DISTRIBUTED LOAD BALANCING IN THE PRESENCE OF NODE FAILURE AND NETWORK DELAYS

J. Ghanem*, C. T. Abdallah*, M. M. Hayat* [1]
J. Chiasson, and J.D. Birdwell**** [2]

**Electrical & Computer Engineering Department*
MSC01 1100
University of New Mexico
Albuquerque, NM 87131-0001
{jean,chaouki,hayat}@eece.unm.edu

***Electrical & Computer Engineering Department*
University of Tennessee
Knoxville, TN 37996-2100
{chiasson,birdwell}@utk.edu

Abstract: In this paper, we present a new dynamic, and adaptive distributed load balancing algorithm. This algorithm is able to handle the loss of some computational nodes, the connectivity of the network, and the variations in tasks and transfer delays. An experimental verification of the algorithm is presented using PlanetLab. *Copyright ©2005 IFAC*

Keywords: Load Balancing, Distributed System, Planet-Lab, Network Control, Parallel Computing.

1. INTRODUCTION

Parallel computer architectures utilize a set of computational elements (CE) to achieve performance that is not attainable on a single processor, or CE, computer. A common architecture is the cluster of otherwise independent computers communicating through a shared network. To make use of parallel computing resources, problems must be broken down into smaller units that can be solved individually by each CE while exchanging information with CEs solving other problems.For a background on mathematical treatments of load balancing, the reader is referred to (Altman and Kameda, 2001). Effective utilization of a parallel computer architecture requires the computational load to be distributed more or less evenly over the available CEs. The qualifier "more or less" is used because the communications required to distribute the load consume both computational resources and network bandwidth. The distribution of computational load across available resources is referred to as the *load balancing* problem in the literature.

Load balancing policies have previously been proposed for categories such as local versus global, static versus dynamic, and centralized versus distributed scheduling (Cybenko, 1989; Lan *et al.*, 2001). Direct methods examine the global distribution of computational load and assign portions of the workload to resources before processing begins. Iterative methods examine the progress of the computation and the expected utilization of resources, and adjust the workload assignments periodically as computation progresses. A comparison of several methods is provided in (Willebeek-LeMair and Reeves, 1993; Ghanem, 2004).

[1] The work of J. Ghanem, C.T. Abdallah, and M. M. Hayat is partially supported by the National Science Foundation under Information Technology Research (ITR) grant No. ANI-0312611.
[2] The work of J. Chiasson and J.D. Birdwell is partially supported supported by the National Science Foundation under Information Technology Research (ITR) grant number ANI-0312182.

To introduce the basic approach to load balancing employed in this paper, consider a computing network consisting of n heterogeneous computers (nodes) all of which can communicate with each other. At start up, the computers are assigned a random number of tasks. The loads on various nodes quickly becomes uneven since applications may generates more tasks and the computational power of each node is varying. To balance the loads, each computer in the network sends (broadcasts) its current state (i.e. current queue size, computational power, etc.) to all other computers in the network. At every balancing instance (either calculated or predefined), each node uses information about its current state along with state information received from other nodes to calculate the new load distribution. Afterward, a portion of the node's current load scaled by a gain parameter K is transmitted to some other nodes as depicted by the load balancing policy. Notice that, networks connecting the different nodes (Wireless LAN, Internet, etc..) typically exhibit significant latency in the exchange of information and the actual transfer of loads. Thus, the balancing policy has no choice but to rely on dated information about the system of nodes, which, in turn, may result in unnecessary transfer of loads while certain nodes remain idle as the loads are in transit.

Previous work focused on analytically modelling the behavior of a system that undergoes load balancing and subsequently deriving adequate load balancing policies. The deterministic time model is a continuous-time described in terms of a nonlinear delay-differential system (Abdallah *et al.*, 2003; Birdwell *et al.*, 2004). It considers deterministic communication and transfer delays. Experimental evaluations of the model using its derived policy were conducted over a LAN and the Internet where optimization over the gain parameter K was performed (Ghanem *et al.*, 2004*a*; Chiasson *et al.*, 2005). The limitations and the overheads involved with the implementation of the deterministic-delay load-balancing policy in a random delay environment has been discussed by the authors in (Hayat *et al.*, 2004). Similarly, in (Dhakal *et al.*, 2003; Ghanem *et al.*, 2004*b*), Monte-Carlo studies and real-time experiments were conducted over the wireless LAN to study one-shot dynamic load-balancing scheme for distributed systems. Moreover, a regeneration-theory-based mathematical model, represented by a set of coupled integro-difference equations, for one-shot load-balancing policy as applied to a two-nodes distributed system was presented (Dhakal, 2003; Ghanem *et al.*, 2004*b*; Dhakal *et al.*, 2004).

In this paper, a new dynamic, adaptive, and distributed load-balancing policy is introduced based on the policies that were previously presented. This strategy takes into account the following three aspects: i) the connectivity among the nodes, ii) the computational power of each node, and iii) the throughput and bandwidth of the system. The occurrence of load-balancing instances, i.e. when to trigger the load distribution strategy, is not discussed in this paper since it is possible to employ one-shot load-balancing, multiple balancing instances, or any other scheme that would be suitable for the system at hand(Ghanem, 2004; Ghanem *et al.*, 2004*b*; Dhakal *et al.*, 2005; Dhakal *et al.*, 2004; Dhakal, 2003). Furthermore, exchanged information about the state of the nodes is still assumed to occur frequently with additional information to be discussed subsequently.

In Section 2, The new load distribution strategy is introduced and in Section 3, the computational methods of the adaptive parameters used by the policy are described. In Section 4, experimental evaluation of this new dynamic policy conducted over the Internet is presented. Section 5 contains our conclusions and direction for future research.

2. DYNAMIC AND ADAPTIVE POLICY DESCRIPTION

As observed in (Barabasi, 2003; Newman, 2003) , the Internet is not as completely connected as one might think. Add to that the fact that in distributed systems, a node may become unavailable or unreachable at any time due to a failure in the node itself, or in the network path leading to it. Therefore, the assumption made by the load-balancing policies that all nodes are accessible at any time is unrealistic especially in Internet scale distributed systems or in Ad-Hoc wireless networks. This will greatly affect the load-balances state of the system since loads assigned to unreachable nodes can never be delivered. The proposed algorithm can detect the connectivity in the system and decide accordingly what nodes may participate in the load sharing. At each load-balancing instance, the group of reachable nodes is referred to as the "current node set."

This load-balancing policy also takes into account the change in the computational performance of nodes and distributes the tasks accordingly. Actually, the system is not dedicated to the application at hand; other users may be using one or more nodes at a given time and therefore alter their computational power. Moreover, tasks are not considered identical, they may greatly differ in their completion time. These facts cannot be guessed a priori and assigning fixed computational power for each node is not always suitable. Therefore, the load strategy should be adaptive to these changes and be able to make decisions accordingly.

Moreover, transfer delays incurred when tasks are migrated from a node to another may be unexpectedly large and result in a negative impact on the overall system performance (Dhakal *et al.*, 2004). To avoid this situation, an a priori estimate of the transfer delays will help the policy at hand decide if the transfer is profitable and adequately decide on the size of load to migrate. These estimates should also be dynamically updated since delays may greatly vary during the system's life as shown for delays of class B in the classification of (Bovy *et al.*, 2002). Network delay and connectivity experiments conducted over Wireless LAN and the Internet can be found in (Ghanem, 2004; Georgatos *et al.*, 2001; Bovy *et al.*, 2002)

To further describe the policy, the following parameters are defined.

- n is the number of nodes present in the system.
- q_i denotes the number of tasks in the queue of node i.
- C_i denotes the average completion time of a task at node i in seconds.
- s_i is the average size of a task in bytes at node i when it is transferred.
- r_{ij} is the throughput or transfer rate in bytes/seconds between node j and node i. Note that in general $r_{ij} \neq r_{ji}$.
- $q_{j,av}$ is the average queue size calculated by node j based on its locally available information.
- $q_{j,excess}$ is the excess number of tasks at node j. (Detailed definition given later.)
- p_{ij} is the fraction of the excess tasks of node j to be transferred to node i as decided by the load balancing policy.

The first 5 parameters are assumed known at the time the load distribution process is triggered. That is, the update of these variables is done before the balancing instance is reached as will be described later. The general steps of the load-balancing policy invoked at node j are described below followed by a detailed description.

(1) ***Determine*** how many nodes are reachable (n_o) from node j.
Calculate the average queue size $q_{j,av}$ and the number of excess tasks $q_{j,excess}$ based on locally available information.

$$q_{j,av} = \frac{1}{n_o} \sum_{i:\ \text{node}(i)\ \text{reachable}} q_i \frac{C_i}{C_j}$$

$$q_{j,excess} = \begin{cases} (q_j - q_{j,av}) * K & \text{if } q_j > q_{j,av}, \\ 0 & \text{otherwise,} \end{cases}$$

where K is the predefined gain parameter. The algorithm ***exits*** if $q_{j,excess} = 0$.

(2) ***Determine*** how many (i.e., n'_o) and which nodes are below the average. These nodes will participate in the load sharing as viewed by node j.

(3) ***Calculate*** the optimal p'_{ij} fraction only for the n'_o nodes using the following formula:

$$p'_{ij} = \frac{q_{i,av} - (C_i/C_j) q_i}{\sum_{k \ni k \neq j} q_{k,av} - (C_k/C_j) q_k}$$

(4) ***Calculate*** p''_{ij} for the n'_o nodes, which is is the maximum portion of the excess load that is judged to be profitable when transmitted to node i.

$$p''_{ij} = \frac{(q_j - q_{j,excess}) C_j r_{ij}}{q_{j,excess} s_j}$$

Set $p_{ij} = \min(p'_{ij}, p''_{ij})$.

(5) *If* $\sum_i p_{ij} > a$ (a is a predefined threshold parameter between 0 and 1)
    transmit ($p_{ij} q_{j,excess}$) tasks to node i,
Otherwise
    repeat step 1 to step 4,
    assign the remaining fraction $(1 - \sum_i p_{ij})$ to nodes that have $p''_{ij} > p'_{ij}$ and call the newly assigned fractions p'''_{ij},
    transmit $(p_{ij} + p'''_{ij}) q_{j,excess}$ tasks to node i.

The first step determines the node set where the load distribution will take place from node j's perspective. This is achieved by checking the last time state information was received from each node. To test for connectivity to node i, the `local timestamp` of the last received state packet is compared to the current time decremented by three times the interval between 2 consecutive state broadcasts. That is, if the last state packet received from node i is one of the last three packet transmitted, node i is considered to be reachable from node j otherwise it is not included as part of the load-balancing node set since most likely, a load transmitted to this node will not be correctly delivered. Therefore, it is more suitable for the policy to base its calculations on nodes where migration of loads have higher probability of success. Note that at every instance of the load distribution process and for every node, the corresponding node set may end up with different elements according to the node's connectivity at that time. After that, the local queue average is calculated where each queue is scaled by the C_i/C_j factor that accounts for the difference in computational power of each node. In the queue excess calculations, a gain factor K is used since it is becoming a requirement in any policy that operates on large-delay systems where outdated state information is prominent. This fact appears in the experimental and theoretical studies conducted in earlier work (Abdallah *et al.*, 2003; Hayat *et al.*, 2004; Ghanem *et al.*, 2004*b*; Chiasson *et al.*, 2005).

The second and third steps employ the method introduced in (Chiasson *et al.*, 2005; Abdallah *et al.*, 2003; Ghanem *et al.*, 2004*a*) and used earlier in the literature in the Sender/Receiver Initiated Diffusion (SID and RID) policies (Saletore, 1990; Willebeek-LeMair and Reeves, 1993) to calculate the fractions p'_{ij}. This method is attractive since it only considers nodes that have queue sizes less than the average, and therefore results in as few connections as possible when the excess load is moved out of node j which makes the policy at hand more scalable. Moreover, this method leads to optimal results when the policy is triggered once on each node and where no delay is present in the system. Note that the formula is adjusted by the C_i/C_j factor.

The fourth step judges if the proportion p'_{ij} determined is worth transmitting to node i when transmission delays are present. This is accomplished by setting an upper bound on the maximum proportion of the excess load that is profitable when the exchange takes place. The task migration is said to be profitable if the time needed to transmit the load to the other end is less than the time needed to start executing the load on the current node (node j in this case). This statement is interpreted as follows,

$$\underbrace{p''_{ij}q_{j,excess}(s_j/r_{ij})}_{\text{transmission delay}} < \underbrace{(q_j - q_{j,excess})C_j}_{\text{start of load execution}}$$

Solving for p''_{ij}, we get the upper bound for p_{ij} as indicated in step 4. In the event that r_{ij} is not available, r_{ji} is used instead to provide an approximation of the bandwidth between node j and i. If neither parameter is available, step 4 is omitted for the node pair (i, j). The rate r_{ij} is detected and updated each time a load is transmitted from node i to node j as will be explained in the subsequent section where r_{ji} is received in the state information packet transmitted by node i to node j.

The fifth step is included for completion and can be omitted at any time. The rationale behind it is that after executing the algorithm, node j may find itself only transmitting a small portion (i.e., less than a variable a) of its excess load due to the delay restrictions. Therefore, it is suitable to reassign the remaining untransferred proportion to the nearest nodes (i.e., nodes reached through links of higher rate) in the hope that they may possibly have better connectivity to the system. This idea is inspired by the Small-World network model and the hubs/connectors concept where special nodes denoted as hubs have better connectivity to other nodes (Watts and Strogatz, 1998; Barabasi, 2003).

3. ADAPTIVE PARAMETERS COMPUTATION

In this section, the computation procedure for the dynamic parameters C and r_{ij} is explained. Note that the s_i parameter can be easily determined by averaging the tasks' sizes upon their creation.

Every time a task is completed by the application layer at node i, the C_i parameter is updated as follows,

```
if $C_i = 0$ then $C_i = T_{task}$
else $C_i = \alpha T_{task} + (1 - \alpha)C_i$
```

where T_{task} is the execution time of the last task and α is a gain parameter that affects the C_i term in its ability to reflect the current computational power of the system. Therefore, the values of α are critical to the stability and efficiency of the load-balancing policy. That is, assigning values in the high range of (0,1] to α may result in fluctuations in the C_i parameter which will have in turn an adverse impact on the decision of the load distribution, leading to bouncing of tasks back and forth between nodes. On the other hand, setting α to low values may not keep the load-balancing policy informed about the latest state of the node. Consequently, the value of α should be selected depending on the application used and the interference degree of external users. The update procedure could be easily modified to suit other methods.

The other parameter that is dynamically updated is the transfer rate r_{ij} incurred between node j and node i. On each data (or tasks) transmission, the transfer delay T_{delay} is recorded and is calculated by taking the difference between the instance the connection is initiated by node j and the instance node i acknowledgment of tasks reception is received by node j. Consequently the average transmission rate (rate $=$ T_{delay}/totalsize) is calculated, where 'totalsize' is the total size of the tasks migrated to node i. After each successful exchange of loads, the r_{ij} parameter is updated as follows:

```
if $r_{ij} = 0$ then $r_{ij} = \text{rate}$
else $r_{ij} = \beta * \text{rate} + (1 - \beta)r_{ij}$
```

This scheme is a simplified version of the method used to update the Round Trip Time (RTT) of the packet exchanged during a TCP connection where the delay variance is additionally taken into consideration (Institute, 1981). In the next section, β was set to 1/8 as suggested by (Jacobson, 1988) for the RTT update method.

Finally, both parameter C_j and r_{ij} are included in node j state information when transmitted to node i for all $i = 1, ..., n$, $i \neq j$.

4. EXPERIMENTAL EVALUATION

To test the performance of the newly proposed load-balancing policy denoted as lb2, a comparative experiment was performed between this policy and the strategy adopted by the stochastic model introduced in (Hayat *et al.*, 2004; Dhakal *et al.*, 2004) and referred to as lb1 (1). The experiments were conducted over PlanetLab(http://www.planet-lab.org) and the initial settings and parameters are shown in Table 1. Details about the implementation of the system used to experimentally test the policies are found in (Ghanem, 2004). The average network transfer rates for each path as detected by the system are shown in Table 2.

	node 1	node 2	node 3
Location	UNM (University of New Mexico)	Frankfurt - Germany	Sinica - Taiwan
Initial Distribution	600 tasks	250 tasks	100 tasks
Average Task Processing Time *C* (ms)	160	400	500

Average size of a task (Kbytes)	3.12
Interval between 2 load balancing instances (s)	1.5
Interval between 2 state transmissions (s)	10

Table 1. Parameters and settings of the experiment.

From - To	node 1	node 2	node 2
node 1	-	34.5 KB/s	73.3 KB/s
node 2	18.7 KB/s	-	45.4 KB/s
node 3	48.9 KB/s	20.2 KB/s	-

Table 2. Average transmission rates between the different nodes.

First, lb2 was evaluated for the gain values K between 0.3 and 1 with 0.1 incremental steps. The α parameter introduced in the previous section was set to 0.05 by running several experiments and observing the behavior of the C parameter. Note that, the first time the load-balancing process was triggered was after 20s from the start of the system and then the strategy was executed regularly at 10s intervals. This was done to ensure that the C parameter had enough time to adapt and reflect the current computational power of each node before the occurrence of any tasks migration between the nodes.

Second, lb1 was evaluated under the same conditions as lb2. Since there is a discrepancy between the computational power of the nodes (as shown in Table 1), the lb1 strategy was adjusted to account for these differences by scaling the queue sizes in the p_{ij} computation as follow,

$$p_{ij} = \begin{cases} \dfrac{1}{n-2}\left(1 - \dfrac{(C_{i,avg}/C_{j,avg})Queue(i)}{\sum_{k=1,k\neq j}^{n}(C_{k,avg}/C_{j,avg})Queue(k)}\right) \\ \qquad\qquad\qquad \text{if all } Q(i) \text{ are known.} \\ 1/(n-1) \qquad\qquad \text{otherwise} \end{cases} \tag{1}$$

Note that the ratios $C_{i,avg}/C_{j,avg}$ are fixed over time: $C_{i,avg}$ is the average computation power of node i. These values were obtained from the lb2 experiments by averaging over all the tasks processing time at each node.

Both policies were evaluated by conducting 5 runs for each value of K between 0.3 and 1 with 0.1 incremental step. Figure 1 shows the overall average completion time versus K and Figure 2 shows the total number of exchanged tasks between all the nodes.

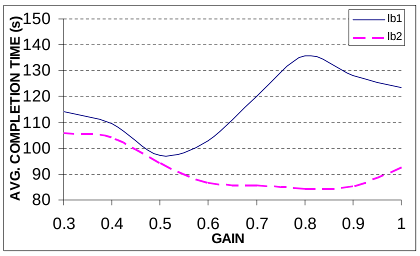

Fig. 1. completion time averaged over 5 runs versus different gain values K. The graph shows the results for both policies.

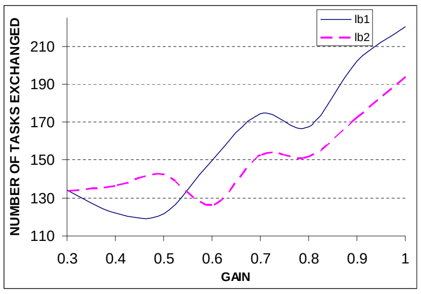

Fig. 2. Total number of tasks exchanged averaged over 5 runs versus different gain values K. The graph shows the results for both policies.

We can clearly see that lb2 outperformed lb1 especially for $K = 0.8$ that corresponds to lb2 earliest completion time whereas lb1 performed best at $K = 0.5$. The reason may be that lb2 used greater predictive computations before distributing the loads, which makes it “more or less” independent of the gain value K (in the range [0.6 0.9]). On the other hand, lb1 only uses static statistical information about the nodes' computational capability. As for the network traffic generated during the lifetime of the system, lb2 had fewer tasks exchanged for most of the gain K values. It is expected that the difference in total tasks migrated between the 2 policies will grow as the number of nodes increases.

5. CONCLUSION AND FUTURE WORK

In this paper, a dynamic load-balancing policy that takes into account the connectivity between the nodes, the variability in computational power and the network transfer delays was introduced.

Preliminary experimental results show that the proposed strategy, thanks to taking advantage of dynamical knowledge of the state of the nodes and network, provides improvements over policies implemented earlier.

Further investigations of this strategy are needed; more experiments should be conducted on a larger number of nodes to test its performance and more importantly its scalability. Moreover, the update methods of the adaptive parameters C and r_{ij} should be enhanced by observing the impact of the gain values α and β on the stability of the system.

REFERENCES

Abdallah, C. T., N. Alluri, J. D. Birdwell, J. Chiasson, V.Chupryna, Z. Tang, and T. Wang (2003). A linear time delay model for studying load balancing instabilities in parallel computations. *IEEE IJSS* **34**, 563–573.

Altman, E. and H. Kameda (2001). Equilibria for multiclass routing in multiagent networks. In: *40th IEEE CDC*. Orlando, FL. pp. 604–609.

Barabasi, Albert-Laszlo (2003). *Linked*. First Plume Printing.

Birdwell, J. Douglas, John Chiasson, Zhong Tang, Chaouki Abdallah, Majeed M. Hayat and Tsewei Wang (2004). *Advances in Time Delay Systems*. Chap. Dynamic Time Delay Models for Load Balancing Part I: Deterministic Models, pp. 355–370. Vol. 38 of *Lecture Notes in Computational Science and Engineering*. Springer: Berlin.

Bovy, C.J., H.T. Mertogimedjo, G. Hooghiemstra, H. Uijterwaal and P. Van Mieghem (2002). Analysis of end-to-end delay measurements in the internet. In: *Proceedings of the PAM2002*. Ft.Collins. pp. 26–33.

Chiasson, J., Z. Tang, J. Ghanem, C. T. Abdallah, J. D. Birdwell, M. M. Hayat and H. Jerez (2005). The effect of time delays on the stability of load balancing algorithms for parallel computations. In: *IEEE TCST, Submitted*.

Cybenko, George (1989). Dynamic load balancing for distributed memory multiprocessors. *IEEE JPDC* **volume 7**(2), 279–301.

Dhakal, S., B.S. Paskaleva, M. M. Hayat, E. Schamiloglu and C. T. Abdallah (2003). Dynamical discrete-time load balancing in distributed systems in the presence of time delays. In: *Proceedings of the IEEE CDC*. Vol. 5. Maui, Hawaii. pp. 5128–5134.

Dhakal, S., M. M. Hayat, J. Ghanem, C. T. Abdallah, H. Jerez, J. Chiasson and J. D. Birdwell (2004). *Advances in Communication Control Networks*. Chap. On the optimization of load balancing in distributed networks in the presence of delay, pp. 223–244. Lecture Notes in Control an Information Sciences (LCNCIS). Springer-Verlag: Berlin.

Dhakal, S., M. M. Hayat, M. Elyas, J. Ghanem and C. T. Abdallah (2005). Load balancing in distributed computing over wireless LAN: Effects of network delay. *Submitted, IEEE, WCNC*.

Dhakal, Sagar (2003). Load balancing in delay-limited distributed systems. Master's thesis. ECE. The University of New Mexico.

Georgatos, Fotis, Florian Gruber, Daniel Karrenberg, Mark Santcroos, Ana Susanj, Henk Uijterwaal and Rene Wilhelm (2001). Providing active measurements as a regular service for isps. In: *PAM2001*. Netherlands. pp. 45–56.

Ghanem, J., C. T. Abdallah, M. M. Hayat, S. Dhakal, J.D Birdwell, J. Chiasson and Z. Tang. (2004*a*). Implementation of load balancing algorithms over a local area network and the internet. 43rd IEEE, CDC. Bahamas.

Ghanem, J., S. Dhakal, C. T. Abdallah, M. M. Hayat and H. Jerez (2004*b*). On load balancing in distributed systems with large time delays: Theory and experiments. IEEE Med. conf. on control and automation. Turkey.

Ghanem, Jean (2004). Implementation of load balancing policies in distributed systems. Master's thesis. ECE. Univ. of New Mexico.

Hayat, M. M., S. Dhakal, C. T. Abdallah, J. D. Birdwell and J. Chiasson (2004). *Advances in Time Delay Systems*. Chap. Dynamic Time Delay Models for Load Balancing Part II: A Stochastic Analysis of the Effect of Delay Uncertainty, pp. 371–385. Vol. 38 of *Lecture Notes in Computational Science and Engineering*. Springer: Berlin.

Institute, Information Sciences (1981). Transmission control protocol darpa internet program protocol specification. RFC 793.

Jacobson, V. (1988). Congestion avoidance and control. In: *Proceedings of SIGCOMM '88*. Palo Alto, CA. pp. 314–329.

Lan, Z., V. E. Taylor and G. Bryan (2001). Dynamic load balancing for structured adaptive mesh refinement applications. In: *IEEE, ICPP*. Valencia, Spain. pp. 571–579.

Newman, M. E. J. (2003). The structure and function of complex networks. In: *SIAM Review*. Vol. 45. pp. 167–256.

Saletore, Vikram A. (1990). A distributed and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks. In: *IEEE DMCC*. Charleston, SC. pp. 994–999.

Watts, D. and S. Strogatz (1998). Collective dynamics of 'small-world' networks. In: *Nature*. Vol. 393. pp. 440–442.

Willebeek-LeMair, Marc H. and Anthony P. Reeves (1993). Strategies for dynamic load balancing on highly parallel computers. *IEEE TPDS* **volume 4**, pages 979–993.