An analytical comparison of nearest neighbor algorithms for load balancing in parallel computers

Chengzhong Xu

Department of Electrical and Computer Engg.
Wayne State University, MI 48202

Burkhard Monien, Reinhard Lüling

Department of Computer Science
University of Paderborn, Germany

Francis C. M. Lau

Department of Computer Science, The University of Hong Kong, Hong Kong

Abstract

With nearest neighbor load balancing algorithms, a processor makes balancing decisions based on its local information and manages workload migrations within its neighborhood. This paper compares a couple of fairly well-known nearest neighbor algorithms, **the dimension exchange** *and the* **diffusion** *methods and their variants in terms of their performances in both one-port and all-port communication architectures. It turns out that the dimension exchange method outperforms the diffusion method in the one-port communication model, and that the strength of the diffusion method is in asynchronous implementations in the all-port communication model. The underlying communication networks considered assume the most popular topologies, the mesh and the torus and their special cases: the hypercube and the k-ary n-cube.*

1 Introduction

Massively parallel computers have been shown to be very efficient at solving problems that can be partitioned into tasks with static computation and communication patterns. However, there exist a large class of problems that have unpredictable computational requirements and/or irregular communication patterns. To solve this kind of problems efficiently in parallel computers, it is necessary to perform load balancing operations during run time.

Nearest neighbor load balancing algorithms are a class of methods in which processors make decisions based on local information in a decentralized manner and manage workload migrations within their neighborhood [1, 2]. Since they have a less stringent requirement on the spread of local workload around the system, they are scalable to support massively parallel computers and suitable for retaining the communication locality inherent in the underlying computations. They are also iterative in nature in the sense that successively imposing local load balancing makes progress towards a global balanced state, and hence flexible in controlling the balance quality over the spectrum from the objective of load sharing that assures no idle processors coexist together with busy processors to the degree of global balanced state.

Nearest neighbor load balancing algorithms rely on successive approximation to a global uniform distribution, and hence at each operation, need only be concerned with the direction of workload migration and the issue of how to apportion excess workloads. There are a number of ways for the choice of the direction of workload migration. Among of them, we are interested in a couple of simple representatives, *the diffusion* (DF, for short) and *the dimension exchange* (DE for short) methods. With the diffusion method, a highly or lightly loaded processor balances its workload with all of its nearest neighbors simultaneously in a load balancing operation [3, 4]. With the the dimension exchange method, by contrast, a processor in need of load balancing balances its workload successively with its neighbors one at a time and its new workload index will be considered in the the subsequent pairwise balancing [3, 5, 6]. They are closely related because they lend themselves particularly well to implementation in two basic communication architectures, the *all-port* and the *one-port* models, respectively. The all-port model allows a processor to exchange messages with all its direct neighbors simultaneously in a communication step, while the one-port model restricts a processor to exchange messages with at most one direct neighbor at a time. Both of these two models are valid in real parallel computers

and were assumed in many recent researches on communication algorithms ([7], for example).

The all-port and one-port models favor the diffusion and the dimension exchange methods, respectively. In a system that supports d-port concurrent communications, a load balancing operation using the diffusion method can be completed in one communication step while that using the dimension exchange method would take d steps. It appears that the diffusion method has an advantage over the dimension method as far as exploiting the communication bandwidth is concerned. A natural but interesting question is whether the advantage translates into real performance benefits in load balancing or not. The performance of a load balancing algorithm is determined by two measures. One is *efficiency* which is reflected by the number of communication steps required by the algorithm to drive an initial workload distribution into a uniform distribution. This measure alone is sufficient for those kinds of problems that need global balancing at run time. However, for the other kinds of applications that need to achieve load sharing rather than global balancing, we need another measure, the *balance quality*, to reflect the ability of the algorithm in bounding the variance of processors' workloads after performing one or more load balancing operations. The objective of this study is to answer the question concerning the performance of the diffusion and the dimension exchange methods in different communication models.

In this paper, we make a comprehensive comparison between the diffusion and the dimension exchange methods with respect to their efficiencies and balancing qualities when they are implemented in both one-port and all-port communication models, using synchronous/asynchronous invocation policies, and with static/dynamic random workload behaviors. The communication networks to be considered include the structures of n-D torus and mesh, and their special cases: the ring, the chain, the hypercube and the k-ary n-cube. We limit our scope to these structures because they are the most popular choices of topologies in commercial parallel computers [8].

Both the dimension exchange and the diffusion methods are parameterized algorithms, and their performance is largely influenced by the choice of the parameter values. We focus on two choices of the parameter value in each method: the average DE (ADE), the optimally-tuned DE (ODE), the local average DF (ADF), and the optimally-tuned DF (ODF). The optimality here is in terms of the efficiency in static synchronous implementations among various choices of the DE and the DF parameters. The average versions (ADE

and ADF) are the most original versions and are still being employed in real applications today; we therefore include them in our comparison. Our main results are that the dimension exchange method outperforms the diffusion method in the one-port communication model; in particular, the ODE algorithm is found to be best suited for synchronous implementation in the static situation; and that the dimension exchange method is most superior in synchronous load balancing even under the all-port communication model; the strength of the diffusion method is in asynchronous implementation under the all-port communication model; the ODF algorithm performs best in high dimensional networks in this case.

The rest of paper is organized as follows. Section 2 provides a framework of load balancing for our comparison of the various algorithms. Section 3 specifies load balancing algorithms in a unified form. Section 4 compares load balancing algorithms when they are implemented in asynchronous and synchronous invocation policies. Section 5 reports the results from simulations that further assess the load balancing algorithms. We conclude in Section 6 with a summary of comparative results for the DE and the DF methods.

2 A generic model of load balancing

The parallel computer we consider is composed of a finite set of homogeneous processors, which are interconnected by a direct communication network. Processors communicate through passing messages. The communication channels are assumed to be full duplex so that a pair of directly connected (nearest neighbor) processors can send/receive messages simultaneously to/from each other. In addition, we assume the sending and the receiving operation of a message in two ends of a channel take place instantaneously. We represent such a system by a simple connected graph $G = (V, E)$, where V is a set of processors labeled 1 through N, and $E \subseteq V \times V$ is a set of edges. Every edge $(i, j) \in E$ corresponds to the communication channel between processors i and j. Let $\mathcal{A}(i)$ denote the set of nearest neighbors of processor i, $d(i) = |\mathcal{A}(i)|$ be the degree of processor i, and $d(G)$ be the maximum of $d(i)$ for $1 \leq i \leq N$.

The underlying parallel computation is assumed to comprise a large number of independent processes, which are the basic units of workload. The total number of processes are assumed to be large enough so that the workload of a processor is infinitely divisible. Processes may be dynamically generated or consumed as the computation proceeds, and may also be migrated

across processors for the purpose of balancing. Correspondingly, we distinguish between two fundamental operations in a processor by their purposes: the computational operation and the balancing operation. An any time, a processor is performing a computational operation and/or balancing operation. Notice that during the execution of a balancing operation, the underlying computation can be suspended or performed concurrently. The concurrent execution of these two operations is possible when processors are capable of multiprogramming or the balancing operation is done in the background by cheap coprocessors. Since the workload of processors is either fixed or varying with time in the load balancing process, we refer to these two execution cases as *static* and *dynamic* situations, respectively.

Let t be an integer time variable, which is proportional to global real time. We quantify the workload of processor i at time t by w_i^t in terms of the number of residing processes. Let $\dot{\mathcal{I}}(t)$ denote the set of processors that are performing balancing operations at time t. Then, the change of workload of a processor at time t in the dynamic situation is modeled by the equation

$$
w_i^{t+1} = \begin{cases} w_i^t + \phi_i^{t+1} & i \notin \mathcal{I}(t) \\ f_i(w_j^t \mid j \in \mathcal{A}(i)) + \phi_i^{t+1} & i \in \mathcal{I}(t) \end{cases} \quad (1)
$$

where ϕ_i^{t+1} denotes the amounts of workload generated or finished from time t to $t + 1$, and $f_i(\cdot)$ represents a load balancing operator. $\phi_i^{t+1} = 0$ in the static situation.

This model is generic because the balancing operator $f_i(\cdot)$ and the set of processors in load balancing at any time t, $\mathcal{I}(t)$, are left undefined. The operator $f_i(\cdot)$ can be any nearest neighbor load balancing algorithms including the diffusion and the dimension exchange methods, which will be specified in the next section. The set $\mathcal{I}(t)$ is determined by invocation policies of load balancing. They are orthogonal to load balancing algorithms in that any invocation policy can be implemented in combination with any load balancing algorithm. Since a load balancing operation incurs nonnegligible overheads, different applications require different invocation policies for better tradeoff between their benefits and extra overheads. In parallel computations using domain decomposition techniques, for example, the computational requirement associated with each portion of a problem domain may change as the computation proceeds. To reduce the penalty of load imbalances, an effective way is to periodically redecompose the problem domain with the aim of achieving a global uniform distribution across processors. To this end, all processors are required to perform load balancing operations synchronously for a short period. That is,

$I(t) = \{1, 2, , \ldots, N\}$ for $t \geq t_0$, where t_0 is the instant when the whole system state satisfies certain conditions as those set in [9]. By contrast, the parallel execution of dynamic tree-structured computations usually requires only local balancing which assures no idle processors exist while there are other busy processors. Thus, each processor is allowed to invoke a load balancing operation at any time in an asynchronous manner according to its own local workload distribution. A simple policy is that once a processor's workload drops below a preset threshold, $w_{underload}$, a load balancing operation is then activated. That is, $\mathcal{I}(t) = \{i \mid w_i^t < w_{underload}\}$. More sophisticated invocation policies were discussed in [10, 2]. Figure 1 presents an illustration of these two implementation models in a system of five processors. The dots and the triangles represent the computation operation and balancing operation, respectively.

3 The dimension exchange versus the diffusion methods

This section briefly describe of the dimension exchange and the diffusion methods. Both of them are parameterized algorithms. We present several instances of these two methods based on different choices of values for their parameters.

3.1 The dimension exchange method

With the dimension exchange method, any processor which invokes a load balancing operation balances its workload with its neighbors successively. For a processor i, it works in the following way that

$$
f_i(\cdot) \equiv \begin{array}{l} \textbf{for} \quad (c = 1; c \leq d(i); c + +) \\ \quad w_i = w_i + \lambda(w_{j_c} - w_i) \end{array} \quad (2)
$$

where $j_c \in \mathcal{A}(i)$, and $0 < \lambda < 1$, called the dimension exchange parameter, is preset to determine the fraction of excess workload to be migrated between a pair of processors. The formula tells that a balancing operation in the dimension exchange method comprises $d(i)$ pairwise balancing steps for processor i. At each step, processor i balances its workload with one of its neighbors, and uses the new result for the subsequent balancing. It is because of the sequential nature in the sequence of balancing steps, a load balancing operation requires $d(i)$ communication steps in both the all-port and the one-port communication models.

The efficiency of the DE method is determined by the dimension exchange parameter. A DE operation with different choices of the parameter will reduce the

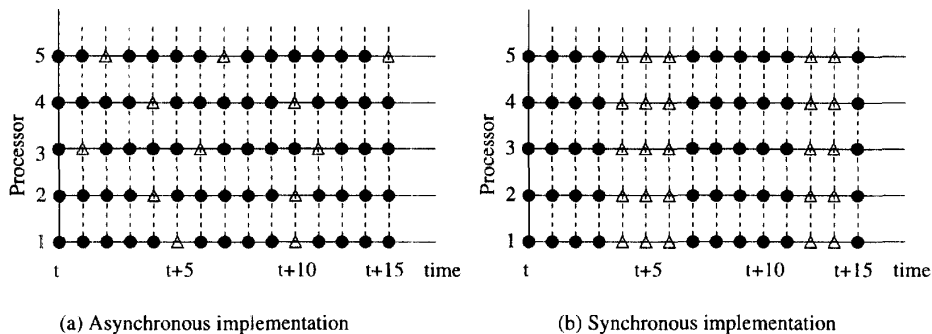| (a) Asynchronous implementation | (b) Synchronous implementation |

Figure 1: An illustration of generic models of load balancing

imbalance factor of a system state by different factors. In the following, we present two choices of the parameter which have been suggested as rational choices in the literature.

1. *Average dimension exchange* (ADE) equally splits the total workload of a pair of processors by the choice $\lambda = 1/2$. It is a straightforward choice for local balancing at each pairwise operation, and has been favored in hypercube networks [3].

2. *Optimally-tunned dimension exchange* (ODE) is a new variant of the DE method, which takes certain specific parameter values that have the effect of maximizing efficiency in global balancing [11]. The optimal parameter depends on the topology and the size of underlying communication network. Let $k = \max\{k_i, 1 \leq i \leq n\}$ in the $k_1 \times k_2 \cdots \times k_n$ mesh and torus. Then, their optimal parameter values were shown, in [11], as

- $\lambda = 1/(1 + \sin(\pi/k))$ in the mesh,
- $\lambda = 1/(1 + \sin(2\pi/k))$ in the torus.

3.2 The diffusion method

With the diffusion method, any processor which invokes a load balancing operation compares its workload with those of its nearest neighbors, and then gives away or takes in certain amount of workload with respect to each nearest neighbor. The diffusion operator in a processor i can be written in the form that

$$f_i(\cdot) \equiv w_i + \sum_{j \in A(i)} \alpha_{ij}(w_j - w_i) \qquad (3)$$

where $0 < \alpha_{ij} < 1$, called the diffusion parameter, is predefined to dictate the portion to be migrated between any two processors. Processor i apportions excess

workload $|w_i - w_j|$ to processor j if $w_i > w_j$, or fetches some workload from processor j otherwise. Clearly, a load balancing operation with the DF method requires only one communication step in the all-port communication model, but $d(i)$ steps in the one-port communication model.

As in the DE method, the efficiency of the DF method is determined by the diffusion parameter. Following are two common choices of the parameter.

1. *Local average diffusion* (ADF) takes an average of workload of neighboring processors by setting $\alpha_{ij} = \frac{1}{1+d(i)}$ [12, 13]. The torus is regular in that processors have the same degree. The mesh is approximately regular when it is in large size. For simplicity, we use a single value $\alpha = \frac{1}{1+d(G)}$ to cover all communication channels in the mesh and the torus.

2. *Optimally-tuned diffusion* (ODF) is a new variant of the DF method, which takes certain specific parameter values for maximizing efficiency in global balancing [3]. As in the DE method, the optimal diffusion parameter depends on the topology and the size of underlying networks. Let $k = \max\{k_1, k_2, \cdots, k_n\}$ in the $k_1 \times k_2 \times \cdots k_n$ mesh and torus. Then, their optimal choices were shown, in [3, 14], as

- $\alpha = 1/2n$ in the mesh,
- $\alpha = 1/(2n + 1 - cos(2\pi/k))$ in the torus,
- $\alpha = 1/(n + 1)$ in the n-D hypercube.

4 An analytical comparison

Assume $t = 0$ when processors invoke a synchronous or asynchronous load balancing procedure. We are concerned with subsequent workload distributions resulting

475

from different load balancing algorithms. Denote the overall workload distribution at certain time t by a vector $W^t = (w_1^t, w_2^t, \cdots, w_N^t)$. Denote its corresponding uniform distribution by a vector $\overline{W}^t = (\overline{w}^t, \overline{w}^t, \cdots, \overline{w}^t)$, where $\overline{w}^t = \sum_{i=1}^N w_i^t / N$. We define a concept of system *imbalance factor*, denoted by ν^t, as the deviation of W^t from $\overline{W}^t$. That is, $\nu^t = ||W^t - \overline{W}^t||^2 = \sum_{i=1}^N (w_i^t - \overline{w}^t)^2$. The system imbalance factor reflects the variance of processors' workloads at a given point in time.

With the system imbalance factor ν^t, we define the efficiency of a load balancing algorithm, denoted by T, as the number of load balancing steps required to reduce the imbalance factor of the initial state to a tolerable level in the static situation; and define the balance quality as the bound for imbalance factors which is to be guaranteed by the load balancing procedure in the dynamic situation. Load balancing algorithms will be compared with each other in terms of these two measures under the following assumptions. Throughout the section, $E[\cdot]$ denotes the expected value of a random variable.

Assumption 4.1 *At initial time, processors' workloads w_i^0, $1 \le i \le N$, are N independent and identically distributed (i.i.d.) random variables with expectation μ_0 and variance σ_0^2. At any time t, $t \ge 0$, processors' workload generation/consumption ratios ϕ_i^t, $1 \le i \le N$, are zero in the static situation or i.i.d. random variables with expectation μ and variance σ^2 in the dynamic situation.*

4.1 Asynchronous implementations

In an asynchronous implementation of load balancing, processors perform load balancing operations discretely based on their own local workload distributions and invocation policies. Since load balancing algorithms can be treated as orthogonal to their invocation policies, we consider the load balancing operations of processors in one time step so as to isolate their effects on the system imbalance factor from the effects of invocation policies. We focus on the static situation of load balancing in which the underlying computation in a processor is suspended while the processor is performing load balancing operations. The dynamic situation makes only a few differences to the analysis of the effects of load balancing.

Let ν^0 be the original system imbalance factor when $t = 0$, and ν^1 be the system imbalance factor when $t = 1$. Our comparison will be made between ν_{ade}^1, ν_{ode}^1, ν_{adf}^1, and ν_{odf}^1 which are resulting from various load balancing operations.

Theorem 4.1 *Suppose processors are running an asynchronous load balancing process under Assumption 4.1. Then, $E[\nu_{ade}^1] \le E[\nu_{df}^1]$ in the one-port communication model, while $E[\nu_{df}^1] \le E[\nu_{ade}^1]$ in the all-port communication model. Moreover, $E[\nu_{adf}^1] \le E[\nu_{odf}^1]$ in the chain and the ring networks but $E[\nu_{odf}^1] \le E[\nu_{adf}^1]$ in two- or higher- dimensional meshes and tori. In addition, $E[\nu_{ade}^1] \le E[\nu_{ode}^1]$ in the all-port communication model.*

The comparison is based on a lemma concerning the sample variance of a combination of random variables in a sample set. We present it without proof. It can be easily shown using fundamental statistical theories.

Lemma 4.1 *Suppose that $\zeta_1, \zeta_2, \ldots, \zeta_N$ are N i.i.d random variables with variance σ^2, and $\overline{\zeta} = \sum_{i=1}^N \zeta_i$. Then,*

1. for any k, $1 \le k \le N$,

$$E(|\sum_{i=1}^k a_i \zeta_i - \overline{\zeta}|^2) = (\sum_{1=1}^k a_i^2 - \frac{1}{N})\sigma^2, \quad (4)$$

where $0 < a_i < 1$ satisfying $\sum_{i=1}^k a_i = 1$; and the variance is minimized at $a_i = 1/k$ for a given k.

2. for any k_1 and k_2 and $1 \le k_1 \le k_2 \le N$,

$$E(|\sum_{i=1}^{k_1} a_i \zeta_i - \overline{\zeta}|^2) \ge E(|\sum_{j=1}^{k_2} b_j \zeta_j - \overline{\zeta}|^2) \quad (5)$$

where $0 < a_i < 1$ satisfying $\sum_{i=1}^{k_1} a_i = 1$ and $0 < b_j < 1$ satisfying $\sum_{j=1}^{k_2} b_j = 1$.

Proof sketch of Theorem 4.1 At certain time in an asynchronous load balancing process, there might be more than one processor which are invoking load balancing within their neighborhoods simultaneously. Let $\tilde{A}(i) = \{i\} \cup A(i)$ denote the balancing domain of an invoker processor i. The balancing domains of concurrent invokers may be overlapping or separated with each other. As a whole, those processors which are running load balancing processes are partitioned into a number of separated spheres, some of which are singular balancing domains and some of which are unions of overlapping domains. Processors in different spheres perform load balancing operations independently, while processors in the same sphere perform load balancing in a synchronous manner.

Suppose initially there are m independent balancing spheres in the system, denoted by $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_m$.

Then, by the definition of the system imbalance factor ν, we have

$$E[\nu^1] = E(\sum_{i=1}^{N} |w_i^1 - \overline{w}^1|^2) = \sum_{i=1}^{N} E(|w_i^1 - \overline{w}^1|^2)$$

$$= \sum_{j=1}^{m} \sum_{i \in B_i} E(|w_i^1 - \overline{w}^1|^2) + \sum_{i \notin \cup_{j=1}^{m} B_j} E(|w_i^1 - \overline{w}^1|^2)$$

The last term is a constant for a given number of processors in load balancing and independent of the topological relationships among the processors in load balancing. The first term is due to load balancing operations in all separated balancing spheres. It is a simple arithmetic sum of imbalance factors of each sphere, $\sum_{i \in B_j} E(|w_i^1 - \overline{w}^1|^2)$. As a whole, $E[\nu^1]$ implies that the expected value of the system imbalance factor is influenced independently by load balancing operations within different balancing spheres. Therefore, it suffices to compare the effects of load balancing algorithms within different spheres using Lemma 4.1. Owing to the limitation of space, the remainder of the proof is omitted here. ■

This theorem says that the dimension exchange and the diffusion methods are suitable for the one-port and the all-port communication models, respectively. More specifically, it reveals that the ODF algorithm outperforms the ADF algorithm in higher dimensional meshes and tori although the ODF was originally proposed for use in synchronous global balancing.

4.2 Synchronous implementations

In a synchronous implementation of load balancing, processors perform load balancing operations concurrently and continuously for a time period in order to achieve a global balanced state in the state situation or to keep the varying system imbalance factor bounded in the dynamic situation. From Eq. (2) and (3), it is known that both the balancing operators, $f_i(\cdot)$, of the DE and the DF methods are linear iterative operators. Hence, the synchronous implementation of Eq.(1) can be modeled by the equation

$$W^{t+1} = FW^t + \Phi^t, \qquad (6)$$

where F is either a DE or a DF matrix defined by Eq. (2) or (3), respectively. The features of synchronous implementations of the DE or DF methods are therefore fully captured by the iterative process governed by F.

In the static situation, $\Phi^t = 0$. According to fundamental iterative theories, we then have

$$T = O(1/\ln \gamma(F)), \qquad (7)$$

where $\gamma(F)$ is the subdominant eigenvalue of F in modulus. The closed expressions of $\gamma(F)$ are readily available in [12, 11] when the DE and the DF methods are applied in the mesh and the torus networks. Substituting them in Eq.(7), we obtain the efficiencies of the DE and the DF methods in both one-port and all-port communication models, as presented in Table 1.

The entries of the table show that both the ADE and the ODE algorithms converge asymptotically faster than the diffusion method in the one-port communication model; and that in the all-port communication model, the ODE algorithm converges also faster than other three algorithms by a factor of k.

In the dynamic situation, Eq.(7) leads to that

$$E[\nu^t] = E(||W^t - \overline{W}^t||^2)$$

$$= E(||FW^{t-1} - \overline{W}^{t-1}||^2) + E(||\Phi^t - \overline{\Phi}^t||^2)$$

$$= E(||F^{t+1}W^0 - \overline{W}^0||^2 + \sum_{i=0}^{t} E(||F^i\Phi^{t+1-i} - \overline{\Phi}^{t+1-i}||^2).$$

From Lemma 4.1, we then obtain that with the DE method in the one-port model,

$$E[\nu_{de}^t] = (sb^{td+d}\sigma_0^2 + s\frac{1 - b^{td+d}}{1 - b^d}\sigma^2)N - (t+1)d\sigma^2 - \sigma_0^2,$$

where $b = (1 - \lambda)^2 + \lambda^2$ and $s = 1 + b + b^2 + \cdots + b^{d-1}$; and with the DF method in the all-port model,

$$E[\nu_{df}^t] = (a^{t+1}\sigma_0^2 + \frac{1 - a^{t+1}}{1 - a}\sigma^2)N - (t+1)\sigma^2 - \sigma_0^2,$$

where $a = (1 - d\alpha)^2 + d\alpha^2$. Easily, we come to the following theorem.

Theorem 4.2 *Suppose processors are running synchronous DE and DF load balancing processes under Assumption 4.1. Then, $E[\nu_{ade}^t] \le E[\nu_{ode}^t]$, $E[\nu_{adf}^t] \le E[\nu_{odf}^t]$, and $E[\nu_{ade}^t] \le E[\nu_{adf}^t]$ in both one-port and all-port communication models.*

5 Experimental results

In the preceding section, we explored a number of relationships between the dimension exchange and the diffusion methods with respect to their efficiencies and balancing qualities. In order to obtain an idea of the magnitude of their differencies, we conducted a statistical simulation of these load balancing algorithms on various topologies and sizes of communication networks and on synthetic workload distributions. The experimental results also serve to verify the theoretical results.

477

Table 1: Efficiencies of load balancing algorithms in the mesh and torus networks, where k is the maximum number of nodes over all dimensions in an n-D network and $* - port$ means the all-port communication model.

	ADE		ODE		ADF		ODF	
	1-port	*-port	1-port	*-port	1-port	*-port	1-port	*-port
torus	$O(nk^2)$	$O(nk^2)$	$O(nk)$	$O(nk)$	$O(n^2k^2)$	$O(nk^2)$	$O(n^2k^2)$	$O(nk^2)$
mesh	$O(nk^2)$	$O(nk^2)$	$O(nk)$	$O(nk)$	$O(n^2k^2)$	$O(nk^2)$	$O(n^2k^2)$	$O(nk^2)$

In the simulation, the initial workload distribution W is assumed to be a random vector, each element w of which is drawn independently from an identical uniform distribution in $[0, 1000]$. Each data point obtained in the experiment is the average of 20 runs, using different random initial workload distributions and different workload generation ratios. We also assume the underlying system is in all-port communication model so that a DE balancing operation takes the time of $2n$ DF operations in the n-D mesh and the n-D torus. A DF operation is taken as a basic time step in a load balancing process.

The first experiment is a simulation of synchronous load balancing in the static behavior of workloads. In the simulation, we measure the number of communication steps, denoted by T, necessary for arriving at a global balanced state. We define the global balance state as the state in which system imbalance factor is less than or equal to one. Figure 2 plots the experimental results from different load balancing algorithms in the 2-D mesh of various sizes from 2×2 to 32×32.

change method outperforms the diffusion method even in the all-port communication model. In particular, it is seen that the ODE algorithm accelerates the DE load balancing process significantly. In Figure 2, we also see that the number of communication steps T in a 2-D mesh is dependent only on the size of its large dimension and insensitive to the size of its small dimension. This observation was proved to be true in both the mesh and the torus in [11].

The second experiment is a simulation of asynchronous load balancing in the dynamic situation of random workload generations/consumptions. In the simulation, we assume the expected workload generation ratio of a processor at each time step is 100 with the variance of 30 and the consumption ratio is a constant 100. In the simulation of asynchronous load balancing, we use a simple invocation policy that once a processor's workload drops or rises beyond a pair of preset bounds, 200 and 800, the processor then activates a load balancing operation. Figure 3 plots the system imbalance factors resulting from different load balancing algorithms in a mesh of size 16×16.

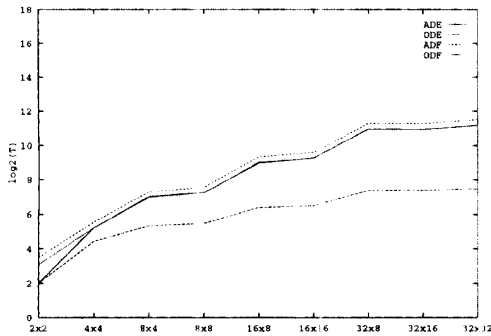
Figure 2: The number of necessary communication steps during a statically synchronous load balancing process in the 2-D mesh of various sizes from 2×2 to 32×32
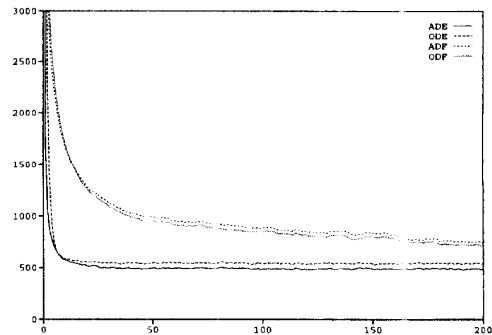

Figure 3: Change of the system imbalance factor in the first 200 steps of a dynamically asynchronous load balancing process in the mesh of size 16×16

This figure clearly indicates that the dimension ex-

From this figure, it is seen that the ADE algorithm re-

duces the initial system imbalance factor more rapidly than the diffusion method and keeps it bounded in a much lower level. It can also be observed that both the ODE and the ODF algorithms, the optimally tuned algorithms for global synchronous load balancing, do not gain significant benefits in asynchronous implementations.

6 Conclusions

In this paper, we made a comparison between two classes of nearest neighbor load balancing algorithms, the dimension exchange (DE) and the diffusion (DF) methods, with respect to their efficiency in driving any initial workload distribution to a uniform distribution and their ability in controlling the growth of variance among processors' workloads. We focused on their four instances—the ADE, the ODE, the ADF and the ODF—which are the most common versions in practice. The comparison was made comprehensively in both one-port and all-port communication models with consideration of various implementation strategies: synchronous/asynchronous invocation policies and static/dynamic random workload behaviors.

We showed that the DE method outperforms the DF method in the one-port communication model. In particular, the ODE algorithm is best suited for synchronous implementation in the static situation. We also revealed of the superiority of the DE method in synchronous load balancing even in the all-port communication model. The strength of the diffusion method is in asynchronous implementation in the all-port communication model. The ODF algorithm performs best in high dimensional networks in that case.

The comparative study not only provides an insight into nearest neighbor load balancing algorithms, but also offers practical guidelines to system developers in designing load balancing architectures for various parallel computational paradigms.

References

[1] V. Kumar, A. Y. Grama, and N. R. Vempaty. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 22(1):60–79, July 1994.

[2] M. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979–993, September 1993.

[3] G. Cybenko. Load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: Numerical methods.* Prentice-Hall Inc., 1989.

[5] B. Ghosh and S. Muthukrishnan. Dynamic load balancing in distributed networks by random matchings. In *Proceedings of 6th ACM Symposium on Parallel Algorithms and Architectures*, 1994.

[6] C.-Z. Xu and F.C.M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16(4):385–393, December 1992.

[7] S. L. Johnsson and C.-T. Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, September 1989.

[8] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26:62–76, February 1993.

[9] D. M. Nicol and P. F. Reynolds. Optimal dynamic remapping of data parallel computation. *IEEE Transactions on Computers*, 39(2):206–219, February 1990.

[10] R. Lüling and B. Monien. A dynamic distributed load balancing algorithm with provable good performance. In *Proceedings of 5th ACM Symposium on Parallel Algorithms and Architectures*, pages 164–172, 1993.

[11] C.-Z. Xu and F.C.M. Lau. The generalized dimension exchange method for load balancing in k-ary n-cubes and variants. *Journal of Parallel and Distributed Computing*, 24(1):72–85, January 1995.

[12] J. B. Boillat. Load balancing and poisson equation in a graph. *Concurrency: Practice and Experience*, 2(4):289–313, December 1990.

[13] X.-S. Qian and Q. Yang. Load balancing on generalized hypercube and mesh multiprocessors with lal. In *Proceedings of 11th International Conference on Distributed Computing Systems*, pages 402–409, 1991.

[14] C.-Z. Xu and F.C.M. Lau. Optimal parameters for load balancing with the diffusion method in mesh networks. *Parallel Processing Letters*, 4(2):139–147, June 1994.