Distributed Offline Load Balancing in MapReduce Networks

Themistoklis Charalambous, Evangelia Kalyvianaki, Christoforos N. Hadjicostis and Mikael Johansson

***Abstract*— In this paper we address the problem of balancing the processing load of MapReduce tasks running on heterogeneous clusters, i.e., clusters composed of nodes with different capacities and update cycles. We present a fully decentralized algorithm, based on ratio consensus, where each mapper decides the amount of workload data to handle for a single user job using only job specific local information, i.e., information that can be collected from directly connected neighboring mappers, regarding their current workload usage and capacity. In contrast to other algorithms in the literature, the proposed algorithm can be deployed in heterogeneous clusters and can operate asynchronously in both directed and undirected communication topologies. The performance of the proposed algorithm is demonstrated via simulation experiments on large-scale strongly connected topologies.**

I. Introduction

Cloud computing is being rapidly adopted since it enables quick application deployment over a potentially very large pool of commodities with little capital investment and with moderate operating costs. One of the fast emerging Cloud applications is *big-data* computing that handles massive-volumes of data. Modern applications in this domain employ the *MapReduce* paradigm [1] to execute scalable and parallel processing of data, leveraging on the scalable infrastructure of Cloud data centers. For example, Google uses MapReduce to execute popular applications such as web search indexing, Google News and Google Maps [2]. Facebook relies on the MapReduce paradigm to perform data warehousing operations [3].

MapReduce is designed for scalable and parallel processing of large volumes of data by dividing the input data into smaller chunks and distributing its processing into a network of processing elements. It employs a programming model that transforms all input data into key/value pairs for ease of processing. Input data is processed by two successively executed phases: the *map* and the *reduce*, also shown in Figure 1. In the *map phase*, each of the input data chunks is processed independently by a *mapper* to produce a new list of key/value pairs. In the *reduce* phase, all pairs with common keys generated in the *map* phase are merged into a final list of key/values pairs. Scalability is achieved by distributing the workload, first within the network of mappers

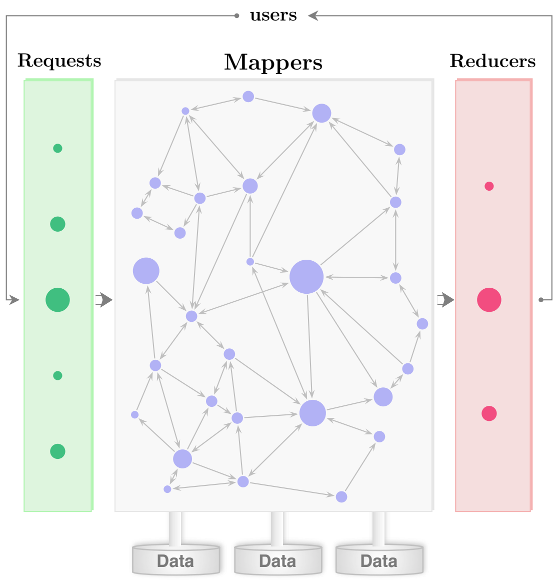

Fig. 1. MapReduce model: a user request is first processed by a network of mappers forming a digraph. The result from each mapper is then sent to the reducers to form the final answer. The area of circles represent the volume of data chunks assigned to mappers and reducers. Arrows among mappers represent the exchange of *coordinating* information.

and then within the network of reducers; different mappers and different reducers can be executed in parallel.

MapReduce is designed to achieve high-job completion speedups through parallelism. However, in practice, a job might be significantly prolonged due to imbalances in workload processing, especially in heterogeneous clusters of nodes [4], [5]. Initial work on MapReduce [1] assumed homogeneous clusters where mappers and reducers are running on nodes with equal processing capacities. In this case, and when the input data set is divided across equally-sized chunks, the completion time of a MapReduce job roughly equals the sum of completion times of a *typical* mapper and a *typical* reducer. However, in several cases, job completion times were observed to be higher than expected because of *stragglers*, i.e., a mapper (or a reducer) that takes significantly longer to complete, even in homogeneous clusters, because of hardware or software misconfigurations [1]. With the wide adoption of the MapReduce paradigm in large-scale heterogeneous clusters [6], others have also observed the existence of stragglers with dramatic effects in performance and losses in revenue [4], [5]. Additional reasons that cause jobs to take longer than usual include heterogeneous clusters in the Cloud [4] and data-based imbalances in the workload

Themistoklis Charalambous and Mikael Johansson are with the School of Electrical Engineering, Royal Institute of Technology (KTH), Stockholm, Sweden. E-mails: {themisc,mikaelj}@kth.se.

Evangelia Kalyvianaki is with the School of Informatics, City University London, UK. E-mail: evangelia.kalyvianaki.1@city.ac.uk.

Christoforos N. Hadjicostis is with the Department of Electrical and Computer Engineering at the University of Cyprus, Nicosia, Cyprus. E-mail: chadjic@ucy.ac.cy.

distribution and transfer of data across racks [5]. Therefore, it is important that the workload distribution among mappers and reducers is balanced and the transfer of data across racks is minimized.

Different approaches exist to mitigate the effects of stragglers at run-time after the detection of a delayed mapper or reducer task. The most common approach replicates the execution of data processed by stragglers using back-up [1] and speculative execution [4] to re-process the data chunks assigned to stragglers. Ananthanarayanan *et al.* [5] present an alternative system that combines specialized solutions based on root-cause analysis of outliers, network-aware placements, and duplication. Ahmad *et al.* [6] take on-line decisions to balance the communication of the mapping phase based on cluster measurements. Others employ centralized solutions to map tasks or data to nodes prior to application execution. For example, Alma *et al.* [7] build a job scheduler that assigns MapReduce tasks to nodes in order to meet pre-defined job completion deadlines. Similarly, Xie *et al.* [8] partition data to nodes for balanced load processing in heterogeneous clusters.

In this paper we present a distributed algorithm to address the problem of balancing the data workload assigned to mappers in *massively* large-scale clusters that consist of several tens of thousands of nodes [9]. Our goal is to address the load-balancing problem in heterogeneous networks of clusters *before* a job commences to process data. In this way, we avoid the increased network communication overhead resulting from balancing the load at run-time, as measured by [6], and allow for advanced data transfers to nodes prior to execution. Our algorithm works in the following way. When a new MapReduce job arrives, the mappers commence to exchange *coordinating* information locally (with neighboring mappers with which they have a communication link established) about their total job workload demand and capacity, as shown in Figure 1. More specifically, a ratio consensus algorithm is deployed which enables (synchronous and asynchronous) asymptotic convergence to proportional balance among mappers of the data to be processed, in a completely distributed fashion. With proportional balancing, each mapper is assigned workload proportional to its resource availability (which in general could be time-varying). In this way, we expect all mappers to finish processing simultaneously with minor variations, thus preventing increased processing time due to imbalances.

The proposed distributed algorithm has the following advantages over other suggested methods (e.g., Gonzalez-Ruiz *et al.* [10]):

(i) The assumption that cluster nodes need to be homogeneous and have the same resource capacity is lifted. We allow for heterogeneous nodes and the load is distributed according to the nodes' capacities (in terms of CPU, memory, disk space, etc.). Also, a network topology might be weighted due to different bandwidth capabilities or time-varying delays.

(ii) The link-level delay in the exchange of loads between nodes needs neither to be negligible nor synchronous. Our algorithm is shown to exhibit asymptotic converge, even in the presence of delays, provided that the delays are bounded.

(iii) The assumption that the network is undirected is lifted and we consider directed graphs (digraphs). In practical scenarios, the network topology might become loosely directed because of delays, packet losses and asymmetric links. In addition, the directed graph structure reduces overhead communication (e.g., there is no need for acknowledgements) and can admit faster convergence.

(iv) The distributed consensus algorithm itself constitutes a novel distributed solution to the problem of workload balancing in heterogeneous large-scale MapReduce clusters.

We perform simulations over small- and large-scale strongly connected topologies of mappers. Results show that, unlike other algorithms, our algorithm converges asymptotically to proportional balancing in both synchronous and asynchronous settings.

The remainder of this paper is organized as follows. In Section II we provide necessary notation and background on graph properties. Section III presents the model adopted, along with theoretical results that support the use of the suggested algorithm. In Section IV we formalize the distributed algorithm for the load balancing problem in heterogeneous MapReduce networks. Section V presents preliminary results with numerical examples from simulations. Finally, Section VI presents concluding remarks and future directions.

II. Notation

The sets of real, integer and natural numbers are denoted by $\mathbb{R}$, $\mathbb{Z}$ and $\mathbb{N}$, respectively; their positive orthant is denoted by the subscript $+$ (e.g., $\mathbb{R}_+$). Vectors are denoted by small letters whereas matrices are denoted by capital letters. A^T denotes the transpose of matrix A. By $\mathbb{1}$ we denote the all-ones vector and by I we denote the identity matrix (of appropriate dimensions). A matrix whose elements are nonnegative, called nonnegative matrix, is denoted by $A \geq 0$ and a matrix whose elements are positive, called positive matrix, is denoted by $A > 0$.

Let the exchange of information between nodes (where mappers are located) be modeled by a weighted directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, P)$ of order n ($n \geq 2$), where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $P = [p_{ji}] \in \mathbb{R}_+^{n \times n}$ is a weighted $n \times n$ adjacency matrix where p_{ji} are nonnegative elements. A directed edge from node v_i to node v_j is denoted by $\varepsilon_{ji} = (v_j, v_i) \in \mathcal{E}$, which represents a directed information exchange link from node v_i to node v_j, i.e., it denotes that node v_j can receive information from node v_i. A directed edge $\varepsilon_{ji} \in \mathcal{E}$ if and only if $p_{ji} > 0$. The graph is undirected if and only if $\varepsilon_{ji} \in \mathcal{E}$ implies $\varepsilon_{ij} \in \mathcal{E}$.

All nodes that can transmit information to node v_j directly are said to be in-neighbors of node v_j and belong to the set $\mathcal{N}_j^- = \{v_i \in \mathcal{V} : \varepsilon_{ji} \in \mathcal{E}\}$. The cardinality of $\mathcal{N}_j^-$, is called the *in-degree* of v_j and it is denoted by $\mathcal{D}_j^- = |\mathcal{N}_j^-|$. The nodes that receive information from node v_j are called out-neighbors of node v_j and belong to the set $\mathcal{N}_j^+ = \{v_l \in \mathcal{V} : \varepsilon_{lj} \in \mathcal{E}\}$. The cardinality of $\mathcal{N}_j^+$, is called the *out-degree* of v_j and it is denoted by $\mathcal{D}_j^+ = |\mathcal{N}_j^+|$. A directed graph is called *strongly* connected if there exist paths from each vertex in the graph to every other vertex. This means paths in each direction: a path from v_i to v_j and vice versa, for all $v_i, v_j \in \mathcal{V}$.

III. Model and Application of Theoretical Results

In this section, we establish some basic notions that are needed for the development of our algorithm. In the MapReduce paradigm, where mappers (nodes) can exchange information via interconnection links (edges), the exchange of information between mappers can be captured by a digraph (directed graph) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Each node v_j (mapper) updates and sends its information regarding its current load and capacity to its out-neighbors (and also receives similar information from its in-neighbors) at discrete times $t_0, t_1, t_2, \ldots$. We index nodes' information states and any other information at time t_k by k. We use $x_j[k] \in \mathbb{R}$ to denote the information state (or estimate) of node v_j at time t_k. When there exist no delays in the communication links, each node updates its information state $x_j[k]$ by combining the available information received by its neighbors $x_i[k]$ ($v_i \in \mathcal{N}_j^-$) using a weighted linear combination. More specifically, the positive weights $p_{ji}[k]$ capture the weight of the information inflow from agent v_i to agent v_j at time k (note that unspecified weights in P correspond to pairs of nodes (v_j, v_i) that are not connected and are set (without loss of generality) to zero, i.e. $p_{ji}[k] = 0$, $\forall \varepsilon_{ji} \notin \mathcal{E}$). In this work, we assume that each node v_j can choose its self-weight and the weights on its out-going links ε_{lj}, $v_l \in \mathcal{N}_j^+$, only. In its general form, each node updates its information state according to the following relation:

$$x_j[k+1] = p_{jj}[k]x_j[k] + \sum_{v_i \in \mathcal{N}_j^-} p_{ji}[k]x_i[k] \ , \quad k \geq 0 \ , \ (1)$$

where $x_j[0] \in \mathbb{R}$ is the initial state of node v_j. If we let $x[k] = (x_1[k] \ \ x_2[k] \ \ \ldots \ \ x_n[k])^T$ and $P[k] = [p_{ji}[k]] \in \mathbb{R}_+^{n \times n}$, then (1) can be written in matrix form as

$$x[k+1] = P[k]x[k], \quad (2)$$

where $x[0] = (x_1[0] \ \ x_2[0] \ \ \ldots \ \ x_n[0])^T \equiv x_0$. In this work, we consider a static network (as it is usually the case for distributed resources in applications such as the Cloud) and hence the graph remains invariant. In this case, the weights can be chosen to be constant for all times k (i.e., $p_{ji}[k] = p_{ji} \ \forall k$), and equation (2) can be expressed as $x[k+1] = Px[k]$.

We will be interested in having the nodes reach asymptotic average consensus, i.e., be able to calculate (for large k) the average of their initial values. In other words, we would like

$$\lim_{k \to \infty} x_j[k] = \frac{\sum_{\ell=1}^{n} x_\ell[0]}{n} \ , \forall v_j \in \mathcal{V} \ . \quad (3)$$

Ratio consensus. In [11], an algorithm is suggested that solves the average consensus problem in a directed graph in which each node v_j distributively sets the weights on its self-link and outgoing-links to be $p_{lj} = \frac{1}{1+\mathcal{D}_j^+}$, $\forall (v_l, v_j) \in \mathcal{E}$, so that the resulting weight matrix P is column stochastic, but not necessarily row stochastic. Average consensus is reached by using this weight matrix to run two iterations with appropriately chosen initial conditions. The algorithm is stated below for the specific choice of weights mentioned above (which assumes that each node knows its out-degree). Note, however, that *the algorithm also works for any set of weights that adhere to the graph structure and form a primitive column stochastic weight matrix.*

Lemma 1: [11] Consider a strongly connected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Let $y_j[k]$ and $z_j[k]$ (for all $v_j \in \mathcal{V}$ and $k = 0, 1, 2, \ldots$) be the result of the iterations

$$y_j[k+1] = p_{jj}y_j[k] + \sum_{v_i \in \mathcal{N}_j^-} p_{ji}y_i[k] \ , \quad (4a)$$

$$z_j[k+1] = p_{jj}z_j[k] + \sum_{v_i \in \mathcal{N}_j^-} p_{ji}z_i[k] \ , \quad (4b)$$

where $p_{lj} = \frac{1}{1+\mathcal{D}_j^+}$ for $v_l \in \mathcal{N}_j^+$ (zeros otherwise), and the initial conditions are $y[0] = (y_0(1) \ \ y_0(2) \ \ \ldots \ \ y_0(|\mathcal{V}|))^T \equiv y_0$ and $z[0] = \mathbb{1}$. Then, the solution to the average consensus problem can be asymptotically obtained as

$$\lim_{k \to \infty} \mu_j[k] = \frac{\sum_{v_\ell \in \mathcal{V}} y_0(\ell)}{|\mathcal{V}|} \ , \quad \forall v_j \in \mathcal{V} \ ,$$

where $\mu_j[k] = y_j[k]/z_j[k]$.

Robustified ratio consensus. An adaptation of the above approach to a protocol where each node updates its information state $x_j[k+1]$ by combining the available (possibly delayed) information received by its neighbors $x_i[s]$ ($s \in \mathbb{Z}, s \leq k, \ v_i \in \mathcal{N}_j^-$) using constant positive weights p_{ji} was developed in [12]. Integer $\tau_{ji}[k] \geq 0$ is used to represent the delay of a message sent from node v_i to node v_j at time instant k. We require that $0 \leq \tau_{ji}[k] \leq \bar{\tau}_{ji} \leq \bar{\tau}$ for all $k \geq 0$ for some finite $\bar{\tau} = \max\{\bar{\tau}_{ji}\}$, $\bar{\tau} \in \mathbb{Z}_+$. We make the reasonable assumption that $\tau_{jj}[k] = 0$, $\forall v_j \in \mathcal{V}$, at all time instances k (i.e., the own value of a node is always available without delay). Each node updates its information state according to the following relation:

$$x_j[k+1] = p_{jj}x_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \sum_{r=0}^{\bar{\tau}} p_{ji}x_i[k-r]I_{k-r,ji}[r], \quad (5)$$

for $k \geq 0$, where $x_j[0] \in \mathbb{R}$ is the initial state of node v_j; p_{ji} $\forall \varepsilon_{ji} \in \mathcal{E}$ form $P = [p_{ji}]$ that adheres to the graph structure, and is primitive column stochastic; and

$$I_{k,ji}(\tau) = \begin{cases} 1, & \text{if } \tau_{ji}[k] = \tau, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

In the absence of delay, we have $\tau_{ji}[k] = 0$ and one should point out that the update relation (5) reduces to (1) with constant weights. Equation (5) is just a mathematical way of saying that each node v_j considers at time k all the packets it receives at time k. These packets were obviously sent at earlier time steps (i.e., $k, k-1, k-2, \ldots, 0$) and the assumption of bounded delay means that we only need to consider in the inner summation $k, k-1, k-2, \ldots, k-\bar{\tau}$; thus, we make the following assumptions:

(A1) The graph is strongly connected, and the (nonnegative) weights p_{ji} are nonzero (strictly positive) for $j = i$ and $(v_j, v_i) \in \mathcal{E}$, and satisfy $\sum_{l=1}^{n} p_{lj} = 1$ (so that they form a column primitive stochastic matrix P).

(A2) There exists a finite $\bar{\tau}$ that uniformly bounds the delay terms; i.e. $\tau_{ji}[k] \leq \bar{\tau} < \infty$ for all links $(v_j, v_i) \in \mathcal{E}$ at time instant k. In addition, $\tau_{jj}[k] = 0$ for all $v_j \in \mathcal{V}$ and all k.

Lemma 2: [12, *Lemma 2*] Consider a strongly connected digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Let $y_j[k]$ and $z_j[k]$ (for all $v_j \in \mathcal{V}$ and $k = 0, 1, 2, \ldots$) be the result of the iterations

$$y_j[k+1] = p_{jj} y_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \sum_{r=0}^{\bar{\tau}} y_{ji}[k-r] I_{k-r,ji}[r] \ ,$$

$$z_j[k+1] = p_{jj} z_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \sum_{r=0}^{\bar{\tau}} z_{ji}[k-r] I_{k-r,ji}[r] \ ,$$

under Assumptions (A1) and (A2). The initial conditions are $y[0] = (y_0(1) \;\; y_0(2) \; \ldots \; y_0(|\mathcal{V}|))^T \equiv y_0$ and $z[0] = \mathbb{1}$, and $I_{k,ji}$ is an indicator function that captures the bounded delay $\tau_{ji}[k]$ on link (v_j, v_i) at iteration k (as defined in (6), $\tau_{ji}[k] \leq \bar{\tau}$). Then, the solution to the average consensus problem can be asymptotically obtained as

$$\lim_{k \to \infty} \mu_j[k] = \frac{\sum_{v_\ell \in \mathcal{V}} y_0(\ell)}{|\mathcal{V}|} \ , \ \forall v_j \in \mathcal{V} \ ,$$

where $\mu_j[k] = y_j[k]/z_j[k]$.

At this stage, this is the only reported distributed coordination algorithm that is able to reach average consensus in directed graphs with asynchronous operation of the nodes. These properties make the robustified ratio consensus suitable for the application we discuss in the sequel.

IV. Workload Balancing in a Heterogeneous MapReduce Network

In this section, we apply the robustified ratio consensus described previously for balancing the workload in massively large-scale (and possibly asynchronous) MapReduce clusters. The suggested algorithm can be applied during either the map or the reduce phase. For simplicity, the current discussion describes only the load distribution on the network of mappers.

Consider a new MapReduce job d that needs to process a set of input data. The total demand of resources required by d in the map phase is given by ρ_d. We assume that ρ_d can be approximately estimated via offline application profiling, e.g., using historical data over similar jobs executed in the same network [7]. We target the problem of dividing the set of input data into *variable-sized* chunks and assign each chunk with a mapper process so that all mappers commence and complete processing synchronously with small differences. In this way, we address the problem of stragglers due to data processing load imbalances in heterogeneous networks. To this end, we exploit each mapper (node) according to its capacity, in order to finish its allocated workload at approximately the same time as the other mappers (we say approximately because in this work we treat load as a real number, essentially ignoring any quantization constraints).

We assume that the input data set is stored on a global file system accessible by all mappers, e.g., GFS [13] or HDFS [14], and suppose that each mapper runs on a separate node. A node can be either a physical or a virtual machine (VM). Let $\pi_j^{\max}$, $\forall j \in \{1, 2, \ldots, n\}$, $n = |\mathcal{V}|$, be the maximum processing capacity that node v_j can provide towards the map phase. For a single resource type, e.g., CPU resource, π_j^{max} corresponds to the maximum available CPU capacity of the node. In the case of a physical machine, the maximum available CPU capacity corresponds to the total capacity of the machine's CPU, if no other job is being processed at the time. In the case of a VM, π_j^{max} equals to the portion of the machine's CPU that this VM is capped to use; we assume that the virtualization layer enforces strict performance isolation among co-located VMs. We also assume that the node measures π_j^{max} using offline analysis, e.g., [7], based on its current utilization. Also, a network topology might be weighted due to different bandwidth utilizations and delays because of traffic from other jobs in the data center. As a result, some node-to-node paths may be in reality very slow during the execution of information exchange (due to large delays), for example [15], thus making the connection topology a directed graph.

The distributed algorithm converges to a balanced load distribution through information exchange between nodes. At time t_k of information exchange between nodes, we use $\pi_j[k] \in \mathbb{R}$ to denote the amount of resources that node v_j intends to contribute towards the map phase and let $\pi_j^{\max}$ $\forall j \in \{1, 2, \ldots, n\}$, be the capacity mapper v_j can provide. As soon as consensus is reached, say after m steps, node v_j chooses its state $\pi_j[m]$, $0 \leq \pi_j[m] \leq \pi_j^{\max}$, based on the available local information at v_j, such that the total resource demanded ρ_d is equal to the total amount of resources offered by the nodes, provided the total capacity of resources available is greater than the demanded one, i.e.,

$$\rho_d = \sum_{v_\ell \in \mathcal{V}} \pi_\ell[m], \tag{7a}$$

such that

$$0 \leq \pi_j[m] \leq \pi_j^{\max}, \quad \forall v_j \in \mathcal{V}, \tag{7b}$$

$$0 \leq \rho_d \leq \sum_{v_\ell \in \mathcal{V}} \pi_\ell^{\max} \triangleq \chi^{\max}. \tag{7c}$$

It is noted in [11] that a simple, feasible solution $\pi^\dagger$ to the problem is given by

$$\pi_j^\dagger = \frac{\rho_d}{\chi^{\max}} \pi_j^{\max}, \ \forall v_j \in \mathcal{V} \ , \tag{8}$$

which is always less than the capacity $\pi_j^{\max}$ of CPU resource at v_j. However, for each node v_j to be assigned $\pi_j^\dagger$, knowledge of $\chi^{\max}$ and ρ_d are required at each node v_j; these values have to be computed in a distributed fashion.

Each node v_j updates its demanded amount $\pi_j[k]$ by combining it with the available (possibly delayed) demanded amount of its neighbors ($\pi_i[s]\ s \in \mathbb{Z}, s \leq k,\ v_i \in \mathcal{N}_j^-$), according to the following relation:

$$\pi_j[k+1] = \frac{1}{1+\mathcal{D}_j^+}\pi_j[k] + \sum_{v_i \in \mathcal{N}_j^-} \sum_{r=0}^{\bar{\tau}} \frac{1}{1+\mathcal{D}_i^+}\pi_i[k-r] I_{k-r,ji}[r], \quad (9)$$

for $k \geq 0$, where $\pi_j[0] \in \mathbb{R}$ is the initial state of node v_j.

If we run the two iterations, as in Lemma 2 (with $y_j[0] = \pi_j[0]$ and $y_j[0] = pi_j^{\max}$), we have

$$\lim_{k\to\infty} \mu_j[k] = \frac{\sum_{v_\ell \in \mathcal{V}} \pi_\ell[0]}{\sum_{v_\ell \in \mathcal{V}} \pi_\ell^{\max}[0]} = \frac{\rho_d}{\chi^{\max}}, \quad \forall v_j \in \mathcal{V}.$$

As a result, node v_j can obtain the amount of its load as

$$\lim_{k\to\infty} \mu_j[k]\pi_j^{\max} = \frac{\rho_d}{\chi^{\max}}\pi_j^{\max}. \quad (10)$$

The workload offered is found in a distributed fashion and it is equal to the feasible solution noted in (8).

V. Numerical Examples

In this section, we demonstrate the properties and performance of our algorithm through simulations for various network types. Comparisons with other approaches are also provided.

First, we consider a small directed network consisting of six mappers (see Figure 2) in order to illustrate how the algorithm operates. Each node v_j chooses its self-weight and

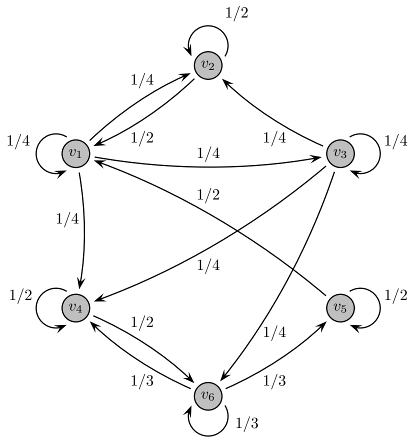

Fig. 2. A directed network consisting of six nodes.

the weights on its outgoing links to be $(1+\mathcal{D}_j^+)^{-1}$ (such that the sum of all weights assigned by each node v_j to its outgoing links is equal to 1). In this example, the demanded workload amount injected into the network of mappers is given by $\pi[0] = (15\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0)^T$, i.e., all the workload is passed to the network through one of the mappers (this could be the case in reality). The capacity of the mappers in this network is given by $\pi^{\max}[0] = (3\ \ 3\ \ 3\ \ 3\ \ 4\ \ 4)^T$, where $\pi[0], \pi^{\max}[0] \in \mathbb{R}_+^n$. We induce *asymmetric time-varying delays*[1] into the communication between the mappers, with maximum delay $\bar{\tau} = 5$ (in the simulations, a link is delayed with a probability 0.5, and the magnitude of the delay is an integer chosen with equal probability from $\{1, 2, 3, 4, 5\}$.

When each node updates its information state $y_j[k]$ using equation (4a), the information state for the whole network is given by $y[k+1] = Py[k]$, where

$$P = \begin{pmatrix} 1/4 & 1/2 & 0 & 0 & 1/2 & 0 \\ 1/4 & 1/2 & 1/4 & 0 & 0 & 0 \\ 1/4 & 0 & 1/4 & 0 & 0 & 0 \\ 1/4 & 0 & 1/4 & 1/2 & 0 & 1/3 \\ 0 & 0 & 0 & 0 & 1/2 & 1/3 \\ 0 & 0 & 1/4 & 1/2 & 0 & 1/3 \end{pmatrix}.$$

When delays are present the two iterations proceed according to equation (9), and the network of resources reaches consensus $\rho_d/\chi^{\max} = 15/20$; this is illustrated in Figure 3. The amount of workload for the MapReduce job offered

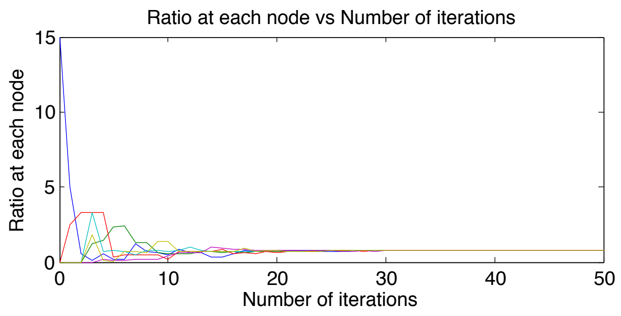

Fig. 3. The ratio at each node converges to $\rho_d/\chi^{\max} = 15/20$.

in the network by each node is proportional to its resource capacities. In this case, we have resources with two different capacities: four of them have capacity equal to 3, and two of them have capacity equal to 4. Since the ratio $\rho_d/\chi^{\max} = 0.75$, then two of the values converge to 9/4 and three of them to 3, as shown in Figure 4. This is achieved by having

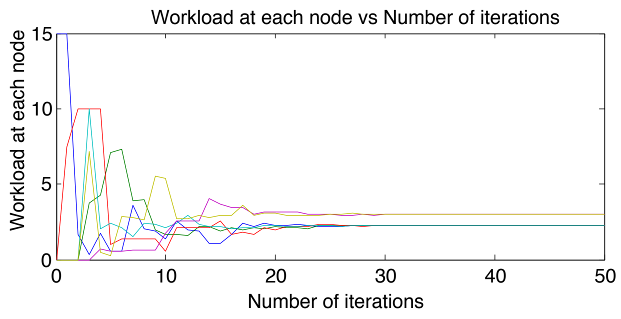

Fig. 4. The amount of workload offered in the network by each node according to (10) in the presence of asymmetric time-varying delays.

the capacity values as the initial conditions for the second iteration, instead of having all ones. Note that delays affect

[1]By asymmetric time-varying delays we mean that at each of the outgoing links of node v_j and each time instant the delay can be different.

the convergence rate; in general, the larger the delays, the lower the convergence rate of the algorithm.

Remark 1: Note that the superiority of the robustified ratio consensus [12], proposed for this application, over other approaches in the literature (e.g., [16]) is justified in *directed* networks and in *asynchronous* operation. In these cases, no comparisons are possible since existing algorithms cannot be used.

In the following example, we consider a set of 500 mappers each with initial workload $\pi_j[0] = 3$. However, half of them have capacity $\pi_j^{\max} = 3$ and the other half $\pi_i^{\max} = 5$, $i \neq j$. By ratio consensus, according to equation (9), with no delays, the network of resources reach consensus $\rho_d/\chi^{\max} = 0.75$. As a result, half of the values converge to $9/4$ and the other half to $15/4$, as shown in Figure 5.

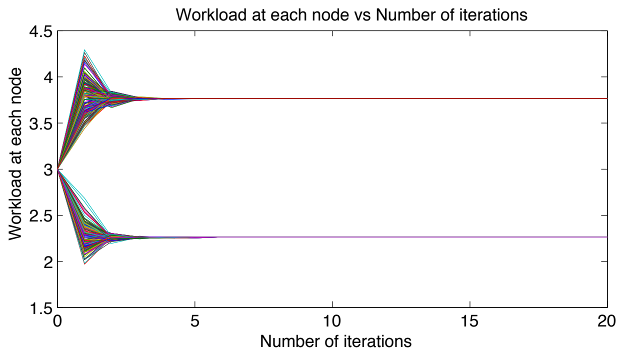

Fig. 5. The amount of workload offered in the network by each node according to (10) for a set of 500 mappers, with no delays.

It is important to note that the time required to converge to the proportional workload would not differ much (or at all) for more mappers in the network, since the information and adaptation is made locally. Traditional schedulers in Hadoop [17]—the popular MapReduce implementation—implement incremental assignments of fixed-size chunks to mappers, i.e., every time a mapper has finished processing a chunk, it notifies the scheduler and receives the next chunk. This scheme requires direct communication of hundreds of thousands of mappers directly with the scheduler multiple times, leading to increased numbers of messages exchanged. Our approach, requiring only local information sharing, shows fast convergence to the aggregate workload that each mapper eventually processes.

VI. Conclusions

Motivated by real-world conditions of node heterogeneity and existence of delays, we proposed a distributed algorithm for workload distribution in MapReduce networks of clusters. Our suggested algorithm allows mappers on heterogeneous nodes to operate asynchronously and converge asymptotically to proportional workload balancing between them, with limited local information exchanges. The algorithm was shown to converge to the exact proportional workload in directed networks in the presence of delays, something that cannot be achieved by classical consensus algorithms. The properties and performance of the suggested algorithm were illustrated for small and large-scale strongly connected topologies of mappers. The algorithm was compared with alternative algorithms for workload balancing.

References

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI).* ACM, 2004, pp. 137–150.

[2] "SIGMETRICS tutorial: MapReduce The Programming Model and Practice," online, 2009, http://research.google.com/archive/papers/mapreduce-sigmetrics09-tutorial.pdf.

[3] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at Facebook," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010, pp. 1013–1020.

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2008, pp. 29–42.

[5] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in MapReduce clusters using Mantri," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010, pp. 1–16.

[6] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Tarazu: optimizing MapReduce on heterogeneous clusters," in *Proceedings of 17th Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 61–74.

[7] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: automatic resource inference and allocation for MapReduce environments," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, 2011, pp. 235–244.

[8] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in *IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW)*, 2010, pp. 1–9.

[9] E. Kotsovinos, "Virtualization: blessing or curse?" *Queue*, vol. 8, pp. 40–46, 2010.

[10] A. Gonzalez-Ruiz and Y. Mostofi, "Distributed load balancing over directed network topologies," in *Proceedings of the American Control Conference (ACC)*, 2009, pp. 1814–1820.

[11] A. D. Domínguez-García and C. N. Hadjicostis, "Coordination and control of distributed energy resources for provision of ancillary services," in *First IEEE International Conference on Smart Grid Communications*, 2010, pp. 537–542.

[12] C. N. Hadjicostis and T. Charalambous, "Asynchronous coordination of distributed energy resources for the provisioning of ancillary services," in *Proceedings of the 49^{th} Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, September 2011, pp. 1500–1507.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003, pp. 29–43.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–10, 2010.

[15] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th Annual Conference on Internet Measurement (IMC)*, 2010, pp. 267–280.

[16] L. Xiao, S. Boyd, and S. Lall, "Distributed average consensus with time-varying metropolis weights," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, 2005, pp. 63–70.

[17] (2012) Hadoop. [Online]. Available: http://apache.hadoop.org