (\alpha_2 - \beta_2)\gamma C = T \tag{14}$$

$$P_2: \quad 2S + \alpha_2(A_2 + \gamma C) + \alpha_1\gamma C = T. \tag{15}$$

From (15) we derive α_1 and substitute it in (14) from which we obtain

$$\alpha_2 = \frac{TA_1 - S\gamma C - 2SA_1 - \beta_2\gamma^2C^2}{A_1A_2 + A_1C\gamma + A_2C\gamma}. \tag{16}$$

By substituting α_2 in (15) we have

$$\alpha_1 = \frac{TA_2 - SA_2 + S\gamma C + \beta_2A_2\gamma C + \beta_2\gamma^2C^2}{A_1A_2 + A_1C\gamma + A_2C\gamma}. \tag{17}$$

Together the processed load is

$$\alpha_1 + \alpha_2 = \frac{(T - S)(A_1 + A_2) - SA_1 + \beta_2A_2\gamma C}{A_1A_2 + A_1C\gamma + A_2C\gamma}. \tag{18}$$

As in (8) it is a function strictly increasing with β_2. Hence, it is most effective to make $\beta_2 = \alpha_2$, i.e. maximum possible. Consequently, bandwidth sharing while reading results from the two mappers is not profitable. Now we will calculate what load is processed in the LIFO mode in the given T, provided that $\beta_2 = \alpha_2$. From (14)

$$\alpha_1 = \frac{T - S}{A_1 + \gamma C}. \tag{19}$$

By observing that $A_1\alpha_1 = S + (A_2 + \gamma C)\alpha_2$ and using the above value of α_1 we get

$$\alpha_2 = \frac{TA_1 - 2SA_1 - S\gamma C}{(A_1 + \gamma C)(A_2 + \gamma C)} \tag{20}$$

Together the load processed in LIFO mode without bandwidth sharing is

$$\alpha_1 + \alpha_2 = \frac{T(A_1 + A_2) + T\gamma C - 2SA_1 - SA_2 - 2S\gamma C}{(A_1 + \gamma C)(A_2 + \gamma C)}. \tag{21}$$

Comparing (13) and (21) we see that FIFO order of reducer input reading is more profitable because the numerator in (13) is bigger by $S\gamma C$.

It remains to determine the optimum order of starting computation on processors. If we switch the order of activating processors from (P_1, P_2), to

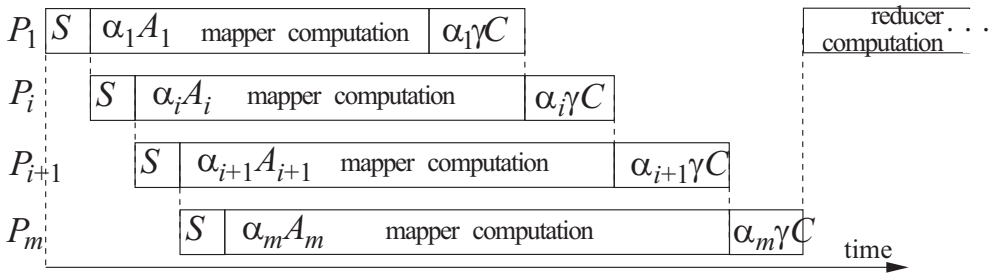

Figure 4: A schedule structure for a single reducer.

(P_2, P_1) then the processor indices in (13) get swapped and the processed load is

$$\alpha_1' + \alpha_2' = \frac{T(A_1 + A_2) + T\gamma C - 2SA_2 - SA_1 - S\gamma C}{(A_1 + \gamma C)(A_2 + \gamma C)} \tag{22}$$

Subtracting $\alpha_1 + \alpha_2$ in equation (13) from $\alpha_1' + \alpha_2'$ in the above equation we get

$$(\alpha_1' + \alpha_2') - (\alpha_1 + \alpha_2) = \frac{SA_1 - SA_2}{(A_1 + \gamma C)(A_2 + \gamma C)}. \tag{23}$$

Thus, the load processed in time T increases after the swap only if $A_1 > A_2$. This means that in the order (P_1, P_2) we would have started computations on a slower processor first. Hence, the faster processor should start computations earlier. □

In the above lemma we demonstrated that sharing bandwidth while reading outputs from the mappers is not profitable both in LIFO, and FIFO order of reading. Of the two orders FIFO is better, and for FIFO faster processor (i.e. with smaller A_i) should be started first. This result can be iteratively extended to more than just two mappers.

4.2 Load Partitioning

In this section we introduce a method for optimally partitioning the load between the mappers if only one reducer exists. Let us remind that from Lemma 1 it follows that the mappers should start computation in the order of increasing A_is, the outputs from the mappers are read sequentially. Let us assume that processors $P_1, \ldots, P_m$ running the mappers are numbered according to increasing A_is, i.e. $A_1 \leq A_2 \leq \ldots \leq A_m$. A schedule for the above setting is shown in Fig.4. From Lemma 1 and from Fig.4 we infer that the time of computing on processor P_i and reading its results is equal to the time of starting and computing on processor P_{i+1}. Hence we get a system of

linear equations determining load partitioning.

$$(A_i + \gamma C)\alpha_i = S + A_{i+1}\alpha_{i+1} \qquad \text{for } i = 1, \ldots, m-1 \tag{24}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{25}$$

The above linear system can be solved in $O(m)$ time for α_is by reduction of α_i to affine linear functions of α_m, i.e. $\alpha_i = l_i + k_i\alpha_m$. More precisely, from (24)

$$l_m = 0 \qquad k_m = 1 \tag{26}$$

$$\begin{aligned} \alpha_i &= \frac{S}{(A_i + \gamma C)} + \frac{A_{i+1}}{(A_i + \gamma C)}\alpha_{i+1} = \\ &= \frac{S}{(A_i + \gamma C)} + \frac{A_{i+1}}{(A_i + \gamma C)}(l_{i+1} + k_{i+1}\alpha_m) = \\ &= l_i + k_i\alpha_m \qquad \text{for } i = m-1, \ldots, 1 \end{aligned} \tag{27}$$

By substituting α_is in (25) we obtain

$$\alpha_m = \frac{V - \sum_{i=1}^{m} l_i}{\sum_{i=1}^{m} k_i} \tag{28}$$

and the remaining α_is are obtained from (28), and (27). Let us note, that α_m in (28) may be negative. This negative solution is a demonstration that, at the current parameters A_i, γ, C, S, V, the number of processors m is too big to use them all. Therefore, if $\alpha_m < 0$ then the number of processors m must be reduced.

In the current case the total schedule length is (cf. Fig.4).

$$T = mS + \alpha_m A_m + t_R = S + \alpha_1 A_1 + \gamma C V + t_R, \tag{29}$$

In the above reasoning we assumed that on average (over all communications from the mappers) there is no advantage in particular reducer placement, and the reducer can be executed on any processor in the cluster. Were it otherwise, the above method of load partitioning can be generalized by using different communication rates instead of C in equations (24).

5 Many Reducers

In this section we tackle load partitioning and communication scheduling for more than one reducer. Unfortunately, a generally optimum schedule structure, similar to the one defined in Lemma 1 for a single reducer, does not

seem to exist for many reducers. Quite contrary, it will be shown in the next subsection that each of the alternative schedule structures with many reducers can dominate the other under certain conditions. As a consequence, we introduce in the following three methods of load partitioning and communication scheduling.

5.1 Schedule Dominance for Multiple Reducers

Here we will show that none of several alternative ways of reading results from the mappers by reducers is generally optimum. As suggested by Lemma 1 we assume FIFO order of finishing computations on the mappers and that a single reducer is not reading results from two (or more) mappers in parallel. As explained in Section 3 the amounts of load read by all reducers are the same. The actual processors running the reducers can be arbitrary free machines. For example, P_1 can execute reducer 1 after completion of mapper 1, or it can be some other processor from a separate computer pool if such pool exists.

The alternative communication schedules are shown in Fig.5. In the first schedule type (Fig.5a) the end of reading results from P_i by the first reducer is synchronized with the end of computations on P_{i+1}. Note that in this schedule different reducers read different mapper results in parallel which may violate bisection width limit. For the time being, we assume that bisection width is not exceeded. We will call this schedule type case A. In the second type of schedule (Fig.5b) reducers read output from the mappers sequentially. The end of reading output by the last reducer from mapper P_i coincides with the end of computation on mapper P_{i+1}. Here all reducer reads are sequential, only one communication channel is used at the time therefore the speed of communication is the same as in one-to-one communication without network contention. We will refer to the second type of schedule as to case B. In the third type of the schedule (Fig.5c) reducers read results from mappers one by one, but bandwidth is equally shared between the reducers. The end of reading from mapper P_i coincides with the end of computation on mapper P_{i+1}. This case is very similar to Case B. Hence, we do not analyze it separately in the further discussion.

To demonstrate lack of dominance of the above communication schedule structures we will calculate the amount of load processed on two mappers and transferred to two reducers ($m = r = 2$) in time T. Note that since execution time of reducers is equal, minimization of T is equivalent to the minimization of the whole schedule length.

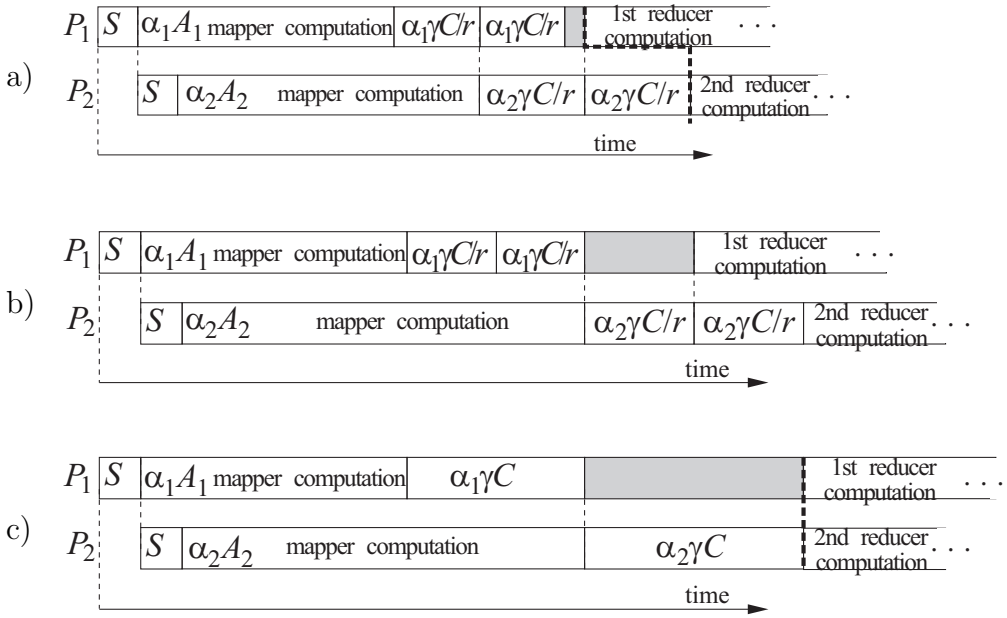

Figure 5: Many reducers exemplary read schedule structures. a) case A, reducers read in parallel, b) case B, reducers read sequentially, c) reducers share bandwidth.

Case A. We can distinguish two sub-cases (Fig.6). In the first one (case A.1) there is an idle time in the communications with P_2. This means that reading results from P_1 is longer than from P_2. Hence $\alpha_1 \geq \alpha_2$. In the second sub-case (case A.2) communication with P_2 is longer than with P_1, and $\alpha_1 \leq \alpha_2$.

Case A.1. In the first sub-case we have conditions:

$$\alpha_1(A_1 + \gamma C/2) = \alpha_2 A_2 + S \tag{30}$$

$$S + \alpha_1(A_1 + \gamma C) + \alpha_2\gamma C/2 = T \tag{31}$$

$$\alpha_1 \geq \alpha_2 \tag{32}$$

From which we obtain:

$$\alpha_1 = \frac{TA_2 - A_2S + \gamma CS/2}{A_1A_2 + A_1\gamma C/2 + A_2\gamma C + \gamma^2C^2/4} \tag{33}$$

$$\alpha_2 = \frac{TA_1 + T\gamma C/2 - 2A_1S - 3/2\gamma CS}{A_1A_2 + A_1\gamma C/2 + A_2\gamma C + \gamma^2C^2/4} \tag{34}$$

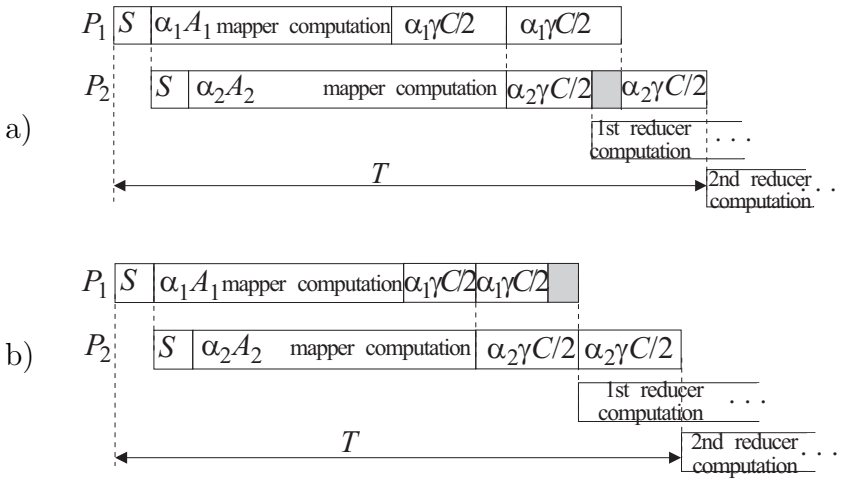

Figure 6: Special cases of the first reducers read orders. a) Case A.1,b) Case A.2.

$$\alpha_1 + \alpha_2 = \frac{T(A_1 + A_2 + \gamma C/2) - S(2A_1 + A_2) - \gamma CS}{A_1A_2 + A_1\gamma C/2 + A_2\gamma C + \gamma^2C^2/4}, \tag{35}$$

with additional requirement $\alpha_1 \geq \alpha_2$ equivalent to:

$$T(A_2 - A_1 - \gamma C/2) \geq A_2S - 2A_1S - 2\gamma CS \tag{36}$$

Case A.2. In the second sub-case we have conditions:

$$\alpha_1(A_1 + \gamma C/2) = \alpha_2 A_2 + S \tag{37}$$

$$2S + \alpha_2(A_2 + \gamma C) = T \tag{38}$$

$$\alpha_1 \leq \alpha_2 \tag{39}$$

From which we obtain:

$$\alpha_1 = \frac{TA_2 - A_2S + \gamma CS}{A_1A_2 + A_1\gamma C + A_2\gamma C/2 + \gamma^2C^2/2} \tag{40}$$

$$\alpha_2 = \frac{TA_1 + T\gamma C/2 - 2A_1S - \gamma CS}{A_1A_2 + A_1\gamma C + A_2\gamma C/2 + \gamma^2C^2/2} \tag{41}$$

$$\alpha_1 + \alpha_2 = \frac{T(A_1 + A_2 + \gamma C/2) - S(2A_1 + A_2)}{A_1A_2 + A_1\gamma C + A_2\gamma C/2 + \gamma^2C^2/2}, \tag{42}$$

with additional requirement $\alpha_1 \leq \alpha_2$ equivalent to:

$$T(A_2 - A_1 - \gamma C/2) \leq A_2S - 2A_1S - 4\gamma CS/2 \tag{43}$$

At least one of these conditions (36), (43) is always satisfied. If both are satisfied, then the load amounts given by (35) and (42) are equal.

Case B. In the current schedule structure we have conditions (cf. Fig.5b):

$$\alpha_1(A_1 + 2\gamma C/2) = \alpha_2 A_2 + S \tag{44}$$

$$2S + \alpha_2(A_2 + 2\gamma C/2) = T \tag{45}$$

From which we obtain:

$$\alpha_1 = \frac{TA_2 - A_2S + \gamma CS}{A_1A_2 + A_1\gamma C + A_2\gamma C + \gamma^2C^2} \tag{46}$$

$$\alpha_2 = \frac{TA_1 + T\gamma C - 2A_1S - 2\gamma CS}{A_1A_2 + A_1\gamma C + A_2\gamma C + \gamma^2C^2} \tag{47}$$

$$\alpha_1 + \alpha_2 = \frac{T(A_1 + A_2 + \gamma C) - S(2A_1 + A_2) - \gamma CS}{A_1A_2 + A_1\gamma C + A_2\gamma C + \gamma^2C^2}. \tag{48}$$

Let us now compare the amounts of load processed in time T in the above analyzed schedule structures. By comparing (35), (42), (48) we can see that none of the schedule structures always results in the biggest processing load for the given time T. Thus, no single communication schedule structure seems to be optimum in all cases. To build the optimum schedule a more general tool, possibly incorporating all possible structures, must be applied. On the other hand, if we concentrate only on the part of (35), (42), (48) which grows with T, then it can be concluded that for very big T (which may result from a need for processing very big loads) the load processed in cases A.1, A.2 dominate case B. For example, the difference between (35) and (48) in the part proportional to T is

$$\frac{T(A_1+A_2+\gamma C/2)}{A_1A_2+A_1\gamma C/2+A_2\gamma C+\gamma^2C^2/4} - \frac{T(A_1+A_2+\gamma C)}{A_1A_2+A_1\gamma C+A_2\gamma C+\gamma^2C^2} = \frac{T\gamma C/2(A_1^2+3/2A_1\gamma C+A_2\gamma C/2+\gamma^2C^2/2)}{(A_1A_2+A_1\gamma C/2+A_2\gamma C+\gamma^2C^2/4)(A_1A_2+A_1\gamma C+A_2\gamma C+\gamma^2C^2)} > 0 \tag{49}$$

Similar inequality can be derived for (42) and (48). Therefore, in the further discussion we will be using schedules based on case A.

5.2 Load Partitioning

In this section we propose methods of load partitioning between mappers taking into account many reducers. Since reducers have equal execution time t_R we concentrate on minimizing length of the partial schedule comprising mapper computations, and mapper to reducer transmissions. Features of the methods are discussed at the end of this section.

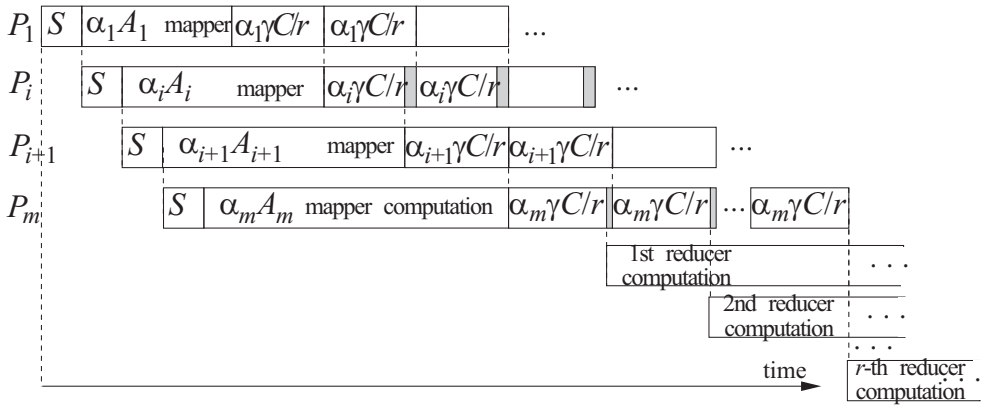

Figure 7: A schedule for many reducers. The first method.

The first method assumes simple limits on communication bandwidth and is a natural extension of the method for a single reducer. Schedule structure is shown in Fig.7. In this method the end of the read by the first reducer from P_i coincides with mapper completion time on P_{i+1}. The method of calculating $\alpha_1, \ldots, \alpha_m$ for $r = 1$ presented in Section 4.2 can be applied here with using $\gamma C/r$ in place of γC. None of the mappers is read simultaneously by many reducers, and no reducer reads outputs from many mappers in parallel. The bandwidth of mappers' network output interfaces, and reducers' network input interfaces are not shared. Hence, we assume constant communication rate C. Yet, the bisection width limitations are ignored. Schedule length is

$$T = \max_{i=1}^{m}\{iS + \alpha_i(A_i + \gamma C) + \gamma C/r \sum_{j=i+1}^{m} \alpha_j\} + t_R \tag{50}$$

The second method more carefully represents bandwidth sharing in transferring the results from mappers to reducers (see Fig.8). Let $t_1 \le t_2 \le \ldots \le t_m$ be the time moments when mappers $P_1, \ldots, P_m$ finish computations, respectively. Let t_{m+1} be the time moment when all mapper to reducer communications finish. We will denote by β_{ijk} the amount of results read in interval $[t_i, t_{i+1})$ from mapper j by reducer k. The partitioning of the reducer reads in the time intervals $[t_i, t_{i+1}]$ can be found from the following linear program:

$$\text{minimize} \quad t_{m+1} \tag{51}$$

$$iS + A_i\alpha_i = t_i \quad \text{for} \quad i = 1, \ldots, m \tag{52}$$

$$C\sum_{j=1}^{i} \beta_{ijk} \le t_{i+1} - t_i \text{ for } i = 1, \ldots, m, k{=}1, \ldots, r \tag{53}$$

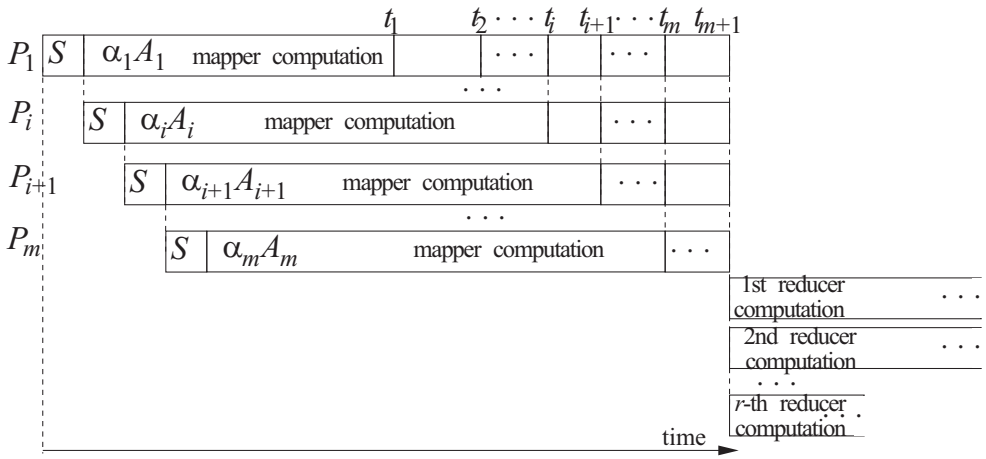

Figure 8: A schedule for many reducers. The second method.

$$C \sum_{k=1}^{r} \beta_{ijk} \leq t_{i+1} - t_i \text{ for } i = 1, \ldots, m,\ j = 1, \ldots, i, \tag{54}$$

$$C \sum_{j=1}^{i} \sum_{k=1}^{r} \beta_{ijk} \leq l(t_{i+1} - t_i) \quad \text{for} \quad i = 1, \ldots, m \tag{55}$$

$$\beta_{ijk} = 0 \quad \text{for } i = 1, \ldots, m-1, j = i+1, \ldots, m,\ k = 1, \ldots, r \tag{56}$$

$$\sum_{i=1}^{m} \beta_{ijk} = \frac{\gamma}{r} \alpha_j \quad \text{for} \quad k = 1, \ldots, r,\ j = 1, \ldots, m \tag{57}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{58}$$

In the above linear program $\alpha_i, \beta_{ijk}, t_i$ are variables. We minimize t_{m+1} which is length of the schedule from the start until the end of mapper-reducer communications. Equations (52) guarantee that communications may start after mapper computations. By (53), (54) computations finish in the FIFO order. Inequalities (53) – (56) ensure a feasible schedule of the communications in each interval. In particular, (53) guarantee that no reducer communicates longer than the communication interval. Similarly, by (54) no mapper communicates longer than the communication interval. By (55) the total used bandwidth in mapper-reducer communications is equivalent to at most l simultaneous connections with communication rate C. This is equivalent to bisection width $\frac{l}{C}$. Inequalities (56) guarantee that no data is read from mappers which have not completed their computations. By equations (57) each reducer reads all its inputs, and all work is done by (58). The number of variables in the above LP is $rm^2 + 2m + 1$.

A feasible communication schedule can be built for each interval $[t_i, t_{i+1})$

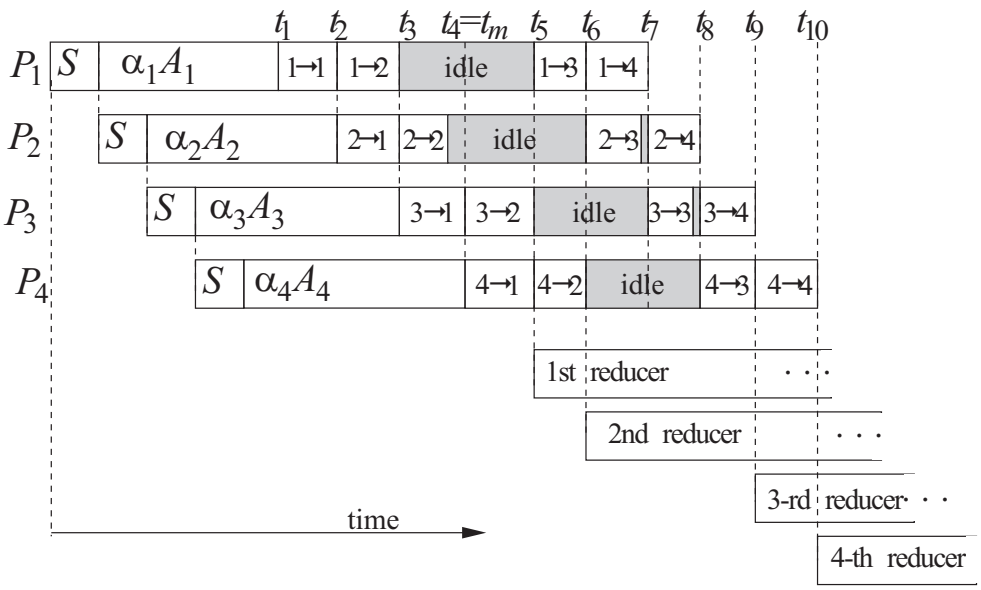

Figure 9: A schedule for many reducers. The third method. Notation: $i \to j$ - reducer j reads mapper i. Here $m = r = 4, l = 2$.

in which communication from mapper j to reducer k of length $\beta_{ijk}C$ is made using the so-called two-stage method for problem $R|pmtn|C_{max}$ [5, 7].

The third method simplifies the second approach (see Fig.9). It presumes a simple assignment of reducer communications to time intervals. This simplification allows for introducing bisection width limitation as well as sequential read of mapper results by the reducers. We assume here that in each interval $[t_i, t_{i+1}]$ complete load transfers are made (i.e. complete set of results of size $\gamma\alpha_j/r$ is read from any mapper P_j), any mapper and any reducer appears at most once. All reducers read mappers in the same order: $P_1, P_2, \ldots, P_m$. Also the order of reducer read operations on the mappers is the same for all read mappers. Thus, it can be said that reducer read operations are ordered as in the permutation flowshop. New communication operations are started as soon as mappers finish their computations, and sufficient number of communication channels (not exceeding bisection width l) is available.

Let us analyze the number of necessary communication intervals. If the number l of channels which can be simultaneously opened is at least equal the number of reducers r, then the number of intervals necessary to perform m reads by each of r reducers is $m + r - 1$. On the other hand, if the number of simultaneous reads $l < r$ then after opening l read channels by l reducers

the $(l+1)$-th reducer shall wait until completion of the read operation of the first reducer from the m-th mapper. This requires $m-l$ additional communications for reducer 1 with mappers $P_{l+1},\ldots,P_m$. Consequently, $m-l$ idle intervals appear in the reads from each mapper. With the end of each interval $[t_m,t_{m+1}],\ldots,[t_{m+l},t_{m+l+1}]$ a new read operation is started by reducers $l+1,\ldots,l+l$. Thus, after $m-l$ idle intervals, read operation are performed in the following l intervals. Sequences of $m-l$ idle intervals are inserted in the schedule $\lceil \frac{r}{l} \rceil - 1$ times. Overall there are $(\lceil \frac{r}{l} \rceil - 1)(m-l) + m + r - 1$ intervals in the communication schedule. For simplicity of notation let us introduce function $itv(i,j)$ which returns the number of the interval in which reducer j reads output of mapper i (counting starts with value 1 for interval $[t_1,t_2]$). Values of $itv(i,j)$ can be calculated as follows:

$$itv(i,j) = \left(\left\lceil \frac{j}{l} \right\rceil - 1 \right) m + i + (j-1) \bmod l \tag{59}$$

for $i=1,\ldots,m, j=1,\ldots,r$. Let $vti(i)$ be the set of read operations performed in interval i, i.e.

$$vti(i) = \{a : itv(a,b) = i, b \in \{1,\ldots,r\}\} \tag{60}$$

Values of $vti(i)$ can be tabulated in $O(mr)$ time. The partition of the load can be calculated from the following linear program:

$$\text{minimize} \quad t_{itv(m,r)+1} \tag{61}$$

$$iS + A_i\alpha_i = t_i \quad \text{for} \quad i=1,\ldots,m \tag{62}$$

$$\frac{\gamma C}{r}\alpha_k \le t_{i+1} - t_i \quad \text{for} \quad i=1,\ldots,itv(m,r),\ k \in vti(i) \tag{63}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{64}$$

In the above linear program α_i, t_i are variables. We minimize the completion time of the last communication $t_{itv(m,r)+1}$. By inequalities (62) computations finish before starting reads from the mappers. Constraints (63) guarantee that all communications fit in the time intervals where they are assigned. LP (61)-(64) has $itv(m,r)+1+m$ variables which is $O(mr/l)$, and at most $m+1+itv(m,r)l$ constraints which is $O(mr)$.

The above linear program can be further simplified. Let us remind that reducers read equal size outputs from each mapper. For example, all communications (i,j) from fixed P_i to reducers $j=1,\ldots,r$ have (approximately) the same size $\gamma\alpha_i/r$. Consequently, intervals $[t_{ma+i},t_{ma+i+1}]$, and $[t_{mb+i},t_{mb+i+1}]$ have equal length because they comprise read operations from the same mappers, for some positive integers $i, a<b$ such that $mb+i \le itv(m,r)$. The

block of intervals $[t_m, t_{m+1}], \ldots, [t_{2m-1}, t_{2m}]$ is repeated $(\lceil \frac{r}{l} \rceil - 1)$ times. After them $(r-1) \bmod l$ intervals follow which repeat the length of some earlier intervals. More precisely, the distance between $t_{itv(m,(\lceil \frac{r}{l} \rceil - 1)l+1)}$ and $t_{itv(m,r)+1}$ is equal to the distance between t_m and $t_{m+((r-1) \bmod l)+1}$. Consequently, length of the schedule until the end of mapper-reducer communications is

$$t_m + (\lceil \frac{r}{l} \rceil - 1)(t_{2m} - t_m) + (t_{m+((r-1) \bmod l)+1} - t_m) = \tag{65}$$

$$= \quad (\lceil \frac{r}{l} \rceil - 1)(t_{2m} - t_m) + t_{m+((r-1) \bmod l)+1} \tag{66}$$

Value of variable t_{am+i} for $1 < a \leq \lceil \frac{r}{l} \rceil - 1$, and $0 \leq i < m$ can be calculated as $t_m + a(t_{2m} - t_m) + (t_{m+i} - t_m) = a(t_{2m} - t_m) + t_{m+i}$. Hence, LP (61)-(64) can be reduced to

$$\text{minimize} \quad (\lceil \frac{r}{l} \rceil - 1)(t_{2m} - t_m) + t_{m+((r-1) \bmod l)+1} \tag{67}$$

$$iS + A_i\alpha_i = t_i \quad \text{for} \quad i = 1, \ldots, m \tag{68}$$

$$\frac{\gamma C}{r}\alpha_k \leq t_{i+1} - t_i \quad \text{for} \quad i = 1, \ldots, 2m, k \in vti(i) \tag{69}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{70}$$

The functions of the constraints in the above LP are the same as in the earlier one. The number of variables is $3m$, the number of constraints is at most $2ml + m + 1$. The objective function (67) reduces to t_{m+r} if $r \leq l$.

The above three methods have advantages and disadvantages. The first model is mathematically simple and easy to implement in practice. On the other hand it ignores network bisection width. The second model is more precise in representing bandwidth limitations, but it uses more demanding mathematical tools (note big number of variables in LP (51)-(58)) and its results are hard to implement in practice (because a lot of information must be distributed, and schedule of many processors must be coordinated, i.e. synchronized). Moreover, in the second method all the reducers finish computations simultaneously, and consequently simultaneously attempt to store their results in the distributed file system. These final reducer writing operations may be delayed by limited performance of the distributed file system. The third method is a compromise between the earlier two: it has small computational demands (smaller number of variables in LP (67)-(70)), takes network bisection width into account. On the other hand, it makes strong (though not unrealistic) assumptions on the structure of the schedule, requires distributing some scheduling information (yet less than in the second method) and coordination. Moreover, since reducer computations start times

are spread in time, then also reducer writes are spread in time to avoid bisection width limitation.

6 Conclusions

In this report we analyzed dominance properties of the schedules for divisible MapReduce computations. It has been show that for a single reducer activating faster mapper first, with sequential FIFO reducer reads is optimum schedule structure. Unfortunately, it is also shown that for many reducers the schedule dominance is dependent on the parameters of the system. Three methods of calculating load partitioning have been proposed.

References

[1] R. Agrawal, H.V. Jagadish, Partitioning Techniques for Large-Grained Parallelism, *IEEE Transactions on Computers* **37**(12), 1627-1634, 1988.

[2] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, Y. Yang, Scheduling divisible loads on star and tree networks: results and open problems, *IEEE Transactions on Parallel and Distributed Systems* **16**(3), 207-218, 2005.

[3] J. Berlińska, M. Drozdowski, M. Lawenda, Experimental study of scheduling with memory constraints using hybrid methods, *Journal of Computational and Applied Mathematics* **232**(2), 638-654.

[4] V.Bharadwaj, D.Ghose, V.Mani, T.Robertazzi, *Scheduling divisible loads in parallel and distributed systems*. IEEE Computer Society Press: Los Alamitos CA, 1996.

[5] J. Błażewicz, W. Cellary, R. Słowiński, J. Węglarz, *Scheduling under resource constraints - deterministic models*, Annals of Operations Research, vol.7, 1986.

[6] J. Błażewicz, M. Drozdowski, Scheduling divisible jobs on hypercubes, *Parallel Computing*, **21**, 1945-1956, 1995.

[7] J. Błażewicz, K.Ecker, E.Pesch, G.Schmidt, J.Węglarz, *Scheduling Computer and Manufacturing Processes*, Springer-Verlag: Heidelberg, 1996

[8] Y.-C.Cheng, T.G.Robertazzi, Distributed computation with communication delay. *IEEE Transactions on Aerospace and Electronic Systems* **24**, 700–712, 1988.

[9] Y.-C. Cheng, T. G. Robertazzi, Distributed Computation for a Tree Network with Communication Delays, *IEEE Transactions on Aerospace and Electronic Systems* **26**, 511-516, 1990.

[10] N. Comino, V.L. Narasimhan, A novel data distribution technique for host-client type parallel applications. *IEEE Transactions on Parallel and Distributed Systems*, 13(2):97–110, 2002.

[11] 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 `http://labs.google.com/papers/mapreduce.html`.

[12] M.Drozdowski, Scheduling for Parallel Processing, Springer, 2009 (to appear), ISBN: 978-1-84882-309-9.

[13] M. Drozdowski, W. Głazek, Scheduling divisible loads in a three-dimensional mesh of processors, *Parallel Computing*, **25**, 381-404, 1999.

[14] M. Drozdowski, P. Wolniewicz. Experiments with scheduling divisible tasks in clusters of workstations. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, Proceedings of 6th Euro-Par Conference. LNCS, vol. 1900, 311–319, 2000.

[15] D. Ghose, H.J. Kim, Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays, *Journal of Parallel and Distributed Computing*, 55(1):32–59, 1998.

[16] X. Li, V. Bharadwaj, C.C. Ko, Processing divisible loads on single-level tree networks with buffer constraints, *IEEE Transactions on Aerospace and Electronic Systems* **36**, 1298 - 1308, 2000.

[17] X. Li, V. Bharadwaj, C.C. Ko, Distributed image processing on a network of workstations, *International Journal of Computers and Applications*, 25(2):1–10, 2003.

[18] T. Lim and T.G. Robertazzi. Efficient parallel video processing through concurrent communication on a multi-port star network. *Proceedings of the 40th Conference on Information Sciences and Systems*, 2006.

[19] K. van der Raadt, Y. Yang, H. Casanova. Practical divisible load scheduling on grid platforms with APST-DV. *Proceedings of the 19th IPDPS'05*, page 29.b, 2005.

[20] T.Robertazzi, Ten reasons to use divisible load theory. *IEEE Computer* **36**, 63-68, 2003.

[21] J.Sohn, T.G.Robertazzi, S.Luryi, Optimizing Computing Costs Using Divisible Load Analysis, *IEEE Transactions on Parallel and Distributed Systems* **9**, 225-234, 1998.

[22] Y. Yang, H. Casanova, M. Drozdowski, M. Lawenda, A. Legrand, On the Complexity of Multi-Round Divisible Load Scheduling, INRIA Rhône-Alpes, Research Report 6096, 2007, `http://hal.inria.fr/inria-00123711/en/`.

[23] Wikipedia, MapReduce, `http://en.wikipedia.org/wiki/MapReduce`, [Online; accessed 16-July-2009].