Network Load Balancing Using Modular Arithmetic Computations

Stavros I. Souravlas and Angelo Sifaleras

Abstract Load-balanced routing has attracted considerable attention, especially in the recent years, where huge data volumes are carried over the computer networks. It is particularly important for non-all-to-all networks, where there is no direct communication between all the nodes of the network.

Telecommunication and network systems constitute complex dynamic systems with an ever-increasing number of users and network services. It has become apparent that, new routing demands can not be easily satisfied by conventional routing methods. Thus, intelligent optimization methods (e.g., nature-inspired methodologies) have arisen to improve network efficiency.

This paper presents a computational method that is based on modular arithmetic for achieving dynamic load balancing on data networks. The proposed algorithm organizes the overall communication into equal-sized packets, it divides the communication into a series of communication steps between the network nodes, and performs packet transfer. The last section includes discussion on the main costs each network routing operation inures: the data movement cost, the load information cost and the data reordering cost.

1 Introduction

Recently, telecommunication and network systems have become more complex and network problems have also been increased [1]. Perhaps, the most important problem that today's networks are facing is the efficiency improvement, in terms of min-

Stavros Souravlas,
University of Macedonia, Department of Applied Informatics, 156 Egnatias Str., Thessaloniki 54636, Greece, e-mail: sourstav@uom.gr

Angelo Sifaleras
University of Macedonia, Department of Applied Informatics, 156 Egnatias Str., Thessaloniki 54636, Greece, e-mail: sifalera@uom.gr

Please cite this paper as:

Souravlas S. and Sifaleras A., "Network load balancing using modular arithmetic computations", in P. Vlamos (Ed.), *Advances in Experimental Medicine and Biology*, Springer, Vol. 988, pp. 271-280, 2017.

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-56246-9_22

imizing the latencies, better utilizing the network sources like links, memory, etc and maximizing the overall throughput. To successfully deal with all these issues, researchers have developed several routing algorithms [16]. However, the increasing number of users and the continuously larger data volumes carried over all networks, necessitate the improvement of these techniques, so that many applications from different fields like scientific computations, medicine, biology [4, 11], etc. can be satisfied.

One of the most important methods towards increasing network efficiency in contemporary network systems is load-balancing. Load balancing is a technique used to keep the packets transmitted across the network as balanced as possible, such that all the communication links are equally loaded during a communication process and accordingly, the nodes involved receive almost equal data volumes at the end of the process. Due to its importance, load balancing has achieved considerable attention from many researchers. Several methods have been proposed in the literature to assure of balanced communication. These methods are divided into two main categories: *static* and *dynamic* ([3], [5], [9], [15]). In static methods, communication scheduling is predefined. On the contrary, a dynamic method takes into account the state of the system and the running application's parameters before taking any scheduling decision.

Generally, routing incurs three important costs: (1) *Data movement cost:* The cost of transferring units of data. Apparently, if good balancing is achieved, the data movement cost should be approximately the same for all network links at a time, (2) *Load information cost:* This cost incurs when a node requests information regarding the system's state. for example, availability of links, nodes with less load, etc., (3) *Data reordering cost:* This cost incurs when a node receives packets from different sources and has to combine all these packets in a single data packet. Then, a data reordering phase has to take place in the node's memory, increasing the total cost of routing, especially if this newly formed data packet has to be further transferred.

A number of network applications require that data elements from some nodes are packed together and form a bigger data packet delivered to a new node. An example is the Fast Fourier Transform (FFT), which necessitates that some network nodes receive and buffer packets of data samples while others process the data sets. Thus, a series of data samples may need to be combined into a single packet, which is then sent to a node for further processing. FFT has many applications, for example in Body Sensor Networks (BSN), a wireless sensing technology for healthcare applications.

In this work, we propose a load-balancing technique based on modular arithmetic, which applies in cases where larger data packets, formed by the data elements of some source network nodes, are subsequently transferred to a target node. The goal is to assure that, at any time, the data volumes carried over the network are equal between the links, thus the network performance is improved. The remaining of this paper is organized as follows: Section 2 briefly describes the related work. Section 3 presents the proposed load-balancing technique and performs computational analysis. Finally, Section 4 concludes the paper and offers aspects for future research.

2 Related Work

Numerous load-balancing techniques over data networks have been proposed in the literature. Some of them consider the load balancing problem as the problem of dividing a graph of all nodes to a finite number of partitions which are mapped then on different servers. To do this, some strategies take advantage of the geometric information of the nodes [17] or their graph connectivity [14]. Others investigate the possibility of controlling the stability of the load balancing system when the system scaling increases and propose techniques [12], while others focus on achieving the dynamic load balancing on MPI applications by creating new task scheduling techniques [13].

In [15], two load balancing algorithms are proposed; MELISA and LBA. A set of equations is defined, which is used to compute the loading parameters, like arrival and service rates. In MELISA, every node uses the model to estimates its arrival rate, service rate, and load, and then the load of its neighbors. These estimations are performed at predefined estimation time periods T_c. Then, each network node tries to distribute its load among its neighbors in such a way, that the processing of this load finishes almost simultaneously. For smaller networks, LBA (Load Balancing on Arrival) was developed. This algorithm achieves balancing by transferring data upon arrival. This approach is much faster, because it does not have to spend time for the estimation time period. All estimations are performed upon data arrival.

In [7], a novel set of load balancing protocols named ECLB (Earliest Completion Load Balancing) was proposed. ECLB includes five different protocols: (1) No Load Balancing, a strategy that does not balance the load, but transfers based on the total latency, (2) Basic Load Balancing, a strategy that transfers to the node with the lowest load, (3) Nearest Neighbor Load Balancing, a technique that transfers data from a source to its nearest neighbor, (4) Earliest Completion Load Balancing, a technique that estimates the time it takes to complete a transfer based on the neighbors' load and latency and selects the target node based on this estimation, and (5) Random Neighbor Load Balancing, a technique used to randomly chose a node from a list of available nodes. Randomized protocols were also proposed be Karger et al. [8].

In [6], the authors proposed a a load-balancing technique that includes two basic load balancing operations: `NBRASJUST` and `REORDER`. The first is used to handle the sharing of the calling node with its neighbor, while the second moves a node to another location for sharing reasons. After these operations are performed, the network information regarding the load changes and routing is accordingly adjusted. An improved extension of this work was presented by Chawacheat at al., in [2], where for each data insertion or deletion in a node, the two balancing operations are called only once, as opposed to [6], where the calls are arbitrarily many.

Konstantinou et al. [10] presented a wave-based model to balance the load, after examining the load of the neighboring nodes. The idea is that a set of nodes transfers their load to a common neighbor and then share the load of the last node of the wave.

3 Load-Balancing Using Modular Arithmetic

In this section, the load-balancing routing algorithm will be described and afterward some performance issues will be discussed.

3.1 The Routing Strategy

To set up our problem, we initially define some variables. The network consists of N nodes and each node stores data in the form of data packets, where each packet consists of s elements (these packets will be referred to as initials), typically each element is a byte or multiple bytes. At any time, the nodes may need to build up larger data packets consisting of t elements (these packets will be referred to as finals), $s < t$. In other words, these larger data packets will contain data elements from a number of nodes.

Now, let us consider a big data set, with data packets equally distributed across the network nodes, say in a round robin fashion. For example newly produced data samples, stored in a round robin fashion throughout the network. Then, a data element indexed by i of a data packet indexed by l and local position inside the packet indexed by x, stored in node $n_p,\ p \in [0 \ldots N-1]$, can be described by $i = (lN + n_p)s + x$. Now, if we assume that larger data packets of size t have to be created to satisfy a specific application, then the data element just mentioned will become the data element indexed by j of a new data packet indexed by m and local position inside the packet indexed by y, stored in node $n_q,\ q \in [0 \ldots N-1]$. Thus, it will be described by $j = (mN + n_q)t + y$. Under these assumptions, we can describe this transfer by:

$$(lN + n_p)s + x = (mN + n_q)t + y \tag{1}$$

The number of variables (l, m, x, y) that solve Eq.(1) is the number of elements that need to be transferred between nodes n_p and n_q. Apparently, the data packets of size t will be formed by data elements from different sources. So, our problem is to define, for any values of $N, s,$ and t, which data elements will be the initial ones in the final packets, which elements will follow, and which will be the last ones. Proposition 1 gives us the starting point:

Proposition 1: The first elements (or bytes) of every final packet to be stored in n_q satisfies:

$$-x \bmod N = \Phi, \tag{2}$$

where Φ is an integer mod N.

Proof: Eq.(1) can be rewritten as $lNs + n_p s + x = mNt + n_q t + y$, so it follows that $mNt - lNs = (x - y) + n_p s - n_q t$. However, N divides Ns and Nt, so $mNt - lNs$ is

a multiple of N, or $mNt - lNs = \lambda N$, where λ is an integer. By setting $\phi = x - y$, Eq.(1) is rewritten as:

$$\lambda N - \phi = n_p s - n_q t \tag{3}$$

Now, if we divide (3) by N and get the modulo:

$$\begin{aligned} & (\lambda N - \phi) \bmod N = (n_p s - n_q t) \bmod N \\ \Rightarrow\ & (\lambda N) \bmod N - (\phi \bmod N) = (n_p s - n_q t) \bmod N \\ \Rightarrow\ & (-\phi) \bmod N = (n_p s - n_q t) \bmod N \\ \Rightarrow\ & (y - x) \bmod N = (n_p s - n_q t) \bmod N \end{aligned} \tag{4}$$

Because y is indexing final packets, it follows that a value of $y = 0$ is indexing the first bytes of a final data packet. By setting $y = 0$ and $\Phi = n_p s - n_q t$ to Eq.(4), we complete the proof. □

Thus, the first bytes of the final data packets to be stored to all n_qs will be found by the solutions of the following system of equations and constraints:

$$(y - x) \bmod N = (n_p s - n_q t) \bmod N, \text{ for all } (p,q) \in [0 \ldots N-1] \tag{5}$$

$$x \leq s - 1 \text{ (initial packet size is } s) \tag{6}$$

$$y = 0 \tag{7}$$

The number of solutions for Eq.(5-7) is given by the Initial Data Volume Transmitted $IDVT$ function:

$$IDVT(n_p, n_q) = \{(l,m,x,y) : Eq.(5-7)\} \tag{8}$$

Now, we can easily find the nodes that will contribute the remaining bytes of the final packets, as described by Proposition 2:

Proposition 2: The remaining elements (or bytes) of every final packet to be stored in n_q are found by repeatedly solving a system similar to the one described by Eq.(5-7), until no more solutions can be found.

Proof: Initially, let us divide a final packet of size t in three parts:

- Starting part t_{start}, that contains the first elements, as computed by Eq.(5-7)
- Middle part t_{middle}, that contains all the data elements not in t_{start} and not in t_{end} (see below)
- Ending part t_{end}, the data elements at the very end of the packet

Now, for any node n_q, t_{start} has been found by solving Eq.(5-7). The size of t_{start}, that is, the number of its bytes is given by IDVT (Eq.8). So these elements will

be found in positions indexed from 0, to $DVT(n_p,n_q)-1$ of the final packet, and there remains to be found other $t-DVT(n_p,n_q)$ elements. It follows, that to find the middle part of a final packet for n_q, we have to solve a similar system like the one described in Eq.(5-7), but with changes regarding y:

$$(y-x) \bmod N = (n_p s - n_q t) \bmod N, \text{ for all } (p,q) \in [0 \ldots N-1] \quad (9)$$

$$x \leq s-1 \quad (10)$$

$$y \in [IDVT(n_p,n_q) \ldots t-1] \text{ (final packet size is } t) \quad (11)$$

Similarly, the Middle Data Volume Transmitted $MDVT$ function computes the number of solutions for Eq.(9-11):

$$MDVT(n_p,n_q) = \{(l,m,x,y) : Eq.(9-11)\} \quad (12)$$

Now, let us express the middle part t_{middle} as the sum of $t_{M1}+t_{M2}+\ldots t_{Mk}$, where each of these summands is the number of the solutions (MDVT) for Eq.(9-11), for a constant n_q and changing n_ps, from 0 to $N-1$. Apparently, as we keep finding solutions, $t_{M1}+t_{M2}+\ldots t_{Mk}$ will reach $t-DVT(n_p,n_q)$, the number of elements apart from t_{start}, that need to be found for each final packet. Recall that the number of such solutions is finite, since x and y are limited by s and t, respectively. Thus, one of the solutions for Eq.(9-11) will also be t_{end}, the ending part of the packet and this completes the proof. □

Now, it is easy to organize the data transmission in such a way, that we achieve load balancing between the network links. We need a definition beforehand.

Definition 1: A *packet set* PS_C is a set of packets that have C elements (bytes), where C is an integer, $C \leq t$.

Now, to achieve load balancing, we follow four phases described below:

`Phase 1:` Solve Eq.(5-7) for all (n_p,n_q) that satisfy Eq.(4) to get all t_{start}s. These will have different IDVTs, so simply put all t_{start}s with equal IDVTs in the same packet set.

`Phase 2:` Transfer the packet sets found in `Phase 1`, in a series of communication steps, one packet set per step.

`Phase 3:` Solve Eq.(9-11) for all (n_p,n_q), to get all t_{middle}s and t_{end}s. Again, these will have different MDVTs, so simply put all t_{middle}s with equal MDVTs in the same packet set. Do the same for all t_{end}s.

`Phase 4:` Repeat `Phase 2`, for the packet sets found in `Phase 3`.

From Steps 2 and 4, it is clear that same packet sizes are transferred between all the network nodes, thus we achieve load balancing in all network links. The next paragraph will give more details about the performance of our strategy, regarding load-balancing. Algorithm 1 gives the pseudocode of the proposed algorithm.

input : Number of nodes N, initial packet size, s, final packet size, t
output: Load balanced packet transmission, final packets of size t stored in the nodes of the network.

1 *Phase 1*;
2 Read $N, s,$ and T **for** $p \leftarrow 0$ **to** $N-1$ **do**
3 | // *Keep* n_p *constant and run through all* n_q*s*
4 | **for** $q \leftarrow 1$ **to** N **do**
5 | | **if** *the pair* (n_p, n_q) *satisfies Eq.(4)* **then**
6 | | Solve Eq.(5-7) for (n_p, n_q)
7 | | Find $C = IDVT((n_p, n_q))$
8 | | Add communication (n_p, n_q) to packet set PS_C
9 | | **else** Move to next pair (n_p, n_q);
10 | | // *All packet sets have been created*;
11 | **end**
12 **end**
13
14 *Phase 2*;
15 *//(Data Transmission, one communication step per packet set)*
16 **for** $C \leftarrow 1$ **to** t **do**
17 | **if** PS_C *exists* **then** Implement all communications added to it (see Line 8)
18 | **else** Move to next packet set C;
19 | // *All nodes now have the first bytes of their final packets*;
20 **end**
21
22 *Phase 3*;
23 Read $w = 0$ // (*w will keep the total number of solutions*) ;
24 **for** $p \leftarrow 0$ **to** $N-1$ **do**
25 | // *Keep* n_p *constant and run through all* n_q*s*
26 | **for** $q \leftarrow 1$ **to** N **do**
27 | | **while** $w < t - DVT(n_p, n_q)$ **do**
28 | | | Solve Eq.(9-11) for (n_p, n_q)
29 | | | Find $C = MDVT((n_p, n_q))$
30 | | | Add communication (n_p, n_q) to packet set PS_C
31 | | | $w = w + MVDT$
32 | | **end**
33 | | $w \leftarrow 0$ // (*to start the next pair*)
34 | **end**
35 | // *All packet sets have been created*
36 **end**
37
38 *Phase 4*;
39 Repeat *Phase 2*, for the packet sets found in *Phase 3*;

Algorithm 1: The four stages of the network load balancing algorithm

3.2 Discussion on Routing Costs and Other Properties

In this section we briefly discuss some cost issues and other properties of the proposed method. The proposed strategy has the following properties regarding the costs described in the introductory section:

1. It's *data movement cost* is at least equal to the cost of any other algorithm that handles the problem of transferring packets of different sizes over the network, like the one described in this paper, where different nodes have to contribute packets of different sizes to the formulation of larger packets. This cost is generally dictated by the number of communication steps multiplied by the number of bytes transferred per step. However, our strategy has two advantages that can be proven important: (a) Because each communication step includes equal-sized packets (load balancing), all communications per step are completed at approximately the same time, thus data link delays are avoided. These delays are present when packets of different sizes are transmitted across the network, so the "faster" links remain idle, waiting for the next transmission, and (b) Due to load balancing, it can work equally-well in cases when there are no direct connections between all the network nodes. The difference is that more communication steps through relay nodes would be required.
2. The system information requests can be handled like data packets, thus the *load information cost* can be restricted. There are two ways in which a node requests information regarding the system's state: (a) Through a centralized server, a case which is of no interest for our strategy, and (b) Through a request to the node it is interested. In the second case, even if the requests vary in terms of total cost, we can still use the proposed strategy to organize these submissions to equal-sized packets, especially in case where these requests are sent in a round-robin like fashion (for example, every node to its neighbor).
3. It's *data reordering cost* can be eliminated with minor changes. This is a big strength of the proposed scheme. We can organize the communication in such a way that the data packets are transmitted "in turn" so that the receiving nodes do not have to do any data reorganization upon receiving the packets. Attention must be paid, so that the selected packets that "take turns" to be transferred to n_qs must be further organized to packet subsets.

Two more important properties are worth to be mentioned [2, 8]:

1. The minimum data load transferred never decreases. Indeed, at any time there is a minimum load on the links, varied from 1 to $t-1$ bytes (assuming that there are at least two contributing nodes n_p to each final packet).
2. In cases where all the nodes are both sources and targets, their load after transmission (the packets stored in their memories) also remains balanced.

4 Conclusions- Future Work

This work proposed a novel network load balancing algorithm in cases where different nodes have to unequally contribute to the formulation of larger packets, that must be transmitted across the network. This computational method uses modular arithmetic to compute the bytes transmitted and it is composed of four stages: In the first stage, the first bytes of each packet are found, in the second stage, these bytes

are transmitted to their destination, in the third stage the middle and last bytes of these packets are found, and in the fourth and last stage, these bytes are transmitted. The first and third stage involve some type of grouping the packets into sets of equal sizes, thus achieving load balancing.

Our future work includes testing on real networks with real application data. Another challenge is to test the algorithm in real network topologies and make comparison. Computer networks which are not fully interconnected are of particular interest, because some type of relaying will be required. Finally, the development of other mathematical models on the same problems is also a subject that attracts attention.

References

1. Bhatia, N., Kundra, R., Chaurasia, A., Chandra, S.: Load Balancing Using Hybrid ACO - Random Walk Approach, pp. 402–412. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
2. Chawachat, J., Fakcharoenphol, J.: A simpler load-balancing algorithm for range-partitioned data in peer-to-peer systems. Networks **66**(3), 235–249 (2015)
3. Deng, Y., Lau, R.W.H.: On delay adjustment for dynamic load balancing in distributed virtual environments. IEEE Transactions on Visualization and Computer Graphics **18**(4), 529–537 (2012)
4. Dressler, F., Akan, O.B.: A survey on bio-inspired networking. Computer Networks **54**(6), 881–900 (2010)
5. El Kabbany, G.F., Wanas, N.M., Hegazi, N.H., Shaheen, S.I.: A dynamic load balancing framework for real-time applications in message passing systems. International Journal of Parallel Programming **39**(2), 143–182 (2011)
6. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04, pp. 444–455. VLDB Endowment (2004)
7. Haque, W., Toms, A., Germuth, A.: Dynamic load balancing in real-time distributed transaction processing. In: 16th IEEE International Conference on Computational Science and Engineering, pp. 268–274 (2013)
8. Karger, D.R., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '04, pp. 36–43. ACM, New York, NY, USA (2004)
9. Kim, S.S., Smith, E.A., Hong, S.J.: Dynamic Load Balancing Using an Ant Colony Approach in Micro-cellular Mobile Communications Systems, pp. 137–152. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
10. Konstantinou, I., Tsoumakos, D., Koziris, N.: Fast and cost-effective online load-balancing in distributed range-queriable systems. IEEE Transactions on Parallel and Distributed Systems **22**(8), 1350–1364 (2011)
11. Meisel, M., Pappas, V., Zhang, L.: A taxonomy of biologically inspired research in computer networking. Computer Networks **54**(6), 901–916 (2010)
12. Meng, Q., Qiao, J., Liu, J., Lin, S.: Research on the stability of load balancing algorithm for scalable parallel computing. In: Proceedings of the International Conference on Communication Software and Networks (ICCSN '09), pp. 309–312 (2009)
13. Nian, S., Guangmin, L.: Dynamic load balancing algorithm for mpi parallel computing. In: Proceedings of the International Conference on New Trends in Information and Service Science, NISS '09, pp. 95–99. IEEE Computer Society, Washington, DC, USA (2009)

14. Prasetya, K., Wu, Z.D.: Performance analysis of game world partitioning methods for multiplayer mobile gaming. In: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames '08, pp. 72–77. ACM, New York, NY, USA (2008)
15. Shah, R., Veeravalli, B., Misra, M.: On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments. IEEE Transactions on Parallel and Distributed Systems **18**(12), 1675–1686 (2007)
16. Sifaleras, A.: Classification of network optimization software packages. 3 edn., pp. 7054–7062. IGI Global, Hershey, PA (2015)
17. Steed, A., Abou-Haidar, R.: Partitioning crowded virtual environments. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST '03, pp. 7–14. ACM, New York, NY, USA (2003)