Load balancing on networks with gossip-based distributed algorithms

Mauro Franceschelli, Alessandro Giua, Carla Seatzu

Abstract

We study the distributed and decentralized load balancing problem on arbitrary connected graphs, representing an homogeneous network. The network contains several tasks, represented by possibly different integer numbers, to be processed at nodes. We propose a randomized algorithm based on gossip that achieves consensus on the load distribution within fixed bounds of the optimal one; we also show by simulations that in most cases the achieved consensus is optimal. We finally present a computationally convenient heuristic and show that it ensures the same bounds: simulation results, however, show that the heuristic performs worse.

Published as:
M. Franceschelli, A. Giua, C. Seatzu, "Load balancing on networks with gossip-based distributed algorithms," *46th IEEE Conf. on Decision and Control* (New Orleans, LA, USA), December 2007.

I. Introduction

In this paper we consider a discrete version of consensus based on gossip. The classical formulation of the consensus problem was *continuous* and different solutions have been presented in [13], [11], [2], [1], where it is assumed that the values at the nodes are real numbers. Recently, this problem has been extended by Kashyap, Başar and Srikant [10] to the *quantized* consensus, where the values at the nodes are integer numbers, pointing out as possible applications flocking [12] and load balancing [4], [15], [14].

M. Franceschelli, A. Giua and C. Seatzu are with the Dept. of Electrical and Electronic Engineering, University of Cagliari, Piazza D'Armi, 09123 Cagliari, Italy. Email: franceschelli@diee.unica.it, {giua,seatzu}@diee.unica.it.

An alternative way of looking at the quantized case presented in [10] is that of assuming that K tasks of weights $c_j = 1$ ($j = 1, \ldots, K$) are traveling on the network, and it is required to reach a balanced load distribution. The problem of load balancing with unity size tokens has been studied in many forms [5], [3], [9], [8] and a fair amount of attention has been given to distributed solutions in which the desired global behavior emerges from the interaction rules between the agents [6], [7].

Here we generalize this setting assuming that $c_j \in \mathbb{N}$, i.e., different integer weights may be associated to each task. In the following, we will refer to such a problem as *generalized quantized load balancing* and we denote the proposed gossip-based algorithms as *generalized quantized gossip algorithms* (GQG).

As pointed out by Gosh in [6] parallel systems benefit from highly distributed task scheduling for two main reasons: the cost of communication with processors in the neighborhood is small and it allows to gain the most from massive parallel architectures. Nevertheless approximate distributed load balancing algorithms may block when non-unitary size tasks are considered [15].

In this paper, as in [10], it is illustrated how increasing the load transfers in distributed load balancing algorithms by simply requiring some load swaps (even at a perfect local balance) may globally improve the load balancing on the network, overcoming eventual blocking configurations.

In this paper we did not study the case when tasks are continuously introduced in the system and disappear after being accomplished: this will be object of future research.

A. Problem statement

Let us consider a network of n nodes whose connections can be described by an undirected connected graph $\mathcal{G} = (V, E)$, where V is the set of nodes, and E is the set of arcs.

Assume that K indivisible tasks should be assigned to the nodes, and an integer cost (or weight) is associated to each task.

Our goal is that of determining an optimal load balancing, starting from any initial condition,

such that the load of each node is as close as possible, in the least-square sense[1], to the average load, namely to

$$c_{ave} = \frac{1}{n} \sum_{j=1}^{K} c_j \tag{1}$$

where $c_j \in \mathbb{N}$ denotes the cost of the j-th task.

This problem can be easily formulated as a quadratic programming problem (QPP) with binary variables. To do this we define a cost vector $c \in \mathbb{N}^K$ whose j-th component is equal to c_j, and n binary vectors $\vec{y}_i \in \{0,1\}^K$ such that

$$y_{i,j} = \begin{cases} 1 & \text{if the } j\text{-th task is assigned to node } i \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The QPP takes the form:

$$\begin{cases} \min V = \sum_{i=1}^{n} \left(c^T \vec{y}_i - c_{ave} \right)^2 \\ s.t. \ \sum_{i=1}^{n} \vec{y}_i = \vec{1} \\ \qquad y_{i,j} \in \{0,1\} \qquad i = 1, \ldots, n; \ \ j = 1, \ldots, K \end{cases} \tag{3}$$

where $\vec{1}$ is a K dimensional vector of ones.

In the following we denote

$$Y = [\vec{y}_1 \ \vec{y}_2 \ \ldots \ \vec{y}_n] \tag{4}$$

any feasible solution of (3) and denote Y^* (resp., V^*) the optimal solution (resp., the optimal value of the performance index) of Problem (3). Finally, we denote

$$c_{\min} = \min_{j=1,\ldots,K} c_j, \quad c_{\max} = \max_{j=1,\ldots,K} c_j \tag{5}$$

the minimum and maximum cost of tasks in the network.

Problem (3) can be solved using standard tools for quadratic programming. However, this may not always be convenient or even possible. In fact, in the case of large number of nodes and tasks the computational complexity may be prohibitive. Moreover, the implementation of the solution requires a centralized control agent, that might be unfeasible.

[1]Note that the optimal load balancing obtained by minimizing the maximal load is equivalent to the one in the least square sense. We chose the second criterion to make sure that at each iteration the performance index is improved whenever two nodes reduce their load difference.

B. Paper content

In this paper we focus on *distributed solutions* looking at a load balancing problem as a special class of *consensus*, i.e., distributed averaging. The original formulation of consensus deals with real variables. These results while not directly applicable in the discretized case, offer useful tools to understand the problem.

We propose solutions that are based on *gossip*. Time is assumed to be discrete and asynchronous and at each step, one edge is selected randomly: the load of the nodes incident on the selected edges is updated using an appropriate rule. In any case it should be guaranteed that at every time instant, the probability of an edge to be selected is strictly greater than zero.

These are the contributions of this paper. Firstly, we propose an algorithm that at each step requires solving an Integer Programming problem (IPP) to balance the load of two communicating nodes. We also show that this algorithm achieves consensus within fixed bounds of the optimal one. Secondly, we modify the algorithm introducing a computationally convenient heuristic that does not require the solution of IPP's and show that it ensures the same bounds. Thirdly, we briefly discuss the convergence time of the two procedures following the results of [10]. Finally, we report the results of several simulations that show the performance of the proposed algorithms. The IPP algorithm converges to the optimum most of the times while the heuristic one usually performs worse.

II. Generalized quantized gossip with IPP

Let us introduce some preliminary definitions. In the following, given a generic node i, we denote $\mathcal{K}_i(t)$ the set of indices of tasks assigned to i at time t.

We first define an operation between two communicating nodes that, while not changing the value of the objective function, modifies the load configuration.

***Definition** 1 (Swap):* Let us consider two nodes i and r incident on the same edge and let $\mathcal{I}_i \subseteq \mathcal{K}_i(t)$ and $\mathcal{I}_r \subseteq \mathcal{K}_r(t)$ be two subsets of their tasks.

We call *swap* the operation that moves the tasks in $\mathcal{I}_i$ to r, and the tasks in $\mathcal{I}_r$ to i at time $t+1$, reaching the distribution

$$\mathcal{K}_i(t+1) = \mathcal{I}_r \cup (\mathcal{K}_i(t) \setminus \mathcal{I}_i),$$
$$\mathcal{K}_r(t+1) = \mathcal{I}_i \cup (\mathcal{K}_r(t) \setminus \mathcal{I}_r).$$

provided that the absolute value of the load difference between the two nodes does not change, i.e.,

$$\left| \sum_{j \in \mathcal{K}_i(t+1)} c_j - \sum_{j \in \mathcal{K}_r(t+1)} c_j \right| = \left| \sum_{j \in \mathcal{K}_i(t)} c_j - \sum_{j \in \mathcal{K}_r(t)} c_j \right|.$$

In particular, we say that a *total swap* occurs if $\mathcal{I}_i = \mathcal{K}_i(t)$ and $\mathcal{I}_r = \mathcal{K}_r(t)$, while we say that a *partial swap* occurs if either $\mathcal{I}_i \subsetneq \mathcal{K}_i(t)$ or $\mathcal{I}_r \subsetneq \mathcal{K}_r(t)$. ■

Now, let us present the *generalized quantized gossip* algorithm based on the solution of integer programming problems (IPP), that we denote in the following with the acronym GQG-IPP.

***Algorithm** 1 (GQG-IPP Algorithm):*

1) Let $t := 0$.
2) Select an edge $\{i, r\}$.
3) Let $\mathcal{K} := \mathcal{K}_i(t) \cup \mathcal{K}_r(t)$.
4) Let $\varrho := |\sum_{j \in \mathcal{K}_i(t)} c_j - \sum_{j \in \mathcal{K}_r(t)} c_j|$.
5) If $\varrho > 1$ *(the load balancing among the two nodes may potentially be improved)*, then
   a) let $c_m := \frac{1}{2} \sum_{j \in \mathcal{K}} c_j$;
   b) solve the integer programming problem (IPP):

$$\begin{cases} \min \sum_{j \in \mathcal{K}} c_j x_j \\ s.t. \ \sum_{j \in \mathcal{K}} c_j x_j \geq c_m \\ \qquad x_j \in \{0, 1\}, \quad j \in \mathcal{K} \end{cases} \tag{6}$$

   c) if $\{j \ | \ x_j = 1\} \neq \mathcal{K}_i(t)$ *(we find a solution that is different from the previous one)*, then let

$$\mathcal{K}_i(t+1) := \{j \ | \ x_j = 1\},$$
$$\mathcal{K}_r(t+1) := \{j \ | \ x_j = 0\}.$$

   else execute a *swap* among i and r.
6) If $\varrho \leq 1$ *(the load balancing cannot be improved)*, then execute a *swap* among i and r.
7) Let $t := t + 1$ and goto step 2. ■

The main idea behind Algorithm 1 is the following. We select randomly one edge, and evaluate the total loads of the nodes i and r incident on it. At step 5 we try to balance the loads between the two nodes (task j is assigned to to node i if $x_j = 1$, else it is assigned to node r). If this fails a (partial or total) swap is executed. Note that when a swap is executed, it may be possible

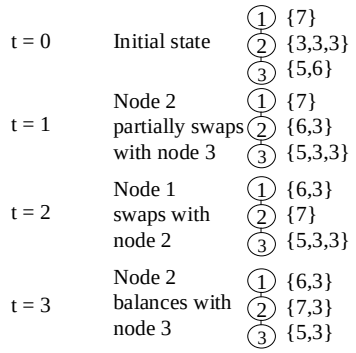

Fig. 1. Evolution of the network in Example 1.

to execute either a total or a partial one. Note that the type of swap considered in [10] is what we call total swap: this is sufficient to solve the problem when all tasks have the same weight. However, in the problem we consider in this paper a partial swap may be necessary to overcome a blocking condition as next example shows.

***Example** 1:* Let us consider the network with three nodes shown in Figure 1. Here there are only two arcs, one between nodes 1 and 2, and one between nodes 2 and 3. In the initial state

$$\mathcal{K}_1(0) = \{1\}, \quad \mathcal{K}_2(0) = \{2, 3, 4\}, \quad \mathcal{K}_3(0) = \{5, 6\},$$

$$c_1 = 7, \quad c_2 = c_3 = c_4 = 3, \quad c_5 = 5 \quad c_6 = 6, \quad c_m = 9.$$

At time $t = 0$ it holds $V(Y(0)) = 8$ and no optimization between any couple of nodes is feasible. Assume that at time $t = 1$ a partial swap between node 2 and 3, such that $\mathcal{K}_2(1) = \{2, 6\}$ and $\mathcal{K}_3(1) = \{3, 4, 5\}$ occurs.

The objective function has not improved but now a load exchange may happen. After a total swap between nodes 1 and 2 occurs at time $t = 2$, node 2 and 3 may balance their load reaching at time 3 the optimal configuration

$$\mathcal{K}_1(1) = \{2, 6\}, \quad \mathcal{K}_2(1) = \{1, 3\}, \quad \mathcal{K}_3(1) = \{4, 5\},$$

with $V(Y(3)) = 2$. ■

We remark that Algorithm 1 is not concerned with the minimization of the exchanged load, that may be an important requirement in real applications. Next example shows a case in which among the possible solutions we may choose one that minimizes the exchanged load.

***Example** 2:* Let us consider the case of two nodes, 1 and 2, such that $\mathcal{K}_1(0) = \{1\}, \mathcal{K}_2(0) = \{2, 3\}$, and $c_1 = 7, c_2 = 8, c_3 = 1$. An optimal solution is reached at $t = 1$ solving IPP (6). In particular, we may either obtain $\mathcal{K}_1(1) = \{1, 3\}$ and $\mathcal{K}_2(1) = \{2\}$, or $\mathcal{K}_1(1) = \{2\}$ and $\mathcal{K}_2(1) = \{1, 3\}$. Note however, that while the first solution simply implies the movement of the third task from node 2 to node 1, the second solution implies the movement of both tasks 1 and 2 from one node to the other one. ■

Algorithm 1 performed very well in a series of random tests we have executed, in the sense that it always reached the optimal solution (see Section V). However, it does not guarantee the optimality of the solution for any initial configuration.

***Example** 3:* Let us consider a three node fully connected[2] network. Whose initial configuration of the network is

$$\mathcal{K}_1(0) = \{1\}, \quad \mathcal{K}_2(0) = \{2, 3, 4\}, \quad \mathcal{K}_3(0) = \{5, 6\}$$

where $c_1 = 7, c_2 = c_3 = c_4 = 3, c_5 = c_6 = 5$. An optimal configuration in terms of load balancing is

$$\mathcal{K}_1^* = \{1, 2\}, \quad \mathcal{K}_2^* = \{3, 5\}, \quad \mathcal{K}_3^* = \{4, 6\}.$$

However, it cannot be reached using Algorithm 1, because neither a swap (be it partial or total) nor a load balancing is possible between any two nodes. ■

The previous example highlights an important general property: an optimal load balancing with non-unitary tasks cannot always be achieved by greedy gossip algorithms, that balance the load between two nodes at each step, even on a fully connected network. In fact, to reach consensus an optimization involving more than two nodes at the same time may be necessary.

III. Generalized quantized gossip with heuristic

Algorithm 1 at each iteration it may require the solution of an IPP which computational complexity is NP hard[3]. In this section we provide an alternative solution to it that has a polynomial complexity.

[2]A network is *fully connected* if there is an arc from each node to any other one.

[3]This problem has the same structure, and thus the same computational complexity, of the knapsack problem. An exhaustive search of the optimal solution would require the investigation of $2^{K'}$ different configurations, where K' is the number of tasks in the two selected nodes.

The proposed heuristic may be summarized in the following algorithm, that we denote *generalized quantized gossip with heuristic* (GQG-H) algorithm.

***Algorithm** 2 (GQG-H):*

1) Let $t := 0$.
2) Select an edge $\{i, r\}$.
3) Let $\mathcal{K} := \mathcal{K}_i(t) \cup \mathcal{K}_r(t)$.
4) Let $\mathcal{K}'_i := \emptyset$ and $\mathcal{K}'_r := \emptyset$ *(we define new temporary sets, initialized to the empty set, including the indices of tasks in the selected nodes).*
5) While $\mathcal{K} \neq \emptyset$, do
   - let $\delta := \text{argmax}_{j \in \mathcal{K}} c_j$;
   - if $\sum_{j \in \mathcal{K}'_i} c_j \leq \sum_{j \in \mathcal{K}'_r} c_j$, then let
   $$\mathcal{K}'_i := \mathcal{K}'_i \cup \{\delta\}, \quad \mathcal{K}'_r := \mathcal{K}'_r;$$
   (if the load of the i-th node is greater or equal to that of the r-th node, then assign the current task to node i)
   - else let $\mathcal{K}'_i := \mathcal{K}'_i, \ \mathcal{K}'_r := \mathcal{K}'_r \cup \{\delta\}$.
   - $\mathcal{K} := \mathcal{K} \setminus \{\delta\}$
6) if $\left| \sum_{j \in \mathcal{K}'_i} c_j - \sum_{j \in \mathcal{K}'_r} c_j \right| \leq \left| \sum_{j \in \mathcal{K}_i(t)} c_j - \sum_{j \in \mathcal{K}_r(t)} c_j \right|$,
   and $\mathcal{K}'_i \neq \mathcal{K}_i(t)$ or $\mathcal{K}'_r \neq \mathcal{K}_r(t)$ *(we find a solution that is not worse and different from the previous one)* then let $\mathcal{K}_i(t+1) := \mathcal{K}'_i, \quad \mathcal{K}_r(t+1) := \mathcal{K}'_r$; else execute a *swap* among nodes i and r.
7) Let $t := t + 1$ and goto step 2.
8) Exit. ■

In practice, Algorithm 2 differs from Algorithm 1 for the fact that, at each step, the solution of an IPP with $|\mathcal{K}|$ binary variables is replaced by $|\mathcal{K}|$ intermediate steps simply consisting in algebraic manipulations.

Obviously, the effectiveness of such an heuristic depends in general on the size of the tasks. If there is a sufficiently large number of "small" tasks, then it can find the optimal solution. Otherwise, only a sub-optimal balancing may be reached.

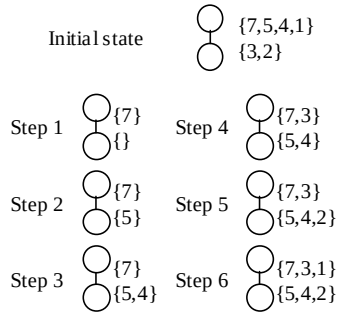

Fig. 2. Evolution of the network in Example 4.

***Example** 4:* To show how the heuristic works, let us consider the case shown in Figure 2. The initial state is

$$\mathcal{K}_1(0) = \{1, 2, 3, 6\}, \quad \mathcal{K}_2(0) = \{4, 5\},$$

$$c_1 = 7, \quad c_2 = 5, \quad c_3 = 4, \quad c_4 = 3, \quad c_5 = 2, \quad c_6 = 1.$$

We define a new temporary state in which

$$\mathcal{K}'_1(0) = \emptyset, \quad \mathcal{K}'_2(0) = \emptyset.$$

At this point we remove the biggest task from $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2$ and put it in $\mathcal{K}'_1$. At step 2 we compare the load between the nodes, since node 1 has a heavier load we remove the biggest task from $\mathcal{K}$ and put it in node 2. We repeat this process until $\mathcal{K} = \emptyset$. Then we check that $|\sum_{j\in\mathcal{K}'_1} c_j - \sum_{j\in\mathcal{K}'_2} c_j| \leq |\sum_{j\in\mathcal{K}_1(t)} c_j - \sum_{j\in\mathcal{K}_2(t)} c_j|$, in our case $|7+3+1-5-4-2| \leq |7+5+4+1-2-3|$, $0 \leq 12$. Finally we let $\mathcal{K}_i(t+1) = \mathcal{K}'_i \quad \mathcal{K}_r(t+1) = \mathcal{K}'_r$ terminating the iteration. ■

IV. Convergence properties

We now provide a characterization of the set of *home states* for Algorithm 1 and Algorithm 2, namely the set of load configurations that can be reached (regardless of the initial configuration) after a sufficiently large number of steps. We also briefly discuss the convergence time.

We first observe that for any time t, each solution of both algorithms can univocally be rewritten in terms of vectors $\vec{y}_i(t)$'s defined as in (2), and matrix $Y(t)$ defined as in (4).

***Theorem** 1:* Let us consider the optimal load balancing problem (3), and let

$$\mathcal{Y} = \{Y = [\vec{y}_1\ \vec{y}_2\ \cdots\ \vec{y}_n]\ \ |\ \ |c^T\vec{y}_i - c^T\vec{y}_r| \leq c_{\max}, \quad \forall\ i, r \in \{1, \ldots, n\}\}. \tag{7}$$

Let $Y(t)$ be the matrix that summarizes the load balancing resulting from Algorithm 1 or Algorithm 2 at the generic time t.

It holds

$$\lim_{t\to\infty} \Pi(Y(t) \in \mathcal{Y}) = 1$$

where $\Pi(Y(t) \in \mathcal{Y})$ denotes the probability that $Y(t) \in \mathcal{Y}$.

Proof: Let us assume as a Lyapunov function the objective function of (3), namely $V(Y(t)) = \sum_{i=1}^{n}(c^T\vec{y}_i(t) - c_{ave})^2$.

We can easily prove that $V(t)$ is a non increasing function of t. In fact, for both algorithms, at any time t it holds $V(t+1) \leq V(t)$. The case $V(t+1) = V(t)$ holds when a swap is executed.

The case of $V(t+1) < V(t)$ holds when a new load balancing occurs. Assume (without any loss of generality) that $c^T\vec{y}_i(t) > c^T\vec{y}_r(t)$, and that a load q with $0 < q < c^T\vec{y}_i(t) - c^T\vec{y}_r(t)$ is moved from i to r at the generic time t. It is easy to verify, by simple computations, that $(c^T\vec{y}_i(t+1) - c_{ave})^2 + (c^T\vec{y}_r(t+1) - c_{ave})^2 < (c^T\vec{y}_i(t) - c_{ave})^2 + (c^T\vec{y}_r(t) - c_{ave})^2$ which implies $V(t+1) \leq V(t)$.

We also observe that if two nodes (e.g., i and r) communicate at time t, the resulting difference among their loads at time $t+1$ is surely less or equal to the largest cost of tasks in the nodes at time t, i.e.,

$$|c^T\vec{y}_i(t+1) - c^T\vec{y}_r(t+1)| \leq \max_{j\in\mathcal{K}_i(t)\cup\mathcal{K}_r(t)} c_j \leq c_{\max}. \tag{8}$$

This result obviously holds for Algorithm 1, because it always determines the optimal balancing between two nodes and, should (8) be violated, a better balancing can be obtained moving any task from the most charged node to the other one. It is also possible to show that the result holds for Algorithm 2 as well. In this case the proof is inductive on the number of loads that have been allocated at step 5. In fact the result holds for the initial allocation (when both nodes are empty). Furthermore when each additional task is allocated the difference between the load of two nodes cannot be greater than the weight of the task.

Finally, let us observe that the possibility of having swaps guarantees that also the load of "distant" nodes (i.e., nodes not sharing an edge), may effectively be balanced. In fact, thanks

to the swap, there exists a probability $\Pi > 0$ that after a finite number of iterations the load of these nodes may be balanced, thus leading the network to a configuration such that $Y(t) \in \mathcal{Y}$. As a consequence, if we allow an infinite number of swaps ($t \to \infty$), the probability that this actually occurs is unitary, thus proving the statement[4]. □

***Remark** 1:* Note that Theorem 1 does not require partial swaps to be performed, and the result holds even if only total swaps are considered. However, as we have shown in Example 1, partial swaps may improve the solution. ■

A characterization of the maximum distance of the performance index of a solution obtained using Algorithm 1 or Algorithm 2 from the optimal one is given by the following proposition.

***Proposition** 1:* Let us consider the optimal load balancing problem (3), and let the set $\mathcal{Y}$ be defined as in equation (7). Let $V(Y) = \sum_{i=1}^{n} (c^T \vec{y}_i - c_{ave})^2$ be the objective function of (3), where $Y = Y(t)$ results from the application of Algorithm 1 or Algorithm 2 for a sufficiently large value of t.

The following inequalities hold for any $Y \in \mathcal{Y}$:

$$0 \leq V^* \leq V(Y) \leq \alpha \tag{9}$$

where

$$\alpha = \begin{cases} \dfrac{n c_{\max}^2}{4} & \text{if } n \text{ is even,} \\ \left\lfloor \dfrac{n}{2} \right\rfloor \left\lceil \dfrac{n}{2} \right\rceil \dfrac{c_{\max}^2}{n} & \text{if } n \text{ is odd.} \end{cases} \tag{10}$$

Proof: The first two inequalities are trivial. To prove the last inequality we look at the worst case, i.e., the load balancing in $\mathcal{Y}$ that has the highest value of the performance index.

If n is even, the worst case corresponds to a balancing where half of the nodes have a load k and the remaining half have a load $k + c_{\max}$. In this case $c_{ave} = k + 0.5 c_{\max}$, and the first value of bound can be computed.

If n is odd, the worst case corresponds to a balancing where $\lfloor n/2 \rfloor$ of the nodes have a load k and the remaining $\lceil n/2 \rceil$ have a load $k + c_{\max}$. Now $c_{ave} = k + \lceil n/2 \rceil c_{\max}/n$, which gives the other value of the bound. □

The above results enable us to characterize some cases in which Algorithm 1 and Algorithm 2 provide the optimal load balancing.

[4]A proof that this actually occurs can be found in [10], see section IV, Theorem 2.

***Proposition** 2:* Let us consider the optimization problem (3), and let $c_{\min}$ and $c_{\max}$ be defined as in (5).

If $c_{\min} = c_{\max} = c$, then all load distributions that belong to a set of final distributions (7) are optimal, hence Algorithm 1 and Algorithm 2 provide an optimal solution to problem (3).

Proof: If $c_{\min} = c_{\max}$ the set of final distributions is

$$\mathcal{Y} = \{[\vec{y}_1 \ \cdots \ \vec{y}_n] \ \mid (\forall \ i) \ c^T \vec{y}_i \in \{\lfloor \tfrac{K \cdot c}{n} \rfloor, \lfloor \tfrac{K \cdot c}{n} \rfloor + c\}\}. \tag{11}$$

We can normalize the weight c so that it is unitary. With this formulation the problem corresponds to that of quantized consensus, and the set $\mathcal{Y}$ coincides with the set of the *quantized-consensus distributions* defined in [10] and shown to be optimal. □

Finally, we discuss some results in [10] that apply to both Algorithm 1 and Algorithm 2. The convergence time is a random variable defined for $Y(0) = Y$ as:

$$T_c(Y, c) = \inf\{t \mid \ (\forall t' \geq t) \ V(Y(t')) = V(Y(t))\}.$$

The following results have been proven in [10].

1) For any initialization on the set

$$\{Y = [\vec{y}_1 \ \cdots \ \vec{y}_n] \mid \ (\forall i) \ m \leq c^T y_i \leq M\},$$

the performance index is bounded by

$$V(Y(0)) \leq \frac{(M-m)^2 n}{4}.$$

2) At each iteration in which a load balancing occurs, the improvement of the performance index is lower bounded by $V(Y(t+1)) \leq V(Y(t)) - 2$ since the minimum load exchange allowed decreases the load difference between two nodes of at least 1. It follows that at most $\frac{(M-m)^2 n}{8}$ load balancing are needed to reach an optimal load distribution while, according to Proposition 1, at most

$$\rho = \frac{((M-m)^2 - c_{\max}^2)n}{4}$$

load balancing are needed to reach the set of final distributions (7).

3) We define $T(\mathcal{G})$ the maximum, among every initial load configuration, of the average time for the first improvement of the performance index. It has been proven in [10] that for a linear[5] network $T(\mathcal{G}) = \frac{n(n^2-1)(n-1)}{4}$.

[5]In a *linear network* each node $i \in \{2, \cdots, n-1\}$ can only communicate with nodes $i-1$ and $i+1$, while the node 1 (resp., node n) can only communicate with node 2 (resp., $n-1$).

The same results hold in our case if only total swaps are considered. In such a case the expected value $E[T_c(Y,c)]$ of the time it takes a linear network to reach the set of final distributions (7) can be bounded by

$$\max_{Y,c} E[T_c(Y,c)] \leq \rho \cdot T(\mathcal{G})$$
$$= \frac{((M-m)^2 - c_{\max}^2)n}{4} \cdot \frac{n(n^2-1)(n-1)}{4}.$$

When partial swaps are also allowed, the value of $T(\mathcal{G})$ changes; we do not provide a bound for such a case.

V. Simulation results

To complete the theoretical analysis we performed some simulations. Two kinds of networks are investigated, fully connected (FC) nets and linear nets. The worst case scenario of a linear net is taken to emphasize the slow down effect of the "swap" dynamic. The best case of a fully connected network is taken to show the behavior of the algorithm when swaps are not necessary.

Three sizes have been considered: networks with 5 , 10 and 20 nodes with respectively a load of 50 , 100 and 200 tasks to keep their average number per node constant at the optimum load distribution. Also the initial configurations considered are a worst case in which all the load is positioned in the first node. The weight related to each task is chosen between a minimum of 1 and a maximum of 10 for the first set of simulations, a minimum of 1 and a maximum of 100 for the second set. In both cases the weights are chosen randomly with uniform probability distribution over the respective bounds.

In Table I the convergence times averaged over 100 realizations are shown. The convergence time T_m represents the number of iterations needed by the algorithm to reach the best load balancing. Furthermore, s_m^2 represents the variance of the generic node's load respect to the network average. Note that a variance greater than zero does not necessarily mean a suboptimal load balancing because, given the discrete nature of the loads and the atomic nature of the tasks, a perfectly balanced load distribution can not always be achieved.

The results show how not only the convergence time is influenced by the network dimensions but that it is also affected by the weight range. This is simply due to the fact that it is easier to balance a load composed by many small objects rather than a load composed by few large objects.

	Maximum weight 10				Maximum weight 100			
	FC net		Linear net		FC net		Linear net	
n	T_m	s_m^2	T_m	s_m^2	T_m	s_m^2	T_m	s_m^2
Algorithm GQG-IPP								
5	33	0.16	101	0.16	46	0.16	148	0.16
10	109	0.16	883	0.16	150	0.16	1393	0.16
20	303	0.16	7354	0.16	529	0.16	1149	0.16
Algorithm GQG-H								
5	34	0.16	102	0.81	56	2.37	148	9.61
10	111	0.16	879	1.96	232	0.81	1460	10.2
20	379	0.25	7171	4	818	0.36	11698	9

TABLE I

SIMULATION RESULTS WITH ARBITRARY LOAD.

Furthermore the simulations show that both algorithms perform much better when the network is well connected, due to the unnecessary role of the swaps in this kind of network. Finally GQG-IPP always performs better than GCG-H as expected.

What cannot be directly seen in the first two tables, is the distance of the final solution from the optimal one. In all randomly generated cases we considered, when the network is sufficiently large we have found that GQG-IPP always reaches the optimum consensus. To show this, we performed a series of simulations on fully connected networks composed by 5, 10 and 20 nodes in which the initial load configuration is chosen so that the optimal solution assigns the same load to all nodes (with an average of 10 tasks per node). Thus the optimal solution is characterized by a value of the load variance $s_m^2 = 0$. The results of these simulations are shown in Table II. The load's variance from the average over 1000 realization is exactly 0 for GQG-IPP, while it is slightly higher for GQG-H. The software used to generate these simulations is available on the web [16].

VI. CONCLUSIONS

In this paper we presented a generalized version of the quantized gossip algorithm in [10] that deals with the load discretization. We demonstrated its convergence properties and found general bounds of the expected balancing performance under no assumptions on the initial load

	GQG-IPP		GQG-H	
n	T_m	s_m^2	T_m	s_m^2
5	51	0	51	1.44
10	175	0	194	0.49
20	595	0	632	0.25

TABLE II

SIMULATION RESULTS WITH PREDEFINED LOAD.

configuration. We provided simulations that show the practical optimality of the algorithm in the vast majority of cases. We have shown that partial swap are a key issue to ensure the good performance of the proposed procedure.

We also proposed a simple sub-optimal heuristic and demonstrated its convergence properties within the same bounds of GQG-IPP. The slight performance decrement of the heuristic (as shown by our simulations) has a trade-off in its simplicity that does not require at each step the solution of an integer programming problem.

REFERENCES

[1] D. Bauso, L. Giarré, and R. Pesenti. Mechanism design for optimal consensus problems. In *Proc. 45th IEEE Conf. on Decision and Control*, pages 3381–3386, San Diego, CA, USA, December 2006.

[2] V.D. Blondel, J.M. Hendrickx, A. Olshevsky, and J.N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proc. 44th IEEE Conf. on Decision and Control*, pages 2996–3000, Seville, Spain, December 2005.

[3] A. Cortes, A. Ripoll, M.A. Senar, P. Pons, and E. Luque. On the performance of nearest-neighbors load balancing algorithms in parallel systems. In *Proc. of the 7th Euromicro Workshop on Parallel and Distributed Processing*, pages 170–177, Funchal, Portugal, February 1999.

[4] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. of Parallel and Distributed Computing*, 7:279–301, 1989.

[5] B. Ghosh, F.T. Leighton, B.M. Maggs, S. Muthukrishnan, C.G. Plaxton, R. Rajaraman, A.W. Richa, R.E. Tarjan, and D. Zuckerman. Tight analyses of two local load balancing algorithms. *SIAM J. on Computing*, 29(1):29–64, 2000.

[6] B. Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *J. of Computer and Systems Sciences*, 53(3):357–370, 1996.

[7] M. Herlihy and S. Tirthapura. Self-stabilizing smoothing and balancing networks. *Distributed Computing*, 2005.

[8] M.E. Houle, A. Symvonis, and D.R. Wood. Dimension exchange algorithms for token distribution on tree-connected architectures. *J. of Parallel and Distributed Computing*, 64:591–605, 2004.

[9] M.E. Houle, E. Tempero, and G. Turner. Optimal dimension exchange token distribution on complete binary trees. *Theoretical Computer Science*, 220:363–377, 1999.

[10] A. Kashyap, T. Başar, and R. Srikant. Consensus with quantized information updates. In *Proc. 45th IEEE Conf. on Decision and Control*, pages 2728–2733, San Diego, CA, USA, December 2006.

[11] X. Lin and S. Boyd. Fast linear iterations for distributed averaging. *Systems and Control Letters*, 53:65–78, 2004.

[12] R. Olfati-Saber. Flocking for multi-agent dynamic system: Algorithms and theory. *IEEE Trans. on Automatic Control*, 51:401–420, 2006.

[13] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Trans. on Automatic Control*, 49:1520–1533, 2004.

[14] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. In *Proc. of the 39th Annual Symposium on Foundations of Computer Science*, pages 694–703, Palo Alto, CA, USA, November 1998.

[15] R. Subramanian and I.D. Scherson. An analysis of diffusive load-balancing. In *Proc. of the 6th annual ACM Symposium on Parallel Algorithms and Architectures*, pages 220–225, Cape May, New Jersey, USA, 1994.

[16] *http//www.diee.unica.it/∼giua/ADD/07CDC/*.