Brief Announcement: Local Deal-Agreement Based Monotonic Distributed Algorithms for Load Balancing in General Graphs

Yefim Dinitz, Shlomi Dolev, and Manish Kumar(✉)

Ben-Gurion University of the Negev, Be'er Sheva, Israel
{dinitz,dolev}@cs.bgu.ac.il, manishk@post.bgu.ac.il

Abstract. In computer networks, participants may cooperate in processing tasks, balancing working loads among them. The distributed load balancing problem is well-known. We present local algorithms solving it based on a short *deal-agreement communication*. Unlike the previous algorithms, they converge *monotonically*, always providing a better feasible state as the execution progresses. Our synchronous algorithms achieve ϵ-Balanced state for the continuous setting in time $O(nD\log(nK/\epsilon))$ and 1-Balanced state for the discrete setting in time $O(nD\log(nK/D)+nD^2)$, for *general graphs* in the worst case, where n is the number of nodes, K is the initial discrepancy, and D is the graph diameter. We also suggest an *asynchronous* load balancing algorithm solving the problem in time $O(nK^2)$ for general graphs, and its *self-stabilizing* version.

Keywords: Distributed algorithms · Deterministic · Load balancing · Self-stabilization · Monotonic

1 Introduction

The distributed load balancing problem is defined when there is an undirected network (graph) of computers (nodes), each one assigned a non-negative working load, and they like to balance their loads. If nodes u and v are connected by an edge, then any part of the load of u may be transferred over that edge from u to v, and similarly from v to u. The information at the nodes is local, and the only way to get more knowledge on the graph is by communicating with its neighbors. The application and scope include grid computing, clusters, and clouds.

The accepted global measure for the deviation of a current state from being balanced is its *discrepancy*, defined as $K = L_{max} - L_{min}$, where L_{max} (L_{min})

This research was (partially) funded by the Office of the Israel Innovation Authority of the Israel Ministry of Economy under Genesis generic research project, the Rita Altura trust chair in computer science, and by the Lynne and William Frankel Center for Computer Science.

is the currently maximum (minimum) node load in the graph. An alternative, local way to measure the deviation is the maximal difference of loads between neighboring nodes: a state is *ϵ-Balanced* if that difference is at most ϵ. In the discrete problem setting, all loads and thus also all transfer amounts should be integers; in the continuous one, transfer amounts are arbitrary. In this paper, we concentrate on *deterministic* algorithms solving the problem in *a worst case* time *polynomial in the global input size*, that is in the number n of graph nodes and in the logarithm of the maximal load (though we deviate from polynomiality for the asynchronous model).

The research on the load balancing problem began from the papers of Cybenko [4] and Boillat [3]. Both are based on the concept of *diffusion*: at any synchronized round, every node divides its load *equally* among its neighbors and itself. As a rule, the case of *d-regular* graphs is considered; only laconic remarks on a possibility to generalize the results to the case of general graphs appear in the literature. Markov chains and ergodic theory are used for deriving the rate of convergence. In the discrete setting, diffusion methods require rounding of every transferred amount, which makes the analysis harder; Rabani et al. [9] made a substantial advancement in that direction; their the time bound for reaching the discrepancy of ϵ in the worst case is $O\left(\frac{\ln(Kn^2/\epsilon)}{(1-\lambda)}\right)$, where λ is the second largest eigen-value of the diffusion matrix. The diffusion approach is popular in the literature. The alternative methods are *mathching* (see, e.g., [10]) and *balancing circuits* (see, e.g., [2,9]). For the discrete setting and the considered computational model, all those approaches do not achieve neither a constant final discrepancy, nor a constant-balanced state. Many suggested algorithms cannot be stopped at any time, since intermediate solutions either might include negative node loads, or might be worse than previous ones. Almost all papers on load balancing use the synchronous distributed model. The only theoretically based approach suggested for *asynchronous* distributed setting is turning it to synchronous by appropriately enlarging the time unit, see e.g., [1].

We suggest using the distributed computing approach based on *short agreement between neighboring nodes* in load balancing. We develop *local* distributed algorithms, with no global information collected at the nodes; the advantage is that the actual time of an algorithm run can be quite small, if the problem instance is lucky. We say that a load balancing algorithm is *monotonic* if the maximal load value never increases and the minimal load value never decreases. Such algorithms produce a not worse feasible state at each step of the execution, and thus are *anytime* in the sense of [5,8]. Our main results on load balancing are as follows, where D is the graph diameter, and ϵ is an arbitrary constant.

– In the continuous setting, the first synchronized deterministic algorithm for *general graphs*, which is monotonic and works in time $O(nD\log(nK/\epsilon))$.
– In the discrete setting, the first deterministic algorithms for *general graphs* achieving a 1-Balanced state in time depending on the initial discrepancy logarithmically. It is monotonic and works in time $O(nD\log(nK/D) + nD^2)$.
– The first asynchronous anytime algorithm, and its self-stabilizing version.

The full version of this paper can be found in arXiv [6].

2 Monotonic Distributed Load Balancing Algorithms

Algorithm 1: Synchronous Single-Proposal Algorithm: Continuous

```
   Input: An undirected graph $G = (V, E, load)$
1  Execute forever do
2      for every node $u$ do
3          if $u$ has at least one neighbor with a strictly smaller load then
4              find the neighbor, $v$, with the minimal load
5              $u$ sends to $v$ a transfer proposal of $(load(u) - load(v))/2$
6      for every node $u$ do
7          if there is at least one transfer proposal to $u$ then
8              find a neighbor, $w$, proposing to $u$ the transfer of maximum value
9              node $u$ makes a deal: increases its load by the value proposed by $w$
                 and informs node $w$ on accepting its proposal
10     for every node $u$ do
11         node $u$ updates its load w.r.t. the deal made on its proposal, if accepted,
             and sends the current value of $load(u)$ to every its neighbor
```

Let us begin with the synchronous model. Algorithm 1 solves the continuous load balancing problem. It is composed of three-phase rounds, one phase upon the global clock tick, cyclically. At each round, each node sends a transfer proposal to *at most one* of its neighbors. In reply, each node accepts a single proposal among those sent to it, if any. (Each node may finally both send and get load at same round.)

The analysis of Algorithm 1 is based on node potentials. Let L_{avg} be the average value of *load* over V. We define potentials $p(u) = (load(u) - L_{avg})^2$ for any node u, and $p(G) = \sum_{u \in V} p(u)$ for entire G. Any transfer of load l from u to v in our algorithms satisfies $load(u) - load(v) \geq 2l > 0$. For any such transfer, we prove that it decreases $p(G)$ by at least $2l^2$. The central point of our analysis is the following statement.

Lemma 1. *If the discrepancy of G at the beginning of some round is K, the potential of G decreases after that round by at least $K^2/2D$.*

Proof. Consider an arbitrary round. Let x and y be nodes with load L_{max} and L_{min}, respectively, and let P be a *shortest* path from y to x, $P = (y = v_0, v_1, v_2, \ldots, v_k = x)$. Note that $k \leq D$. Consider the sequence of edges (v_{i-1}, v_i) along P, and choose its sub-sequence S consisting of all edges with $\delta_i = load(v_i) - load(v_{i-1}) > 0$. Let $S = (e_1 = (v_{i_1-1}, v_{i_1}), e_2 = (v_{i_2-1}, v_{i_2}), \ldots, e_{k'} = (v_{i_{k'}-1}, v_{i_{k'}}))$, $k' \leq k \leq D$. Observe that by the definition of S, interval $[L_{min}, L_{max}]$ on the load axis is covered by intervals $[load(v_{i_j-1}), load(v_{i_{j-1}})]$, since $load(v_{i_1-1}) = L_{min}$, $load(v_{i_{k'}}) = L_{max}$, and for any $2 \leq j \leq k'$, $load(v_{i_{j-1}}) \geq load(v_{i_j-1})$. As a consequence, the sum of load differences $\sum_{j=1}^{k'} \delta_{i_j}$ over S is at least $L_{max} - L_{min} = K$.

Since for every node v_{i_j}, its neighbor v_{i_j-1} has a strictly lesser load, the condition of the first **if** in Algorithm 1 is satisfied for each v_{i_j}. Thus, each v_{i_j} proposes a transfer to its minimally loaded neighbor; denote that neighbor by w_j. Note that the transfer amount in that proposal is at least $\delta_{i_j}/2$. Hence, *the sum of load proposals issued by the heads of edges in S is at least $K/2$.* By the algorithm, each node w_i accepts the biggest proposal sent to it, which value is at least $\delta_{i_j}/2$. Consider the simple case when all nodes w_j are different. Then, the total decrease of the potential at the round, Δ, is at least $\sum_j 2(\delta_{i_j}/2)^2$. By simple algebra, for a set of at most D numbers with a sum bounded by K, the sum of numbers' squares is minimal if there are exactly D equal numbers summing to K. We obtain $\Delta \geq D \cdot 2(K/2D)^2 = K^2/2D$, as required.

The rest of the proof reduces the general case to the simple case as above.

We prove that Algorithm 1 is monotonic, and that it arrives at the discrepancy of at most ϵ in time $O(nD\log(nK/\epsilon))$.

The algorithm for the discrete setting differs by the rounding of proposal values only. Its analysis up to the arrival at a discrepancy of at most $2D$ is similar; the rest of its execution is analyzed separately. Also that algorithm is monotonic, and it arrives at a 1-Balanced state in time $O(nD\log(nK/D)+nD^2)$.

We believe that the running time bounds of deal-agreement distributed algorithms for load balancing could be improved by future research. This is since the current bounds are based on analyzing only a single path at each iteration.

Multiple-Proposal Load Balancing Algorithm. We suggest also the monotonic synchronous deal-agreement algorithm based on *multiple proposals*. There, each node may propose load transfers to several of its neighbors with smaller load, aiming to *equalize the loads* in its neighborhood as much as possible. We formalize this as follows. Consider node p and the part $\mathcal{V}_{less}(p)$ of its neighbors with loads smaller than $load(p)$. Node p proposes load transfers to some of the nodes in $\mathcal{V}_{less}(p)$ in such a way that if all its proposals would be accepted, then the resulting minimal load in the node set $\mathcal{V}_{less}(p) \cup \{p\}$ will be maximal. (Compare with the scenario, where we pour water into a basin with unequal heights at its bottom: the flat water surface will cover the deepest pits.) Performing deals in parallel with several neighbors has a potential to yield faster convergence in practice, as compared with the single-proposal algorithm.

Asynchronous Load Balancing Algorithm. The asynchronous version of the load balancing algorithm is based on repeated enquiries of the load of the neighbors and whenever proposing a deal to a neighbor with a lower load, wait for the acknowledgment of the proposal acceptance or rejection prior to reexamination. In more detail, our asynchronous load balancing algorithm is based on distributed proposals. There, each node may propose load transfers to several of its neighbors by computing $\mathcal{PV}_{less}(p)$, which is part of $\mathcal{V}_{less}(p)$. $\mathcal{PV}_{less}(p)$ is the resulting minimal loaded node set whose load is less than *TentativeLoad* after all proposal gets accepted. While sending the proposal, each node sends the value of *LoadToTransfer* (load which can be transferred to neighboring node) and *TentativeLoad* (load of the node after giving loads to its neighbors) with all set of nodes in $\mathcal{PV}_{less}(p)$. After receiving a proposal, the node sends an acknowledgment to the sender node; the sender node waits for an acknowledgment from all nodes of $\mathcal{PV}_{less}(p)$. The asynchronous algorithm ensures that the local computation between two nodes is assumed to be before the second communication starts. Consider an example where a node q of $\mathcal{PV}_{less}(p)$ receives a proposal and the deal happens between node p and node q. In this case, *TentativeLoad*(p) is always greater than the load of node q (when q responds to the deal) because node p is waiting for acknowledgments from all nodes of $\mathcal{PV}_{less}(p)$.

Self-stabilizing Load Balancing Algorithm. The self-stabilizing load balancing algorithm is based on the asynchronous version, where a self-stabilizing data link algorithm is used to verify that eventually (after the stabilization of the data-link) whenever a neighbor sends and acknowledge accepting a deal, the invariant of load transfer, from a node with load higher than the load of the acknowledging node, holds. This solution can be extended to act as a super-stabilizing algorithm [7], gracefully, dealing with dynamic settings, where nodes can join/leave the graph anytime, as well as handle received/dropped loads.

References

1. Aiello, W., Awerbuch, B., Maggs, B.M., Rao, S.: Approximate load balancing on dynamic and asynchronous networks. In: 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 632–641 (1993)
2. Aspnes, J., Herlihy, M., Shavit, N.: Counting networks and multi-processor coordination. In: 23rd Annual ACM Symposium on Theory of Computing (STOC), pp. 348–358 (1991)
3. Boillat, J.E.: Load balancing and Poisson equation in a graph. Concurrency Pract. Experience **2**(4), 289–314 (1990)
4. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. J. Parallel Distrib. Comput. **7**(2), 279–301 (1989)
5. Dean, T.L., Boddy, M.S.: An analysis of time-dependent planning. In: 7th National Conference on Artificial Intelligence, pp. 49–54 (1988)
6. Dinitz, Y., Dolev, S., Kumar, M.: Local deal-agreement based monotonic distributed algorithms for load balancing in general graphs. CoRR abs/2010.02486 (2020)
7. Dolev, S., Herman, T.: Superstabilizing protocols for dynamic distributed systems. Chic. J. Theor. Comput. Sci. **1997**, 3.1–3.15 (1997)
8. Horvitz, E.: Reasoning about beliefs and actions under computational resource constraints. Int. J. Approx. Reason. **2**(3), 337–338 (1988)
9. Rabani, Y., Sinclair, A., Wanka, R.: Local divergence of Markov chains and the analysis of iterative load balancing schemes. In: 39th Annual Symposium on Foundations of Computer Science, (FOCS), pp. 694–705 (1998)
10. Sauerwald, T., Sun, H.: Tight bounds for randomized load balancing on arbitrary network topologies. In: 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 341–350 (2012)