A Dynamic Distributed Load Balancing Algorithm with Provable Good Performance *

Reinhard Lüling Burkhard Monien

Department of Mathematics and Computer Science
University of Paderborn, Germany
e-mail : rl@uni-paderborn.de, bm@uni-paderborn.de

Abstract

The overall efficiency of parallel algorithms is most decisively effected by the strategy applied for the mapping of workload. Strategies for balancing dynamically generated workload on a processor network which are also useful for practical applications have intensively been investigated by simulations and by direct applications.

This paper presents the complete theoretical analysis of a dynamically distributed load balancing strategy. The algorithm is adaptive by nature and is therefore useful for a broad range of applications. A similar algorithmic principle has already been implemented for a number of applications in the areas of combinatorial optimization, parallel programming languages and graphical animation. The algorithm performed convincingly for all these applications.

In our analysis we will prove that the expected number of packets on each processor varies only by a constant factor compared with that on any other processor, independent of the generation and consumption of workload on each processor. We give exact bounds for these values and prove an exact upper bound, independent of the number of processors. Thus, the algorithm achieves a well-balanced workload distribution on any network for any underlying application. We also prove that the variation of the expected number of packets on a processor is very small and only dependent on the parameters of the algorithm. Furthermore, we present some analysis of the costs of our algorithm. We will also show that all tradeoffs between balancing quality, variation and costs can be determined by the parameters of the algorithm.

*This work was partly supported by the German Department of Science and Technology (BMFT), PARAWAN project 413-5839-ITR 9007 BO and the DFG Forschergruppe "Effiziente Nutzung massiv paralleler Systeme"
This paper has been published in Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA-93)

1 Introduction

To use a parallel computer, the workload of the computation has to be partitioned and mapped onto the processors. Since the overall efficiency of most practical applications on parallel computers is highly dependent on that of the workload mapping, these algorithms have intensively been studied in the recent past. In general, they are tailored to the characteristics of the workload to achieve highest performance. One can distinguish between different workload classes according to the time the work units are generated (dynamically during runtime or statically before executing the algorithm).

We are interested in the distributed load balancing problem where load packets are generated and consumed dynamically during runtime in an unpredictable way. In practice, this problem has two possible variations : For some applications it is sufficient to balance the workload in a way that every processor has some load at any time. For others, it is necessary that all processors have nearly the same load at any time. In general, we consider the second type of applications, but it will be shown that the balancing quality of our algorithm is scalable, thus making also an application to problems of the former type possible.

To study dynamic load balancing algorithms, we begin with listing those principles, found to be of vital importance for their efficiency, effectivity and practicability [9]: First, it should be fully distributed, since centralized strategies are not scalable and therefore lead to less performance on larger networks. As the balancing quality normally effects the number of balancing steps (migration activity), this quality should be scalable, because different applications might require different degrees of balancing qualities. Another important characteristic of efficient load balancing algorithms is their adaptability to changing load situations, which means that any static activity bounds should be avoided, because they have to be adjusted to the specific application. A natural requirement is the minimization of migration activities and of communications for the organization of the algorithm. Normally, there is a tradeoff between this figure and the balancing quality.

Dynamically distributed load balancing algorithms have been intensively investigated by direct applications or by simulations [6, 7, 8, 9, 15, 21]. Only one theoretical result for a fully dynamic load balancing algorithm was published in [20]. The proof presented in this paper, stating that the expected load on each processor varies only by a constant from the average load, makes some incorrect assumptions [10] but the general idea can be proven to be correct if some simple modifications are made.

The dynamic generation and consumption of workload has also been considered for the so-called "dynamic tree embedding" problem, which maps a dynamically growing tree onto a fixed interconnection network [5, 19]. The practicability of this method is limited by the fact that newly generated nodes of the tree are spanned on randomly chosen neighboring processors making large amounts of communications necessary. The classical mapping problem considers the case that the graph to be mapped onto the processor network is known before executing the algorithm. The aim is to minimize edge dilation, processor load differences and edge congestion. A huge number of results have been achieved during the last years in this area. For an overview see [14].

The problem of on-line load balancing and on-line scheduling is in some way related to the problem studied in this paper. However, this problem has usually been studied for closely coupled systems since most of the presented strategies make extensive use of accessing remote memory, a method which is not useful for distributed memory parallel computing systems. The gained results are mostly based on competitive ratio analysis [1, 2, 18].

Other results were found for the so-called packet distribution problem [12, 16, 17], which does not consider the dynamic generation and consumption of workload and has therefore only little importance for practical applications.

In this paper we will present the complete analysis of a dynamically distributed load balancing algorithm, which is based on a method described in [7], fulfills the requirements listed above and has successfully been used for a number of different applications, like best first branch and bound [8], implementing concurrent prolog [4] and distributed graphic algorithms [11]. It is based on direct load balancing between "neighbored" processors and achieves very good performance even on networks containing up to 1024 processors. If the number of load packets generated by a processor decreases or increases by a factor $f > 1$, a new load balancing phase of this processor and a random chosen neighborhood, containing δ processors, is initiated, equalizing their number of load packets. To allow an analysis of this basic principle, we slightly modify the general strategy. We will show that for a large number of parameters f and δ this algorithm can balance the workload in such a way that the difference between the expected load of any two processors is bounded by a constant independent of the network size and the generation and consumption of workload, thus always achieving a good workload balance. This shows that a purely local load balancing algorithm, working independent of network size and only initiated by local load changes, can balance the workload evenly throughout any network. To analyze the load differences between any two processors, a technique based on the simple model of only one processor able to generate new load packets (one-processor-producer-model) is used. This will be extended to the one-processor-producer-consumer-model and finally to the generation and consumption of load packets by any processor. This technique can also be used for the analysis of other algorithms.

Since the factor between the expected load of any two processors is important for the practicability of an algorithm, we will present the exact values and show that they can be scaled by the parameters of the load balancing algorithm allowing any balancing quality.

The quality of the expected values is proven by analyzing their variation. It is found that variation is small in general and can also be scaled by the load balancing parameters. Some bounds of the expected costs end the analysis.

It is found that all values (difference between expected load of any two processors, variation of expected loads and costs) can generally be scaled by the parameters of the load balancing algorithm. Since all results are also independent of the actual behavior of the underlying application (load pattern), this algorithm provides a general tool for practical dynamic load balancing on parallel machines.

In chapter two we will define our model in detail. Chapter three presents the analysis of the one-processor-producer-consumer-model, followed by that of the complete model. Chapter five focuses on the variation and chapter six gives some performance figures of the algorithm. In chapter seven we describe some experiments which have been done for different parameters of the load balancing algorithm.

2 Basic Model

In one time step a processor can generate a new load packet, consume a locally available packet or do nothing. We can prove the same results if the processors are allowed to generate/consume up to a constant number of packets per time step as used in [20], since this can be modeled as a consecutive generation/consumption of one load unit. We make no assumptions about the distribution of generating and consuming activities, thus the results are valid for any load pattern of the underlying computation. All generated load packets, representing data or processes, have the same characteristic and can be consumed by any processor in any order. If a processor p initiates a load balancing action, it chooses a subset $M \subseteq \{1, \ldots, n\} - \{p\}$, $\mid M \mid = \delta$ of the n processors ($\delta < n$) at random and balances the workload of the $\delta + 1$ processors so that all processors have the same load (± 1) after this operation. The processors can be connected in any way. We only assume that a load balancing operation, as described above, can be performed in constant time independent of the amount of communicated data. This does not effect the practical relevance of the achieved results, since the amount of transmitted data is negligible for most state-of-the-art parallel computers using low latency wormhole routing and randomization techniques to avoid hotspots [13].

3 One-Processor-Generator-Consumer-Model

To analyze the complete algorithm, we first study the load balancing algorithm for a special load pattern: a single processor (w.l.o.g. processor 1) generates load packets and performs the load balancing algorithm. The load is distributed throughout the network, but not consumed by any processor. Thus, the overall system load is steadily increasing. We show that the factor between the expected load of processor 1 and any other processor can be bounded by a value independent of the number of processors and is very near to 1 for all practical cases. Figure 1 presents the concrete algorithm performed by processor 1.

```
$l_{old} = l_{new} := 0$
repeat
  $l_{new} := l_{new} + x,\ x \in \{1, 0\}$
  if ($l_{new} \geq f \cdot l_{old}$) then
    choose $M \subseteq \{2, \ldots, n\}$, $| M | = \delta$ at random
    balance workload of processors $M \cup \{1\}$
    $l_{new}$:=load of processor 1; $l_{old} := l_{new}$
until termination
```

Figure 1: Algorithm for one-processor-generator-model

Let $l_{i,t}$, $i \in \{1, \ldots, n\}$, $t > 0$ be the number of load packets on processor i after performing t load balancing operations. In the following we will analyze the ratio between the expected values of $l_{1,t}$ and $l_{i,t}$ for $i \in \{2, \ldots, n\}$.

Lemma 1 : Let $i \in \{2, \ldots, n\}$. If $E(l_{1,t}) = k \cdot E(l_{i,t})$ then $E(l_{1,t+1}) = \frac{(kf+\delta)(n-1)}{\delta kf+\delta(n-2)+(n-1)} \cdot E(l_{i,t+1})$, $\forall\ t$.

The proof of this lemma is based on the fact, that all processors $2, \ldots, n$ have the same expected number of load packets at any time. Lemma 1 provides the possibility of evaluating the ratio between $E(l_{1,t})$ and $E(l_{i,t})$ for any t by defining an operator $G : I\!R \rightarrow I\!R$, $G(k) = \frac{(kf+\delta)(n-1)}{\delta kf+\delta(n-2)+(n-1)}$. The behavior of the one-processor-generator-model at step t is described by $G^t(1)$ if all processors have initially the same amount of load packets. We prove an upper bound of $G^t(1)$ and its behavior as $t \rightarrow \infty$. Let $A := \frac{f - fn + \delta(n-2)+(n-1)}{2\delta f}$.

Lemma 2 : $G(k) \left(\begin{array}{c} \geq \\ = \\ \leq \end{array} \right) k \Leftrightarrow k \left(\begin{array}{c} \leq \\ = \\ \geq \end{array} \right) \sqrt{\frac{n-1}{f} + A^2} - A$

The proofs of lemma 1 and 2 are straightforward. Some further technical lemmas enable us to prove a result using Banach's well known contraction theorem for operator G. Define $FIX(n, \delta, f) := \sqrt{\frac{n-1}{f} + A^2} - A$.

Theorem 1 : Let $1 \leq f < \delta + 1$, then :
$G^t(1) \leq FIX(n, \delta, f)$, $\forall t$ and $lim_{t \rightarrow \infty} G^t(1) = FIX(n, \delta, f)$.

Theorem 1 shows that the factor between the expected load of processor 1 and that of any other processor i is bounded by $FIX(n, \delta, f)$ and that this bound is reached when $t \rightarrow \infty$. It is interesting to notice that, due to the contraction theorem, the result of theorem 1 also holds for any other starting situation than that of a completely balanced load on all processors. So the algorithm is able to escape from any initial imbalance to a nearly balanced situation.

The following theorem studies the one-processor-producer model independently of the network size n.

Theorem 2 : If $1 \leq f < \delta + 1$ and the system starts in a balanced state, then : $lim_{n \rightarrow \infty} FIX(n, \delta, f) = \frac{\delta}{\delta + 1 - f}$ and $FIX(n, \delta, f) \leq \frac{\delta}{\delta+1-f}$.

This result shows that it is possible to restrict the factor between the expected load of any two processors by using an entirely local strategy, acting independent of the actual network size n. Furthermore, the balancing quality can be scaled by the parameters of the load balancing algorithm.

The previous results can also be used for the model that a processor generates or consumes one load packet per time step. To analyze the behavior of the one-processor-producer-consumer model, we define two operators G and C. $G(k) = \frac{(kf+\delta)(n-1)}{\delta kf+\delta(n-2)+(n-1)}$ models a workload increase by a factor f, $C(k) = \frac{(k/f+\delta)(n-1)}{\delta k/f+\delta(n-2)+(n-1)}$ models the corresponding decrease. We can show that the results of Theorem 1 and 2 hold for both operators :

Lemma 3 : If $1 \leq f < \delta + 1$ and the system starts in a balanced state, then

(1) $G^t(1) \leq FIX(n, \delta, f) \leq \frac{\delta}{\delta+1-f}$ and
$lim_{t \rightarrow \infty} G^t(1) = FIX(n, \delta, f)$

(2) $C^t(1) \geq FIX(n, \delta, 1/f) \geq \frac{\delta}{\delta+1-1/f}$ and
$lim_{t \rightarrow \infty} C^t(1) = FIX(n, \delta, 1/f)$

(3) $lim_{n \rightarrow \infty} FIX(n, \delta, f) = \frac{\delta}{\delta+1-f}$ and
$lim_{n \rightarrow \infty} FIX(n, \delta, 1/f = \frac{\delta}{\delta+1-1/f}$

As all results are based on Banach's contraction theorem, allowing any starting point of the iterative process, we can achieve the same results for operator G with any other initial value of $\frac{E(l_{1,1})}{E(l_{i,1})} \leq FIX(n, \delta, f)$. For operator C the initial quotient has to be greater or equal to $FIX(n, \delta, \frac{1}{f})$. This leads to the following results for the one-processor-producer-consumer-model :

Theorem 3 : If the system starts in a balanced state and $1 \leq f < \delta + 1$, then for any processor $i \in \{2, \ldots, n\}$ $FIX(n, \delta, 1/f) \leq \frac{E(l_{1,t})}{E(l_{i,t})} \leq FIX(n, \delta, f)$ after t initiations of the load balancing operation. For a network of arbitrary size and for all $t > 0$: $\frac{\delta}{\delta+1-1/f} \leq \frac{E(l_{1,t})}{E(l_{i,t})} \leq \frac{\delta}{\delta+1-f}$.

These results show that the factor between the expected load of the load generating and any other processor in the network is very small in general and strictly depending on the parameters of the algorithm. An impression of performance of this algorithm will be given in section 7 by the simulation for different parameters f and δ.

4 n-Processor-Generator-Consumer-Model

To study the general model of n processors generating and consuming workload, we first redefine our timing model. Each processor has its own local clock performing a tick if the respective processor initiates a load balancing operation. Additionally, there is a global clock which is adjusted to the time needed to consume or generate one workload unit.

We describe the complete workload of processor i, l_i, at global time t by l_i^t. The algorithm is similar to the one described in [7] which has been used for different practical applications and is based on the following principles: [1]

In each global time step a processor can increase or decrease its workload by one packet or do nothing.

A load balancing algorithm at global time step t between processors $\pi(1), \ldots, \pi(\delta+1)$ is performed by balancing the workload of all participating processors. After this operation the number of packets on any two processors $i, j \in \{\pi(1), \ldots, \pi(\delta)\}$ differs at most by one. A difference of one load packet is possible since the load packets cannot be split. In the following we describe when this load balancing operation is performed and how the results of the previous section can be used to analyze the complete algorithm if some local modifications are made to the general strategy described in section 1.

Each processor i holds variables $d_{i,1}, \ldots, d_{i,n}$. We denote the value of $d_{i,j}$ at local time step t of processor i by $d_{i,j,t}$. This value represents the "virtual" number of packets generated by processor j (load class j) mapped onto processor i at local time t of processor i (variable $d_{i,i}$ is increased/decreased if processor i generates/consumes a load packet). Whenever $d_{i,i}$ has increased/decreased by a factor f a load balancing operation is initiated by processor i. Thus, i chooses a random set of δ processors of the network and balances the complete workload of all $\delta+1$ participating processors as described above.

If a processor k is engaged with other processors $\pi(1), \ldots,$-$\pi(\delta)$ in a load balancing operation at time step t, then after this operation $d_{\pi(1),j,t+1}, \ldots, d_{\pi(\delta),j,t+1}, d_{k,j,t+1}$, $j \in \{1, \ldots, n\}$ differ at most by one. The real load of all participating processor differ also at most by one. This means, that $\forall l_1, l_2 \in \{\pi(1), \ldots, \pi(\delta), k\} : \mid \sum_{j=1}^{n} d_{l_1,j,t} - d_{l_2,j,t} \mid \leq 1$. Such an assignment is always possible (snake like distribution of packets).

Thus, if a balancing operation is initiated by a processor, the "virtual" loads of all n load classes mapped onto the participating processors are virtually balanced. In this way a load balancing action can be regarded as $\delta+1$ independent load balancing actions initiated by the engaged processors balancing their self-generated virtual load. This makes the application of the previous results possible. To show this, consider the load classes $\{\pi(1), \ldots, \pi(\delta), k\}$ and the load classes $\{1, \ldots, n\} \setminus \{\pi(1), \ldots, \pi(\delta), k\}$.

- Let $i \in \{\pi(1), \ldots, \pi(\delta), k\}$. The initiation of the load balancing operation by processor k has the same effect on the load packets generated by i as a load balancing operation initiated by processor i. Since the load generated by processor i mapped onto i might only have changed by a factor $f' < f$ after the last balancing operation, the results of section 3 prove that the quotient between the expected value of the load generated by processor i on itself and on any other processor is bounded by $G^{t'}(1)$ if t' is the number of load balancing operations, processor i was involved in. The problem that $\{\pi(1), \ldots, \pi(\delta)\} \setminus \{i\}$ are not randomly choosen candidates of i is avoided if load class i on the participating processors is only balanced between processors $\{\pi(1), \ldots, \pi(\delta)\} \setminus \{i\}$.
- The virtual load generated by processor $i \in \{1, \ldots, n\} - \{\pi(1), \ldots, \pi(\delta), k\}$ on the participating processors $\pi(1),$-$\ldots, \pi(\delta), k$ are also virtually balanced (by adjusting the variables representing these values to the average). Since all these variables have the same expected value (see lemma 1), this operation does not change this expected values. Therefore, the balancing operation is invariant for these load classes.

If a processor i has no self-generated load packets ($d_{i,i} = 0$) but $l_i > 0$, it is allowed to consume up to C load packets, which are "borrowed" from other virtual load classes. Thus, processor i holds variables $b_{i,j}, j \in \{1, \ldots, n\}$ indicating the number of load packets borrowed by processor i from the virtual load class j. These values are handled as the variables $d_{i,j}$ at each load balancing operation. Processor i is allowed to borrow at most one load packet from any load class. If C load packets have been borrowed in total or one from all available load classes, at least one of the borrowed load packets generated by a processor j must be replaced by a normal load packet from processor j or has to be consumed by processor j. We distinguish two cases:

If $d_{j,j} > 0$, the packet which has been borrowed from load class j on processor i is exchanged with a real load packet of load class j on processor j. Afterwards, processor j simulates a load decrease of one packet perhaps making a load balancing operation necessary. We can increase the effectivity of this method if $x \leq min\{d_{j,j}, \sum_{k=1}^{n} b_{i,k}\}$ borrowed packets of processor i are exchanged with load packets of load class j on processor j. We chose x in a way that only one load balancing operation of processor j is necessary to delete the borrowed load packets. Thus, we maximize the number of load packets borrowed from load class j which are exchanged by real packets, and therefore minimize the total number of borrowed packets mapped onto processor i after this operation.

In case of $d_{j,j} = 0$, a load balancing operation is performed on load class j. To do this, processor j chooses δ processors $\pi(1), \ldots, \pi(\delta)$ at random. We first assume that $i \notin \{\pi(1), \ldots, \pi(\delta)\}$ and distinguish between two cases :

- If there exists one $k \in \{\pi(1), \ldots, \pi(\delta)\}$ such that processor k has load packets of class j ($d_{k,j} > 0$) or has no borrowed packets of class j ($b_{k,j} = 0$) or $d_{i,j} > 0$, then the load class j of processors $\pi(1), \ldots, \pi(\delta)$ and i is balanced. This can be done without effecting the expected load of class j on the participating processors as described above. After this, processor j initiates a load balancing operation with processors $\pi(1), \ldots, \pi(\delta)$ for the workload class j. By performing these two operations, processor j gets a packet of load class j which can be replaced with a borrowed packet of processor i, making eventually a simulated workload decrease

[1] The detailed algorithm is listed in the appendix.

necessary, or the borrowed packet on processor i has migrated to processor j where it is also consumed.

- If for all processors $k \in \{\pi(1), \ldots, \pi(\delta), i\}$: $b_{k,j} > 0$ and $d_{k,j} = 0$, processor j performs a load balancing operation with processors $j, \pi(1), \ldots, \pi(\delta)$. Since $b_{j,j} = 0$, at least one processor $k \in \{\pi(1), \ldots, \pi(\delta)\}$ migrates its borrowed packet to j. If a load balancing operation of $\pi(1) \ldots, \pi(\delta), i$ is performed afterwards, at least one borrowed load packet of i is transferred to $\pi(1), \ldots, \pi(\delta)$.

If $i \in \{\pi(1), \ldots, \pi(\delta)\}$ a load balancing operation of the load class j on processors j and $\pi(1), \ldots, \pi(\delta)$ transfers some borrowed load packets from processor i to processor j where they are consumed or migrates some real load packets of class j to processor j where they are exchanged with the borrowed packets of processor i.

Thus, in any case processor i is allowed to borrow some new load packets after performing this algorithm or has received some of his own load packets.

As this algorithm can be modeled by that of the one-processor-generator-consumer model, performed by each processor we can use the results of section 3 to prove the following theorem :

Theorem 4 : If the system starts in a balanced state and $1 \leq f < \delta + 1$, then for any two processors $i, j \in \{1, \ldots, n\}$ at global time t :

(1) For a network containing n processors and performing the proposed load balancing algorithm :
$E(l_i^t) \leq f^2 \cdot G^{t'}(1) \cdot (E(l_j^t) + C)$ if t' is the local time of processor i at global time t.

(2) $E(l_i^t) \leq \frac{f^2 \cdot \delta}{\delta + 1 - f} \cdot (E(l_j^t) + C)$.

Proof scetch : For the proof we consider real load packets plus the borrowed load packets of a processor ($l_{i,j} := d_{i,j} + b_{i,j}$, $l_i := \sum_{j=1}^{n} l_{i,j}$). We will show that for this load model the following conditions hold : $E(l_i^t) \leq f^2 \cdot G^{t'}(1) \cdot E(l_j^t)$. Since the number of real packets ($\sum_{j=1}^{n} d_{i,j}$) on each processor i is at most C units less than the virtual load, this is sufficient.

We split the load of processor i into n classes according to the processor having generated a specific load packet. Therefore $E(l_i^t) = E(l_{i,1}^t) + \ldots + E(l_{i,i}^t) + \ldots + E(l_{i,j}^t) + \ldots + E(l_{i,n}^t)$. Since $f^2 \cdot G^{t'}(1) \geq 1$ in any case, it is sufficient to show that $E(l_{i,i}^t) + E(l_{i,j}^t) \leq f^2 \cdot G^{t'}(1) \cdot (E(l_{j,i}^t) + E(l_{j,j}^t))$.

$E(l_{i,i}^t)$ has an upper bound of $f \cdot G^{t'}(1) \cdot E(l_{j,i}^t)$ at local time t' on processor i and $E(l_{j,j}^t) \geq \frac{1}{f} \cdot C^{t''}(1) \cdot E(l_{i,j}^t)$ at local time t'' on processor j. This follows directly from the results of section 3. In addition to the results described in section 3, a multiplicative constant f ($\frac{1}{f}$ in the case of operator C) has to be added, since the results for section 3 hold only for the global time step in which a load balancing action has taken place. Since the load can increase/decrease at most by a factor f between any two load balancing steps, this factor has to be taken into account.

Now it remains to show that $E(l_{i,j}^t) \leq f^2 \cdot G^{t'}(1) \cdot E(l_{j,j}^t)$. This follows directly from the inequality $E(l_{j,j}^t) \geq \frac{1}{f} \cdot C^{t''}(1) \cdot E(l_{i,j}^t)$ and the fact that $f \cdot G^{t'}(1) \cdot C^{t''}(1) \geq 1$ which is because $C^{t''}(1)$ has a lower bound of $\frac{\delta}{\delta + 1 - \frac{1}{f}}$ and $G^{t'}(1) \geq 1$.

5 Variation

To study the balancing quality of the proposed algorithm, the expected variation of the load on a specific processor is an important measure. In fact, a large number of load balancing algorithms with very good expected behavior but very large variation can be constructed. Consider e.g. the simple algorithm that sends all its packets in each time step to a single random chosen processor. The expected load of all processors is the same, but the variation of this value is very large, indicating that the algorithm is not able to balance the load.

Since our algorithm can be modeled as n independent algorithms for the one-processor-generator-consumer-model, we study the variation of $l_{i,t}$ for a processor $i > 1$ at time t for the one-processor-generator-model,

We present a recursive scheme for the exact computation of the variation density of $l_{i,t}$ for $\delta = 1$. This scheme can also be used to compute the variation density for a relaxed algorithm in the case that $\delta > 1$.

The variation density $VD(l_{i,t})$ is defined by $\frac{\sqrt{E(l_{i,t}^2) - E(l_{i,t})^2}}{E(l_{i,t})}$. To compute $E(l_{i,t}^2)$ we first define a class of graphs to represent any computation pattern and afterwards describe parts of the algorithm based on this model.

The following example should make the definition of this computation graphs clear. Suppose processor 1 generates workload and all other processors $2, \ldots, n$ do neither consume nor generate new workload. As an example let $(2, 4, 3, 3, 4, 2, 2)$ be the 7 first load balancing candidates of processor 1. This computation can be described by the graph presented in figure 2.

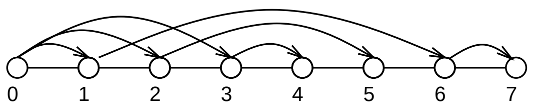


Figure 2: Computation graph, example

Informally speaking, there is a bow edge between node i and j, $i > j$, iff in time step i a processor is used as a load balancing candidate that was also used in step j and not in any step k, $j < k < i$. This edge is labeled with $\frac{1}{2}$, all so-called forward edges $(i, i+1)$ with $\frac{f}{2}$. Let v_t be the load of processor 1 at time step t. Then $v_t = \frac{1}{2} v_i + \frac{f}{2} v_{t-1}$ for some $i < t$.

For a fixed computation graph, v_t is the sum of all weighted paths from node 0 to node t. The expected value of v_t is the average over all possible graphs. In the following we will describe an algorithm for computing $E(v_{t,u}^2)$, which is

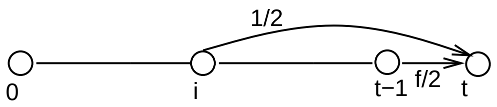


Figure 3: General structure of computation graph

the expected value of v_t^2 for all computation graphs if the computation uses exactly u processors.

Let $n(t,u)$ [2] be the number of computations of length t using exactly u processors and $n(t,u,i)$ be the number of computations of length t using exactly u processors where in step t a processor is used as a balancing candidate that has also been used in step i and not in any step $\in \{i+1,\dots,t-1\}$ (see figure 3).

To compute the average over all possible graphs, we have to consider all possible edges (i,t) $i \in \{0,\dots,t-1\}$. Therefore, the following formula holds :

$$E(v_{t,u}^2) = \frac{1}{n(t,u)} \sum_{i=0}^{t-1} n(t,u,i) E(\Big(\frac{1}{2}v_i + \frac{f}{2}v_{t-1}\Big)^2).$$

This formula can recursively be decomposed into :

1. $\frac{1}{4}E(v_{i,u_1}^2)\ \forall u_1 \le u$
2. $\frac{f}{2}E(v_{i,u_1} \cdot v_{t-1,u} \setminus \{i\})\ \forall u_1 \le u$
3. $\frac{f^2}{4}E(v_{t-1,u} \setminus \{i\}^2)$

$E(v_i \setminus M)$, $i \le t-1$, $M \subseteq \{1,\dots,t-1\}$ is defined to be the expected load value of processor 1 at time step i for those computations which have no outgoing bow edge for all nodes $x \in M$ in the corresponding computation graph.

The cases $i=0$ and $i=t-1$ can be solved directly. We consider the three terms for the case $i \in \{1,\dots,t-2\}$. The formula for the first term is described by figure 4.

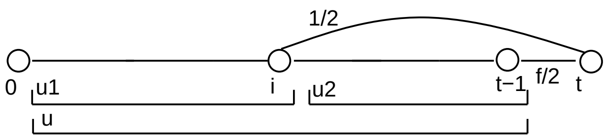


Figure 4: Computation of $E(v_{t,u}^2)$

We have to consider all possibilities of used processors in the interval $1,\dots,i$, which can be $1,\dots,u$. In the interval $i+1,\dots,t-1$ one can use any number of processors $\in \{u-u1,\dots,u-1\}$. If $u1+u2>u$, we get the additional factor $\binom{u_1-1}{u_1+u_2-u}$. This results in the following value for the first term :

$$\frac{1}{4}\sum_{i=1}^{t-2}\sum_{u1=1}^{u}\sum_{u2=u-u1}^{u-1} n_1 E(v_{i,u_1}^2)$$

and

$$n_1 = \binom{u}{u_1} n(i,u_1) n(t-i-1,u_2) \binom{u_1-1}{u_1+u_2-u}.$$

The third term can be computed as :

$$\frac{f^2}{4} n(t-1,u)(u-1)E(v_{t-1,u}^2).$$

To compute the second term, we have to consider all possible edges $(j,t-1)$, $j \in \{0,\dots,t-2\}$ and $j \ne i$. We use that $v_i \cdot v_{t-1} = \frac{1}{2}v_i \cdot v_j + \frac{f}{2}v_i \cdot v_{t-2}$ for some $0 \le j \le t-2$. This leads to 4 different cases for the term $v_i \cdot v_j$, namely $j=0$, $j=t-2$, $j<i$ and $j>i$, and 2 different cases for the term $v_i \cdot v_{t-1}$. The first two cases of the first term can be solved directly while the latter two require recursive formulas. For the case $i>j$, figure 5 explains the formula.

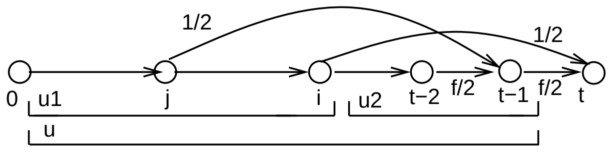


Figure 5: Computation of $E(v_{i,u_1} \cdot v_{t-1,u} \setminus \{i\})$

$$\frac{1}{2}\sum_{j=1}^{i-1}\sum_{u1=1}^{u}\sum_{u2=u-u1}^{u-2} n_2 E(v_{j,u} \cdot v_{i,u} \setminus \{j\})$$

and

$$n_2 = \binom{u}{u_1} n(i+1,u_1,j) n(t-2-i,u_2) \binom{u_1-2}{u_1+u_2-u}.$$

For $j>i$ we have a similar structured formula. The recursive formulas for the second term can be built using the same technique.

The complete algorithm to compute the variation density for a computation of length t, using exactly p processors, has complexity $O(p^2 \cdot t^3)$. This algorithm also enables us to evaluate a relaxed algorithm for the case $\delta > 1$. The relaxation is done in a way that for one load balancing step not δ candidates are chosen, but δ times one candidate to perform a load balancing operation. This algorithm is modeled by labeling the first forward-edge with $\frac{1}{\delta+1}$ and the other δ forward-edges with 1. Every bow-edge is labeled with $\frac{f}{\delta+1}$.

As can be seen from figure 6 showing the variation density for different parameters δ, f, processor numbers $\in \{2,3,\dots,$-$10,15,20,25,30,35\}$ and number of load balancing steps up to 150, the variation density is small in general and converges very quickly as the number of processors and load balancing steps increases. Thus, it can be bounded independent of the network size and number of balancing steps.

In general one can observe that there is a tradeoff between load balancing quality in terms of a low variation density and expected costs for the load balancing algorithm for different parameter sets.

[2] $n(t,u) = u^t - \sum_{j=1}^{u-1} n(t,j)\binom{u}{j}$

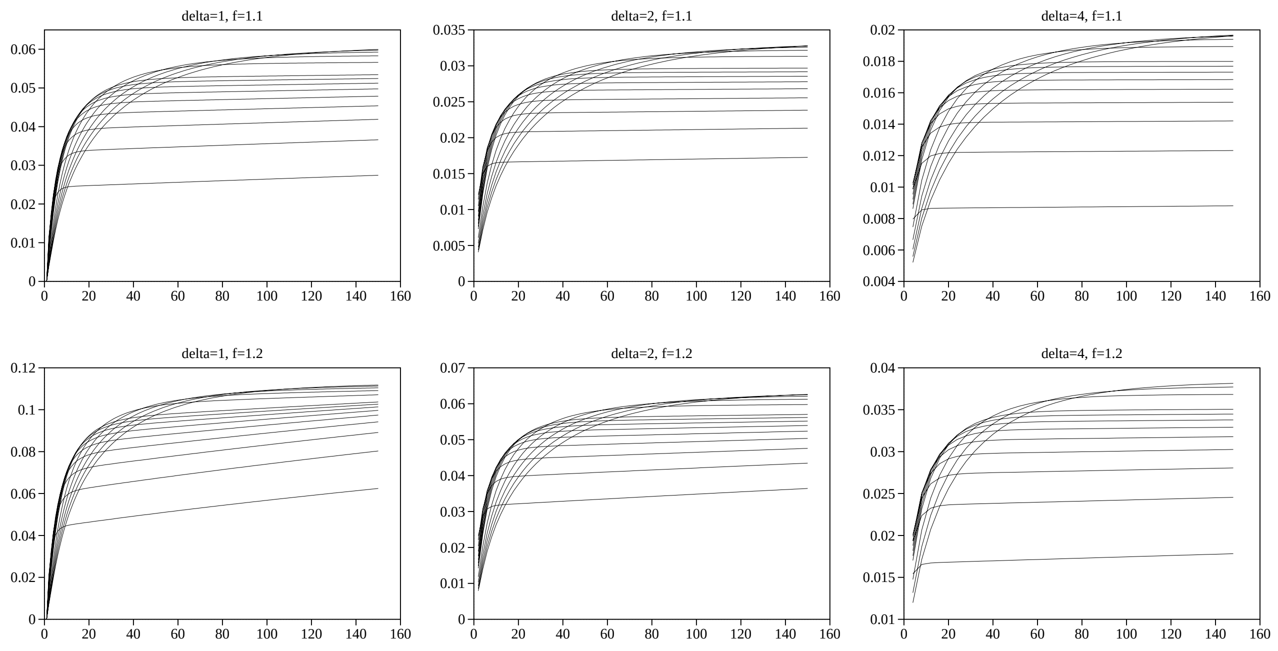


Figure 6: Variation density for $\delta \in \{1,2,4\}, f \in \{1.1, 1.2\}$

6 Complexity

The concrete costs in terms of necessary load balancing steps and number of migrated packets depend directly on the way the underlying computation generates and consumes load packets. Therefore a general analysis of the costs independent of the concrete application cannot be given for our algorithm. We present some results which describe the general tradeoff between balancing quality and costs of the algorithm. As one benchmark we consider the situation that only one processor generates load and distributes it evenly onto the network. As the costs of the complete algorithm are mainly effected by that of the one-processor-consumer model, we argue that it can serve as a benchmark situation.

As the simulation of a workload decrease is another important part of our strategy, we give bounds of the number of necessary load balancing steps to simulate a workload decrease. For the reasons described in section 2, we consider constant costs per load balancing action.

Lemma 4 : Let only processor i generate packets. After $t \geq \left\lceil log(\frac{m \cdot FIX(n,\delta,f)}{n-1+FIX(n,\delta,f))}) \cdot \frac{1}{log(\frac{f+\frac{\delta}{FIX(n,\delta,f)}}{\delta+1})} \right\rceil$ load balancing steps the expected number of workload packets generated and distributed on the network is $\geq m$.

This result demonstrates the tradeoff between the balancing quality and the costs of the algorithm

To study the effect of different parameters C, we count the number of necessary load balancing operations to decrease the number of packets from x to $x-c$ on one processor, thus, simulating a workload decrease of c packets.

Define $U := \frac{1}{f(\delta+1)} \cdot (1 + \frac{f\delta}{FIX(n,\delta,\frac{1}{f})})$ and $D := \frac{1}{f(\delta+1)} \cdot (1 + \frac{\delta f}{FIX(n,\delta,f)})$. The following lemma presents lower and upper bounds for the number of iterations to decrease the expected load units of class i on processor i from x to $x-c>0$.

Lemma 5 : Let x be the number of load units of class i assigned to processor i at a fixed time. Then the expected number t of load balancing operations to decrease this value to $x-c>0$ is bounded by $t \geq max\{0, \left\lfloor \frac{1}{logU} \cdot log(\frac{f^2(c-x)+x-1}{(f-1)(x+1)}(U-1)+1) \right\rfloor\}$ and $t \leq \left\lceil \frac{1}{logD} \cdot log(\frac{c+xf-x-f}{(x-1)f(1-\frac{1}{f})}(D-1)+1) \right\rceil$. The latter bound only holds in case that $\frac{1}{1-D} \geq \frac{c+xf-x-f}{(x-1)f(1-\frac{1}{f})}$.

The proof of lemma 5 is based on the fact that we can bound the expected load value of a processor after decreasing its load by a factor f and after performing a load balancing operation. In fact, the upper bound can be computed more precisely, since the factor between the expected load of processor i and any of its candidates is changing after performing some load decreases (application of operator C). Therefore, we can give an improved upper bound for the expected number of iterations by :

Lemma 6 : Let $D_i := \frac{1}{f(\delta+1)} \cdot (1 + \frac{\delta f}{C^i(FIX(n,\delta,f))})$. The expected number of load balancing operations to decrease the number of load packets of class i on processor i from x to $x-c>0$ is bounded by $\lceil t \rceil$ such that $\sum_{i=0}^{t-2} \prod_{j=0}^{i} D_j = \frac{c-1}{(x-1)f(1-\frac{1}{f})}$.

We simulated the algorithm and measured the number of iterations to reduce the load by a constant factor c and compared it with the lower and the two upper bounds. We observed that the bounds are very close to reality and that the number of load balancing operations is very small for typical applications. We could also conclude that the expected number of iterations is nearly independent of the parameter δ and the number of processors n, but that it is very sensitive to the parameter f of the load balancing algorithm. Thus, we also have a tradeoff between the load balancing quality, which is better for smaller values of f, and the costs of the algorithm, which is higher for low f-values. Furthermore, the number of iterations is nearly independent of the actual x-value. The same results can be achieved for any other x and c if $\frac{c}{x}$ remains constant.

7 Experiments

To study the behavior of the presented load balancing algorithm, we present some simulation results for the general algorithm on synthetic workload patterns. The aim of this work is to observe the presented tradeoffs for different applications.

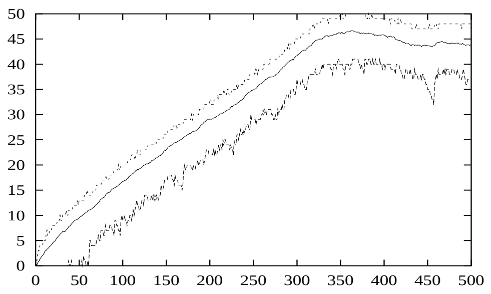
The workload of a processor is described by $(g_i, c_i, start_i,$-$end_i)$, $i = 1, \ldots, m$. At any time step t, $start_i \le t \le end_i$ the processor generates a workpacket with probability g_i and consumes a load packet if available with probability c_i. Thus one tuple describes a short phase of the algorithm. The overall behavior of the distributed algorithm is described by parameters $g_l, g_h, c_l, c_h, len_l, len_h$. The workload describing parameters g_i, c_i, $start_i$ and end_i are randomly chosen such that $g_l \le g_i \le g_h$, $c_l \le c_i \le c_h$ and $len_l \le end_i - start_i \le len_h$.

In the following we will present experiments for 64 processors. The simulation is performed for 500 time steps using the paramters : $g_l = 0.1, g_h = 0.9, c_l = 0.1, c_h = 0.7, len_l = 150, len_h = 400$. Thus, workload generation and consumption have nearly the same probability. The large parameters len_l and len_h result in a very inhomogeneous distribution of generation and consumption activities on the different processors.

As load balancing candidates and load classes for borrowing load are chosen at random, every experiment was performed 100 times. We measured the average load of a processor as well as the minimal and maximal load of a processor which ever occurred during these 100 runs of the algorithm.

Figures 7 and 8 present the behavior of the load balancing algorithm for time step 1 to 500 and parameters $f \in \{1.1, 1.8\}$ and $\delta \in \{1, 4\}$. One can observe the expected benefit from low parameters f and high parameters δ in terms of load balancing quality. In all cases the maximal derivations from the expected value are low. As these are the maximal and minimal load values over all experiments, the expected variation from the expected load value is low for all the presented experiments as already found out in chapter 5.

To study the load balancing quality in more detail, the figures 9 and 10 present the expected load, the minimal and maximal load which occurred during all runs of all 64 processors at time step 50, 200 and 400. The figures show the large impact of parameter δ on the balancing quality, whereas the parameter f plays only a minor role, if δ is already large.



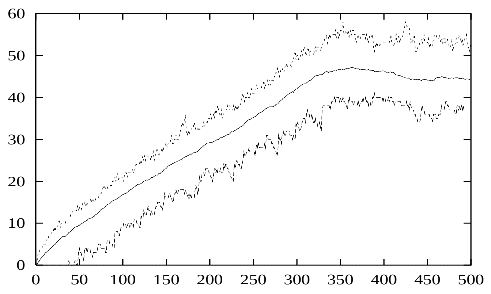
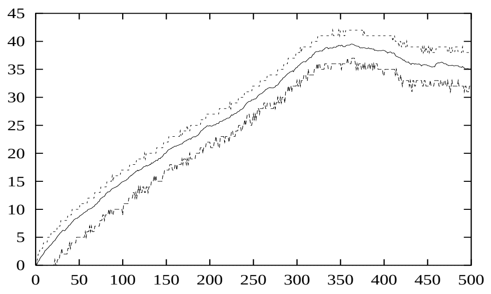


Figure 7: Balancing quality, $\delta = 1, f \in \{1.1, 1.8\}$



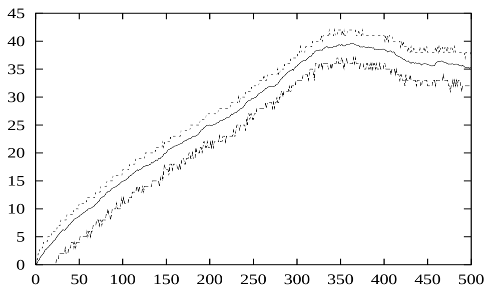


Figure 8: Balancing quality, $\delta = 4, f \in \{1.1, 1.8\}$

The experiments described so far were performed using $C = 4$. A larger parameter C increases the load imbalance as stated in theorem 4, but on the other hand decreases the

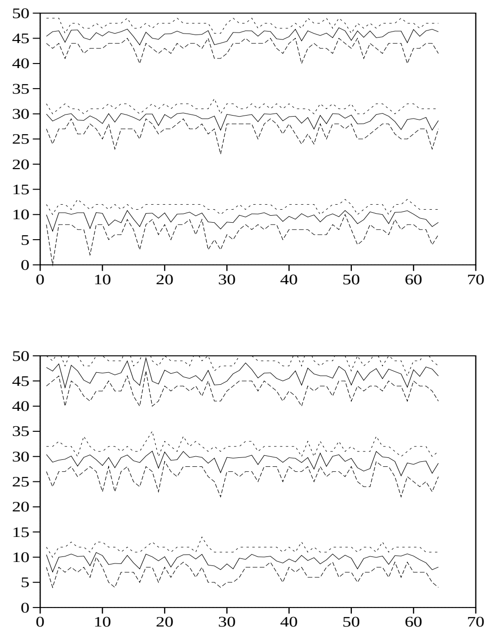


Figure 9: Distribution, $\delta = 1, f \in \{1.1, 1.8\}$

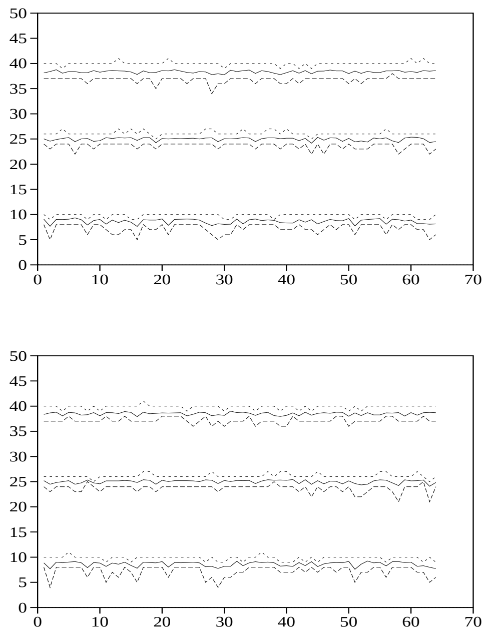


Figure 10: Distribution, $\delta = 4, f \in \{1.1, 1.8\}$

	$C = 4$	$C = 8$	$C = 16$	$C = 32$
total borrow	107.777	109.451	109.661	109.616
remote borrow	3.949	0.333	0.033	0.032
borrow fail	0.298	0.019	0.016	0.019
decrease sim	3.838	1.899	1.609	1.637

Table 1: Variations of parameter C

number of operations to borrow load from remote processors. We describe the results of some experiments to find out to what extend this value could be reduced for the benchmark load pattern presented above.

Table 1 presents the number of initiated borrowing operations (total borrow), the number of operations where a load packet of another processor is exchanged with a former borrowed packet (remote borrow), the number of initiations of the algorithm presented in chapter 4 to remove one borrowed packet (borrow fail) and the number of initiated simulations of load decrease to consume borrowed load packets (decrease sim). All this measurements were done for $f = 1.1$ and $\delta = 1$.

As one can observe the amount of communication introduced by the borrowing of load packets is very low in general and up to some extend strictly depending on the parameter C.

8 Conclusion

We analyzed the performance of a distributed dynamical load balancing algorithm and proved that this algorithm balances the work packets ensuring that the factor between the expected loads of any two processors is bounded by a constant independent of the way of workload generation and of the network size. This balancing quality can be scaled in several ways by the parameters of the algorithm and is inversely proportional to the costs of the load balancing algorithm.

We also analyzed the variation of the expected load of a processor. It was found that it is very small in general and can also be scaled by the parameters of the algorithm. The practical relevance of this work has been shown by the application of this algorithmic principle to various problems, achieving good results even on very large networks.

Further research will be done by applying the presented methods to other load balancing algorithms. We will also investigate the performance of dynamic load balancing algorithms taking locality issues on specific networks into account.

Acknowledgements:
Special thanks belong to Peter Mysliwietz and Thomas Lauer for many useful discussions.

References

[1] Y. Azar, A. Karlin, A. Broder, *On-line load balancing*, Proc. of ACM Symposium of Foundations of Computer Science (FOCS), 1992, pp. 218-225

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, *On-Line Load Balancing with Applications to Machine Scheduling and Virtual Circuit Routing*, Proc. of ACM Symposium on Theory of Computing, (STOC), 1993

[3] S.N. Bhatt, J. Cai, *Take a walk, grow a tree*, Proc of ACM Symposium of Foundations of Computer Science (FOCS), 1988, pp. 469-478

[4] U. Glässer, M. Kaercher, G. Lehrenfeld, *A Distributed Implementation of Flat Concurrent Prolog,* 1st Int. Conf. of the Austrian Center for Parallel Computing, Salzburg, 1991

[5] T. Leighton, M. Newman, A. Ranade, E. Schwabe, Dynamic Tree Embedding in Butterflies and Hypercubes, Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1989, pp. 224-234

[6] F. C. H. Lin, R. M. Keller, *The Gradient Model Load Balancing Method*, IEEE Transactions on Software Engineering, Vol. 13, No. 1 January 1987

[7] R. Lüling, B. Monien, *Load Balancing for Distributed Branch & Bound Algorithms*, Proc of Int. Parallel Processing Symposium (IPPS), 1992, pp. 543-549

[8] R. Lüling, B. Monien, M. Räcke, S. Tschöke, *Efficient Parallelization of a Branch & Bound Algorithm for the Symmetric Traveling Salesman Problem*, European Workshop on Parallel Computing (EWPC), 1992, Barcelona

[9] R. Lüling, B. Monien, F. Ramme, *Load Balancing in Large Networks : A Comparative Study* Proc. of IEEE Symposium on Parallel and Distributed Processing, 1991, pp. 686-689

[10] K. Mehlhorn, *A Counterexample for the Load Balancing Algorithm of Rudolph et.al.*, personal communication

[11] K. Menzel, M. Ohlemeyer, *Walking-Through Animation in Three-Dimensional Scenes on Massive Parallel Systems*, to appear in Visual Computer, Int. Journal of Computer Graphics, Springer Verlag, 1993

[12] F. Meyer auf der Heide, B. Oesterdickhoff, R. Wanka, *Strongly Adaptive Token Distribution*, to appear in Proc. of Int Conf. on Algorithms Languages and Programming (ICALP), 1993

[13] B. Monien, R. Lüling, F. Langhammer, *A Realizable Efficient Parallel Architecture*, Proc. of 1st Int. Heinz Nixdorf Symposium : Parallel Architectures and their Efficient Use, Paderborn, 1992, Lecture Notes in Computer Science, Springer Verlag

[14] B. Monien, I.H. Sudborough, *Embedding One Interconnection Network in Another*, Computing Suppl. 7, 1990, pp. 257-282

[15] L. M. Ni, C. W. Xu, T. B. Gendreau, *Drafting Algorithm - A Dynamic Process Migration Protocoll for Distributed Systems*, Proc. of Int. Conf. on Distributed Computing Systems, 1985, pp 539-546

[16] C. G. Plaxton *Load Balancing, Selection and Sorting on the Hypercube*, Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1989, pp. 64-73

[17] D. Peleg, E. Upfal, *The Token Distribution Problem*, SIAM Journal of Computing, vol. 18, no. 2, April 1989, pp. 229-243

[18] S. Phillips, J. Westbrook, *Online Load Balancing and Network Flow*, Proc. of ACM Symposium on Theory of Computing, 1993

[19] A. Ranade, *Optimal Speedup for Backtrack Search on a Butterfly Network*, Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1991, pp. 40-48

[20] L. Rudolph, M. Slivkin-Allalouf, E. Upfal, *A Simple Load Balancing Scheme for Task Allocation in Parallel Machines*, Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA), 1991, pp. 237-245

[21] J. A. Stankovic, I. S. Sidhu, *An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups*, Proc. of Int. Conf. on Distributed Computing Systems, 1984, pp 49-59

Appendix

Each processor $i \in \{1, \ldots, n\}$ holds the following variables :

l_i	total number of load packets on processor i
$l_{i,old}$	number of load packets generated by i after the last load balancing operation
$d_{i,1}, \ldots, d_{i,n}$	number of virtual load packets generated by processor $1, \ldots, n$ on processor i
$b_{i,1}, \ldots, b_{i,n}$	number of load packets which have been borrowed from the load class $1, \ldots, n$

Processor $i \in \{1, \ldots, n\}$ performs the following algorithm:

```
$l_i = l_{i,old} := 0$
$d_{i,j} := 0,\ b_{i,j} := 0\ \forall j \in \{1, \ldots, n\}$
**repeat**
  choose $x \in \{0, 1, -1\}$ such that $l_i + x \geq 0$
  **if** $x = 1$ **then**
    **if** $\exists b_{i,j} > 0$ **then** $b_{i,j} := b_{i,j} - 1,\ d_{i,j} := d_{i,j} + 1$
      **else** $d_{i,i} := d_{i,i} + 1$
    $l_i := l_i + 1,$
  **if** $x = -1$ and $d_{i,i} \geq 1$ **then** $d_{i,i} := d_{i,i} - 1,\ l_i := l_i - 1$
  **if** $x = -1$ and $d_{i,i} = 0$ **then**
    **if** $\sum_{j=1}^{n} b_{i,j} < C$ and $\exists d_{i,j} > 0,\ b_{i,j} = 0$ **then**
      choose $j,\ d_{i,j} > 0,\ b_{i,j} = 0$ at random
      $b_{i,j} := b_{i,j} + 1,\ d_{i,j} := d_{i,j} - 1\ j$
    **if** $\sum_{j=1}^{n} b_{i,j} = C$ or $\forall j(d_{i,j} = 0$ or $b_{i,j} = 1)$ **then**
      choose $j,\ b_{i,j} > 0$ at random
      **if** $d_{j,j} > 0$ **then**
        $x := min\{d_{j,j}, \sum_{k=1}^{n} b_{i,k}\}$
        exchange maximal $x$ borrowed packets with packets
          of $j$ such that at most one load balancing step
          on $j$ is initiated
        simulate a workload decrease of $x$ packets on $j$
      **if** $d_{j,j} = 0$ **then**
        perform algorithm to reduce $b_{i,j}$ (see section 4)
      choose $k$ with $d_{i,k} > 0,\ b_{i,k} = 0$
        $b_{i,k} := b_{i,k} + 1,\ d_{i,k} := d_{i,k} - 1$
    $l_i := l_i - 1$
  **if** $(d_{i,i} \geq l_{i,old} \cdot f)$ or $(d_{i,i} \leq l_{i,old} \cdot \frac{1}{f})$ **then**
    choose $M \subseteq \{1, \ldots, n\} - \{i\},\ \mid M \mid = \delta$ at random
    $lb_{set} := M \cup \{i\}$
    distribute $\sum_{j \in lb_{set}} l_j$ packets evenly on all $p \in lb_{set}$
    assign $d_{k,j}$ and $b_{k,j}$ such that:
      $\mid d_{l_1,j} - d_{l_2,j} \mid \leq 1,\ \mid b_{l_1,j} - b_{l_2,j} \mid \leq 1\ \forall l_1, l_2 \in lb_{set}$
      $\mid \sum_{j=1}^{n} d_{l_1,j} - \sum_{j=1}^{n} d_{l_2,j} \mid \leq 1,\ \forall l_1, l_2 \in lb_{set}$
      $\mid \sum_{j=1}^{n} b_{l_1,j} - \sum_{j=1}^{n} b_{l_2,j} \mid \leq 1,\ \forall l_1, l_2 \in lb_{set}$
      $\sum_{k=1}^{n} d_{l_1,k} = l_j,\ \forall l_1 \in lb_{set}$
    $l_{i,old} := d_{i,i}$
    **if** $b_{i,i} > 0$ **then** simulate a load decrease of $b_{i,i}$ on $i$
**until** termination
```