ICOT Technical Report: TR-0708

TR-0708

Asymptotic Load Balance of Distributed Hash Tables

by

N. Ichiyoshi & K. Kimura

August, 1991

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome

(03)3456-3191～5
Telex ICOT J32964
Minato-ku Tokyo 108 Japan

Institute for New Generation Computer Technology

Asymptotic Load Balance of Distributed Hash Tables

Nobuyuki Ichiyoshi and Kouichi Kimura
Institute for New Generation Computer Technology
1 - 4 - 28 Mita, Minato-ku, Tokyo 108, Japan
{ichiyoshi,kokimura}@icot.or.jp

Abstract

The distributed hash table is a parallelization of the hash table obtained by dividing the table into subtables of equal size and allocating them to the processors. It can handle a number of search/insert operations simultaneously, increasing the throughput by up to p times that of the sequential version, where p is the number of processors. However, in the average case, the peak throughput is not attained due to load imbalance.

It is clear that the table size m must grow at least linearly in p to balance the load. In this paper, we study the rate of growth of m relative to p necessary to maintain the load balance on the average (or to make it approach the perfect load balance). It turns out that linear growth is not enough, but that moderate growth—namely $\omega(p \log^2 p)$—is sufficient. The probabilistic model we used is fairly general and can be applied to other load balancing problems.

We also discuss communication overheads, and find that, in the case of mesh multicomputers, unless the network channel bandwidth grows sufficiently as p grows, the network will eventually become a performance bottleneck for distributed hash tables.

1 Introduction

Parallel computation achieves speedup over sequential computation by sharing the computational load among processors. The load balance between processors is central in determining the parallel runtime (though other factors also affect performance). Unlike uniform computational tasks in which almost perfect load balance is achieved by allocating data uniformly to the processors, non-uniform computational tasks such as search problems pose non-trivial load balancing problems.

In most non-uniform tasks, worst-case computational complexity is far larger than average-case complexity; and the worst case is usually a very rare case. Thus, the study of average case performance is important, and it has been conducted for sorting and searching [Knuth 1973], optimization problems [Coffman and Lueker 1991], and many others [Vitter and Flajolet 1990]. However, there seems to have been little work on average-case performance analysis in regard to parallel algorithms, especially on highly-parallel computers, a notable exception being [Kruskal and Weiss 1985].

In this paper, we study the average-case load balance of distributed hash tables on highly parallel computers. A distributed hash table is a parallelization of a hash table, in which the table is divided into subtables of equal size to be allocated to the processors. It can handle a number of search/insert operations simultaneously, increasing the throughput up to p times that of the sequential version, where p is the number of processors.

However, in average cases, the peak throughput is not attained due to load imbalance. Intuitively, the more buckets allocated to each processor, the better the average load balance becomes. It is clear that under a constant load factor $\alpha = n/m$ (n is the number of elements in the table, m is the table size), m must grow at least linearly in p to balance the load. We shall investigate the necessary/sufficient rate of growth of m relative to p so that the load balance factor—the average processor load divided by the maximum processor load—approaches 1 as $p \to \infty$. It turns out that linear growth is not enough, but that moderate growth—namely, $\omega(p \log^2 p)$—is sufficient. This means that the distributed hash table is a data structure that can exploit the massive computational power of highly parallel computers, with problems of a reasonable size.

We also briefly discuss communication overheads on multicomputers, and find that, in the case of mesh multicomputers, unless the network channel bandwidth grows sufficiently as p grows, the network will eventually become a performance bottleneck for distributed hash tables.

The rest of the paper is organized as follows. Section 2 describes the distributed hash table and defines the problem we shall analyze. The terminology of average-case scalability analysis is introduced in Section 3. The analysis of load balance is presented in Section 4. The full proofs of the propositions appear in [Kimura and Ichiyoshi 1991]. The communication overheads are considered in Section 5. The last section summarizes the paper.

2 Distributed Hash Tables

2.1 Distributed Hash Tables

The distributed hash table is a parallelization of the hash table. A hash table of size $m = pq$ is divided into subtables of equal size q and the subtables are allocated to p processors. The two most simple bucket allocations are:

The block allocation

> The k-th bucket ($k \geq 1$) belongs to the ($\lfloor (k-1)/q \rfloor +$ 1)-th subtable,[1] and

The modular allocation

> The k-th bucket ($k \geq 1$) belongs to the ((($k-1$) mod p) + 1)-th processor.

At the beginning of a hash operation (search or insert) for an element x, the hash function is computed for x to generate a number h ($1 \leq h \leq m$), and the element (or the key) is dispatched to the processor which contains the h-th bucket. The rest of the operation is processed at the target processor.

For better performance, it is desirable to maximize the locality. Thus, when the indirect chaining scheme is employed for hash collision, the entire hash chain for a given bucket should be contained in the same processor which contains the bucket. With open addressing, linear probing has the best locality (under the allocation scheme (1)) but its performance degrades quickly as the load factor increases. Other open addressing schemes have better sequential performance characteristics [Knuth 1973], but have less locality. For this reason and also for simplicity of analysis, we choose the indirect chaining scheme. The bucket allocation scheme does not influence the load balance analysis in this case.

The absence of a single entry point that can become a bottleneck makes the distributed hash table a suitable data structure for highly parallel processing. The peak throughput increases linearly with the number of processors. The problem is: When does the "real" performance approach the "peak" performance? When elements are evenly distributed over the processors, linear growth in the number of data elements is sufficient for linear growth in performance. On the other hand, in the worst case, all elements in the hash table might belong to a single subtable so that performance does not increase at all. We are not interested in these two extremes, but in average performance, just as we are more interested in the average complexity of hash operations in sequential hash tables rather than worst-case complexity.

2.2 Problem Definition

There can be a number of uses of hash tables depending on the application. Here we examine the following particular use of the hash table.

Concurrent Data Generation, Search and Insertion

Initially, there is an "old" distributed hash table containing "old elements" and an empty "new" distributed hash table. The old and new tables are of the same size $m = pq$ (p is the number of processors and q is the number of buckets assigned to each processor) and use the same hash function. Also, some "seeds" of new elements are distributed randomly across the processors.

(1) Concurrent Data Generation

> Each processor generates "new elements" from the allocated seeds. It is assumed that the time it takes each processor to generate new elements is proportional to the number of generated elements.

(2) Concurrent Data Dispatch

> Each processor computes the hash values of the new elements and dispatches the elements to the target processors accordingly.

(3) Concurrent Search

> Each processor does a search in the old table for each of the new elements it has received.

(4) Concurrent Insert

> Each processor inserts those new elements that are not found in the old table into the new table. No interprocessor communication arises, because the old and new hash tables use the same hash function.

The above usage may seem a little artificial, but the probabilistic model and the analysis for it should be easily applicable to other usages. In the analysis of load balance, the data dispatch step is ignored (equivalently, instantaneous communication is assumed). This is discussed in Section 5.

3 Scalability Analysis

Average Speedup and Efficiency We denote the sequential runtime by $T(1)$ and the parallel runtime using p processors by $T(p)$. The *speedup* is defined by $S(p) = T(1)/T(p)$, and the *efficiency* by $E(p) = S(p)/p$. Efficiency is the ratio between the "real" performance (obtained for a particular problem instance) and the "peak" performance of the parallel computer. In the absence of speculative computation, the efficiency is less than or equal to 1.

[1] When p does not divide m, taking $q = \lceil m/p \rceil$ works but it may lead to a sub-optimal load balance (e.g., consider the case $m = p + 1$). A better load balance can be realized by a mapping function which is a little more complicated than simple division.

Since we intend to engage ourselves in an average-case analysis, we need to define the "average speedup" and the "average efficiency".

Definition 1 We define the *average collective speedup* $\sigma(p)$ by $E(T(1))/E(T(p))$ ($E(X)$ denotes the expectation of X) and the *average collective efficiency* $\eta(p)$ by $\sigma(p)/p$.

The reason why we analyze the above defined average collective speedup, and not the expected speedup in the literal sense—$E(T(1)/T(p))$—is that: (1) it is much simpler to analyze $E(T(1))/E(T(p))$ than analyze $E(T(1)/T(p))$, and (2) in cases where any average speedup figure is meaningful our definition is a better indicator of overall speedup. Suppose we run a number of instances $I_1, I_2, \ldots$ from some problem class, then the *collective speedup* defined by $\sum_i T(1, I_i)/\sum_i T(p, I_i)$ ($T(1, I_i)$ and $T(p, I_i)$ are sequential and parallel runtimes for problem instance I_i) and represent overall speedup. This is more meaningful than any one of arithmetical mean, geometric mean, or harmonic mean that may be calculated from the individual speedups $T(1, T_i)/T(p, T_i)$.

Scalability Analysis and Isoefficiency We would like to study the behavior of $\eta(p)$ as p becomes very large. In general, for a fixed amount of total computation W, $\eta(p)$ decreases as p increases, because there is only finite parallelism in a fixed problem. On the other hand, in many parallel programs, for a fixed p, $\eta(p)$ increases as W grows. Kumar and Rao [1987] introduced the notion of *isoefficiency*: if W needs to grow according to $f(p)$ to maintain an efficiency E, then $f(p)$ is defined to be the *isoefficiency function* for efficiency E. A rapid rate of growth in the isoefficiency function indicates that near-peak performance of a large-scale parallel computer can be attained only when very—sometimes unrealistically—large problems are run. Such a parallel algorithm and/or data structure is not suitable for utilizing a large-scale parallel computer. (We will refer to the isoefficiency by this original definition by *exact isoefficiency*.)

Since it is sometimes impossible to maintain an exact E because of the discrete nature of the problem, the following weaker definitions of isoefficiency may be more suitable or easier to handle.

Asymptotic Isoefficiency f is an *asymptotic isoefficiency* function for E if

$$\lim_{p\to\infty} \eta(p) = E \quad \text{under } W = f(p).$$

Asymptotic Super-Isoefficiency f is an *asymptotic super-isoefficiency* function for E if

$$\liminf_{p\to\infty} \eta(p) \geq E \quad \text{under } W = f(p).$$

f is an *asymptotic super-isoefficiency* function if it is an asymptotic super-isoefficiency function for some $E > 0$, i.e., the efficiency is bounded away from 0 as $p \to \infty$.

An exact isoefficiency function for E is an asymptotic isoefficiency function for E; and an asymptotic isoefficiency function for E is an asymptotic super-isoefficiency function for E.

In the analysis of load balance, we study the balance of essential computation. Essential computation is the total computation performed by processors excluding the parallelization overheads. The amount of essential computation is equal to $pT(p)$ minus the total overhead time spent on things such as message handling and idle time. In the absence of speculative computation, we can identify the amount of essential computation with the sequential runtime.[2] The terminology for load balance analysis is defined like that for speedup/efficiency analysis, except that "essential computation" replaces "runtime": the *total essential computation* corresponds to sequential runtime; *maximum processor load* corresponds to parallel runtime; and *load balance factor*[3] corresponds to efficiency. We use the same terminology for isoefficiency functions. In the following analysis, we study asymptotic isoefficiency for 1 and asymptotic super-isoefficiency. (Since we are not dealing with exact isoefficiency, we drop the adjective "asymptotic" for brevity.)

4 Analysis of Load Balance

4.1 Assumptions

For the sake of probabilistic analysis, we consider a model in which the following values are treated as random variables (RVs): the number of old and new elements belonging to the j-th bucket on the i-th processor ($1 \leq i \leq p$, $1 \leq j \leq q$) denoted by A_{ij} and B_{ij} respectively, and the number of new elements generated at the i-th processor denoted by G_i.

First, we make some assumptions on the distributions of these random variables. The two alternative models of hash tables are the Bernoulli model in which the number of elements n inserted in m buckets is fixed ($\alpha = n/m$) and the probability that an element has a given hash value is uniformly $1/m$, and the Poisson model in which the occupancy of each bucket is an independent Poisson random variable with parameter α [Vitter and Flajolet 1990]. We choose the Poisson model, because it is simpler to analyze directly, and because, with regard to the distributions of maximum

[2] If we ignore various sequential overheads such as cache miss, process switching, and paging.

[3] Not to be confused with the load factor of hash tables.

bucket occupancy in which we are interested, those under the Bernoulli model approach those under the Poisson model as $m \to \infty$ [Kolchin *et al.* 1978].

For a similar reason, we assume that G_i ($1 \le i \le p$) are independent identically distributed (i.i.d.) random variables having a Poisson distribution with some parameter γ. It follows that the total number of new elements has a Poisson distribution with parameter $p\gamma$, and by the assumption on the hash function, B_{ij}'s are i.i.d. random variables having a Poisson distribution with parameter $\beta = p\gamma/m = \gamma/q$. We assume that load factors α and β of the old and new hash tables are constant (do not change with p, q).

To summarize, A_{ij} and B_{ij} are i.i.d. random variables having a Poisson distribution with parameters α and β, and G_i are i.i.d. random variables with a Poisson distribution with parameter $q\beta$. Note that G_i's and B_k's are not independent because $\sum_i G_i = \sum_{ij} B_{ij}$.

4.2 Essential Computation and Load Balance Factor

Since each data generation is assumed to take the same time, the essential computation of the data generation step is:

$$W_{gen} = \sum_{1\le i\le p} G_i.$$

(ignoring the constant factor).

As for the search step, some searches are successful (the new element is found in the old table) and others are unsuccessful. For simplicity of analysis, we choose a pessimistic estimate of the essential computation and assume that all searches are unsuccessful. We also assume that an unsuccessful search involves comparison of the new elements against all the old elements in the bucket. Thus, the number of comparisons made by an unsuccessful search in the bucket with A_{ij} elements is $A_{ij} + 1$ (the number of elements plus one for the hash table slot containing the pointer to the collision chain). Therefore, the essential computation of the search step is:

$$W_{search} = \sum_{1\le i\le p} \sum_{1\le j\le q} (A_{ij} + 1)B_{ij}.$$

(again ignoring the constant factor).

We make a similar assumption for the insert step: every insert is done after an unsuccessful search in the new table. Thus, the essential computation of the search step for bucket j on processor i is:

$$\sum_{0\le l\le B_{ij}-1} (l+1) = B_{ij}(B_{ij}+1)/2,$$

and the total essential computation for the search step is

$$W_{insert} = \sum_{1\le i\le p} \sum_{1\le j\le q} B_{ij}(B_{ij}+1)/2.$$

Thus, the total essential computation is:

$$W(1) = \sum_{1\le i\le p} (W'_i + W''_i + W'''_i),$$

where

$$W'_i = G_i, \quad W''_i = \textstyle\sum_{1\le j\le q}(A_{ij}+1)B_{ij}, \quad \text{and} \\ W'''_i = \textstyle\sum_{1\le j\le q} B_{ij}(B_{ij}+1)/2.$$

The maximum processor load is

$$W(p) = \max_{1\le i\le p}(W'_i + W''_i + W'''_i)$$

The average load balance factor $\eta(p)$ is $E(W(1))/pE(W(p))$. We would like to know what rate of growth of q is necessary/sufficient so that $\eta(p) \to 1$ as $p \to \infty$.

Since

$$\begin{aligned} & E\left(\sum_{1\le i\le p} (W'_i + W''_i + W'''_i)\right) \\ &= E\left(\sum_{1\le i\le p} W'_i\right) + E\left(\sum_{1\le i\le p} W''_i\right) + E\left(\sum_{1\le i\le p} W'''_i\right) \\ &= p(E(W'_1) + E(W''_1) + E(W'''_1)), \end{aligned}$$

and

$$\begin{aligned} & E\left(\max_{1\le i\le p}(W'_i + W''_i + W'''_i)\right) \\ &\quad \le E\left(\max_{1\le i\le p} W'_i\right) + E\left(\max_{1\le i\le p} W''_i\right) + E\left(\max_{1\le i\le p} W'''_i\right), \end{aligned}$$

we have

$$\eta(p) \ge \frac{E(W'_1) + E(W''_1) + E(W'''_1)}{E\left(\max_{1\le i\le p} W'_i\right) + E\left(\max_{1\le i\le p} W''_i\right) + E\left(\max_{1\le i\le p} W'''_i\right)}.$$

Thus, if

$$\begin{aligned} E\left(\max_{1\le i\le p} W'_i\right) &\sim E(W'_1), \\ E\left(\max_{1\le i\le p} W''_i\right) &\sim E(W''_1), \text{ and} \\ E\left(\max_{1\le i\le p} W'''_i\right) &\sim E(W'''_1) \\ & \qquad (\text{as } p \to \infty), \end{aligned}$$

then $\eta(p) \to 1$. The above are also necessary conditions, because all three summands are significant as $p \to \infty$.

The random variable G_i, having a Poisson distribution with parameter $q\beta$, has the same distribution as the sum of q i.i.d. random variables H_{ij} ($1 \le j \le q$) with a Poisson distribution with parameter β. Thus, we are led to the study of the average maximum of p sums of q i.i.d. random variables W_{ij} ($1 \le i \le p$, $1 \le j \le q$) with a distribution that does not change with p and q. In our distributed hash table example, we are interested in the cases in which each W_{ij} is either a Poisson variable, the product of two Poisson variables, or a polynomial of a Poisson variable.

4.3 Average Maximum of Sum of i.i.d. Random Variables

We give sketches of the proofs or cite the results. The details are presented in [Kimura and Ichiyoshi 1991].

4.3.1 Poisson Variable

The asymptotic distribution of the *maximum bucket occupancy* has been analyzed by Kolchin *et al.* [1978]. The following is the result as cited in [Vitter and Flajolet 1990].

Theorem 1 (Kolchin et al.) *If X_i $(1 \leq i \leq p)$ are i.i.d. random variables having a Poisson distribution with parameter μ, the expected maximum bucket occupancy is*

$$\overline{M_\mu} = E\left(\max_{1\leq i\leq p} X_i\right) \sim \begin{cases} \tilde{b} & \text{if } \mu = o(\log p); \\ \mu & \text{if } \mu = \omega(\log p), \end{cases}$$

where $\tilde{b}$ is an integer greater than μ such that

$$\frac{e^{-\mu}\mu^{\tilde{b}+1}}{(\tilde{b}+1)!} \leq \frac{1}{p} < \frac{e^{-\mu}\mu^{\tilde{b}}}{\tilde{b}!}.$$

When $\mu = \Theta(1)$, $\tilde{b} \sim \log p/\log\log p$.

The proof is based on the observation that, as p becomes large, $P\{M_\mu > b\}$ as a function of b approaches the step function having value 1 for b smaller than $\tilde{b}$ and 0 for b larger than $\tilde{b}$, and the expectation of M_μ is equal to its summation from $b = 0$ to $b = \infty$.

We extend Kolchin's theorem to the product of Poisson variables and polynomials of a Poisson variable.

4.3.2 Product of Two Poisson Variables

We introduce a partial order on the class $\mathcal{M}$ of non-negative random variables with a finite mean.

Definition 2 For $X, Y \in \mathcal{M}$, we define $X \prec Y$ iff $E(\max\{X,c\}) \leq E(\max\{Y,c\})$ for all $c \geq 0$.

There are a number of natural properties concerning this partial order. For example, if $X \prec Y$ and Z is independent of X, Y, then $X + Z \prec Y + Z$, $XZ \prec YZ$, $\max\{X,Z\} \prec \max\{Y,Z\}$, etc. Note $X \prec Y_1$ and $X \prec Y_2$ do not imply $2X \prec Y_1 + Y_2$. The utility of $\prec$ in analyzing the expected maximum is illustrated by the following lemma.

Lemma 1 *Let X_i $(1 \leq i \leq p)$ and Y_i $(1 \leq i \leq p)$ be i.i.d. random variables distributed as X and Y. If $X \prec Y$, then*

$$E\left(\max_{1\leq i\leq p} X_i\right) \leq E\left(\max_{1\leq i\leq p} Y_i\right).$$

SKETCH OF PROOF:

$$\begin{aligned} \max\{X_1, X_2, \ldots, X_p\} &\prec \max\{Y_1, X_2, \ldots, X_p\} \\ &\prec \cdots \prec \max\{Y_1, \ldots, Y_p\} \quad \square \end{aligned}$$

For the convex sum of i.i.d. variables, we have the following lemma.

Lemma 2 *Let X_i $(1 \leq i \leq p)$ be i.i.d. random variables distributed as X. For all $a_i \geq 0$ $(1 \leq i \leq p)$ such that $a_1 + \cdots + a_p = 1$, $a_1X_1 + \cdots + a_pX_p \prec X$.*

SKETCH OF PROOF: Let $a_1, a_2 \geq 0$ and $a_1 + a_2 = 1$. For arbitrary $c \geq 0$, $\max\{a_1X_1 + a_2X_2, c\} + \max\{a_1X_2 + a_2X_1, c\} \leq \max\{X_1, c\} + \max\{X_2, c\}$. The expectation of the left hand side is equal to $2E(\max\{a_1X_1 + a_2X_2, c\})$, and that of the right hand side is

$$\begin{aligned} & E(\max\{X_1, c\} + \max\{X_2, c\}) \\ = \;& E(\max\{X_1, c\}) + E(\max\{X_2, c\}) \\ = \;& 2E(\max\{X, c\}). \end{aligned}$$

Thus, $a_1X_1 + a_2X_2 \prec X$. The case for $p > 2$ can be reduced to $p - 1$ using the above. □

Finally, the following lemma gives an upper bound on the sum of the product of two sets of i.i.d. random variables.

Lemma 3 *Let X_i $(1 \leq i \leq rs)$ and Y_i $(1 \leq i \leq rs)$ be i.i.d. random variables. We have*

$$X_1Y_1 + \cdots + X_{rs}Y_{rs} \prec (X_1 + \cdots + X_r)(Y_1 + \cdots + Y_s).$$

SKETCH OF PROOF: We can prove $X_1Y_1 + \cdots X_tY_t + Z \prec X_1(Y_1 + \cdots + Y_t) + Z$ (Z independent of X_is, Y_is) by conditioning Z and using Lemma 2. By repeatedly "collecting" the $X_{ij}Y_{ij}$'s and replacing them with the bracketed terms, we have the desired result. □

Theorem 2 *Let X_i $(1 \leq i \leq q)$ and Y_i $(1 \leq i \leq q)$ be i.i.d. having a Poisson distribution with parameter α and β. If $q = \omega(\log^2 p)$, then*

$$E\left(\max_{1\leq i\leq p} \sum_{1\leq j\leq q} X_{ij}Y_{ij}\right) \sim q\alpha\beta = E\left(\sum_{1\leq j\leq q} X_{1j}Y_{1j}\right)$$

(as $p \to \infty$).

SKETCH OF PROOF: Let $q = r^2$,

$$\begin{aligned} & E\left(\max_{1\leq i\leq p}(X_{i1}Y_{i1} + \cdots + X_{iq}Y_{iq})\right) \\ \leq \;& E\left(\max_{1\leq i\leq p}(X_{i1} + \cdots + X_{ir})(Y_{i1} + \cdots + Y_{ir})\right) \\ \leq \;& E\left(\max_{1\leq i\leq p}(X_{i1} + \cdots)\right) E\left(\max_{1\leq i\leq p}(Y_{i1} + \cdots)\right) \end{aligned}$$

by the Lemma 2 and 3. The sum of r i.i.d. Poisson variables with parameter α is distributed as a Poisson variable with parameter $r\alpha$. Thus, if $r = \omega(\log p)$, then

$$E\left(\max_{1\leq i\leq p}(X_{i1} + \cdots + X_{ir})\right) \sim r\alpha = E(X_{11} + \cdots + X_{1r})$$

by Kolchin's theorem. This is similar for the sum of Y_{ij}. This is what we needed. □

4.3.3 Polynomial of Poisson Variable

The treatment of upper bounds on the expected maximum of the sums of a polynomial of i.i.d. random variables is more involved. We only list the result.

Theorem 3 *Let X_i $(1 \le i \le q)$ be i.i.d. having a Poisson distribution with parameter α, and $c(X)$ be a polynomial of degree $d > 0$ with non-negative coefficients. If $q = \omega(\log^d p)$, then*

$$E\left(\max_{1\le i\le p} \sum_{1\le j\le q} c(X_{ij})\right) \sim qc^*(\alpha) = E\left(\sum_{1\le j\le q} c(X_{ij})\right)$$

(as $p \to \infty$), where $c(X) = a_d^* X^{(d)} + \cdots + a_1^* X^{(1)} + a_0^*$, and $c^*(X) = a_d^* X^d + \cdots + a_1^* X^1 + a_0^*$ ($X^{(k)} = X(X-1)\cdots(X-k+1)$ is the falling power of X).

As for corresponding lower bounds on the necessary growth rate of q, we only know at present that if $q = o((\log p/\log\log p)^2)$, the ratio between the expected maximum and the mean tends to ∞ as $p \to \infty$.

4.4 The Isoefficiency for Load Balance

Now, let us suppose $q = \omega(\log^2 p)$. Then,

$$E\left(\max_{1\le i\le p} G_i\right) \sim E(G_1) \quad (p \to \infty)$$

is immediate from Kolchin's theorem. Also,

$$\begin{aligned} & E\left(\max_{1\le i\le p} \sum_{1\le j\le q} (A_{ij}+1)B_{ij}\right) \\ \le \; & E\left(\max_{1\le i\le p} \sum_{1\le j\le q} A_{ij}B_{ij}\right) + E\left(\max_{1\le i\le p} \sum_{1\le j\le q} B_{ij}\right) \\ \sim \; & E\left(\sum_{1\le j\le q} A_{1j}B_{1j}\right) + E\left(\sum_{1\le j\le q} B_{1j}\right) \end{aligned}$$

by Kolchin's theorem and the proposition for the product of two Poisson variables. Finally, since $X(X+1)/2$ is a polynomial of degree 2,

$$E\left(\max_{1\le i\le p} \sum_{1\le j\le q} \frac{B_{ij}(B_{ij}+1)}{2}\right) \sim E\left(\sum_{1\le j\le q} \frac{B_{1j}(B_{1j}+1)}{2}\right).$$

if $q = \omega(\log^2 p)$.

We have shown that if $q = \omega(\log^2 p)$, the average collective load balance factor $\eta(p) \to 1$ as $p \to \infty$. Therefore, $W = \Theta(pq) = \omega(p\log^2 p)$ is a sufficient condition for isoefficiency for 1.

4.5 Simulation

A simple simulation program was run to test the applicability of the asymptotic analysis for p up to 4096. Fig. 1 shows the results for $\alpha = \beta = 4$, $p = 4$, 16, 64, 256, 1024, and 4096 and $q = 1$, $\lg p$, $\lg^2 p$, and $\lg^3 p$ (lg denotes the logarithm with base 2). The experimental load balance factors (on the vertical axis) are plotted against the number of processors (on the horizontal axis). The experimental load balance factor $\overline{\eta_{p,q}}$ for p, q are calculated by

$$\overline{\eta_{p,q}} = \frac{E\left(\sum_{1\le j\le q} W_{1j}\right)}{\overline{\max_{1\le i\le p}\sum_{1\le j\le q} W_{ij}}},$$

where W_{ij} is one of $X_{ij}, X_{ij}Y_{ij}$ and $X_{ij}^{(2)}$ ((a), (b) and (c), respectively in the figure), and the average $\overline{\max_{1\le i\le p}\sum_{1\le j\le q} W_{ij}}$ is calculated from the result of 50 simulation runs.

X_{ij} and Y_{ij} are generated according to the Bernoulli model (*i.e.*, a table $X[1..pq]$ is prepared, and $n = pq\alpha$ random numbers x's with $x \ge 0$ were generated, each x going to the $((x \bmod p)+1)$-th table entry, etc.). The coefficient of variation (the ratio of standard deviation to average) of $\max_{1\le i\le p}\sum_{1\le j\le q} W_{ij}$ is larger for $X^{(2)}$ and XY than for X, and it decreases as p becomes larger or q becomes larger. Table 1 gives the coefficients of variation for $p = 64$ and 4096.

By and large, the results seem to confirm the asymptotic analysis. For the product and the second falling power, $\Theta(\log^2 p)$ appears to be a sufficient rate of growth of q for η to converge to 1. Even logarithmic growth ($q = \lg p$) does not lead to very poor load balance factors at least up to $p = 4096$ (approx. 0.5 for XY and approx. 0.4 for $X^{(2)}$).

5 Communication Overheads

We briefly discuss the communication overheads when distributed hash tables are implemented on multicomputers. A multicomputer (also referred to as a distributed-memory computer and a message-passing parallel computer) consists of p identical processors connected by some interconnection network. On such computers, the time it takes to transfer a message of length L (in words) from a processor to another which is D hops away in the absence of network contention[4] is $t_s + t_h D + t_w L$, where t_s is the constant start-up time, t_h is the per-hop time, and t_w is the per-word communication time. We choose the mesh architecture for consideration (two-dimensional square meshes in particular) since many of the recent "second generation" multicomputers have such topologies. Examples include J-Machine, Intel Paragon, and parallel inference machines Multi-PSI and PIM/m.

We note that the average traveling distance of a random message (a message from a randomly chosen processor i to another randomly chosen processor i', allowing $i = i'$) is $\frac{2}{3}(\sqrt{p} - \frac{1}{\sqrt{p}}) \sim \frac{2}{3}\sqrt{p}$ on the meshes. It is roughly

[4] Communication latency in the absence of network contention is called *zero-load latency*.

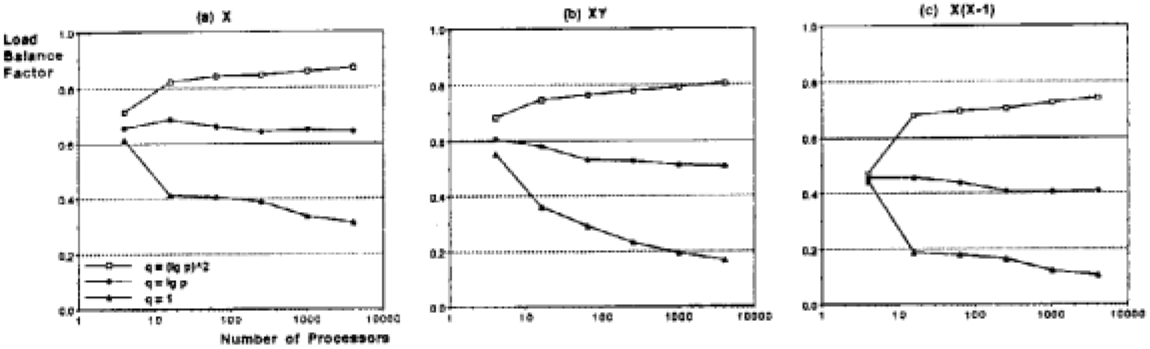

Figure 1: Experimental Load Balance Factors ($\alpha = \beta = 4$)

Table 1: Coefficients of Variation of Maximum Load ($\alpha = \beta = 4$)

	$p = 64$			$p = 4096$		
	$q = 1$	$q = 6$	$q = 36$	$q = 1$	$q = 12$	$q = 144$
X	11.0%	6.3%	2.6%	7.1%	3.5%	1.0%
XY	17.8%	12.2%	5.0%	12.3%	5.3%	2.1%
$X^{(2)}$	24.8%	13.1%	6.0%	15.4%	6.8%	2.6%

1/3 of the diameter of the network, which is $2(\sqrt{p} - 1)$. We can easily see that $W = \Omega(p^{3/2})$ is a necessary and sufficient condition for super isoefficiency due to zero-load latency, which is a situation worse than that due to load imbalance.

In real networks, the impact of message collisions must be taken into account. Instead of estimating the time required for data dispatch using a precise model of contention, we compare the amount of traffic generated by random communication and the capacity of the network. The *traffic* of a message is defined by the product of its traveling distance and its length. It indicates how much network resource (measured by channel × network cycle time) the message consumes. The *capacity* of a network is defined by the sum of the bandwidth of all network channels (channels that connect routers). It indicates the peak throughput of the network. The basic fact is that the time required for completely delivering a set of messages is at least M/C, where M is the total message traffic and C is the capacity of the network.

The average traffic generated by $\sum_{1 \leq i \leq p} G_i$ random messages is $\sim \frac{2}{3}p^{3/2}q\beta L$ (L is the constant message length). The network capacity is $2\sqrt{p}(\sqrt{p} - 1)/t_w \sim 2p/t_w$. Thus, the average data dispatch time is at least $\sim \frac{1}{3}\sqrt{p}q\beta Lt_w = \Theta(\sqrt{p}q) = \omega(T(1)/p)$. This means that meshes with constant channel bandwidth cannot sustain the traffic generated by random communication, forcing the efficiency to approach zero as $p \to \infty$. The network channel bandwidth must grow at least in proportion to $\sqrt{p}$, to maintain the communication latency under heavy random communication within a constant factor of the zero-load latency.

A similar analysis for the hypercube architecture shows that $W = \Omega(p \log p)$ is a necessary and sufficient condition for super-isoefficiency due to zero-load latency, and is less than that due to load imbalance, and that the network capacity has the same growth rate as that of the random traffic.

The degradation of performance due to network contention in the mesh architecture has been pointed out by several authors. Gupta and Kumar [1990] have done scalability analysis for a parallel FFT algorithm, and Singh *et al.* [1990] for parallel quicksort algorithms. In both of these types of algorithms, the communication patterns are nonlocal as in our distributed hash table example, and the growth in the problem size makes local computation per message increase very slowly. This means that isoefficiency function must grow very rapidly (nearly exponential). In our case, since local computation per message does not increase with problem size, it is impossible to maintain efficiency as p gets larger.

Our analysis does *not* suggest that hypercubes are superior to mesh networks for building very large-scale multicomputers. On the contrary, Dally [1990] showed that if we fix the wire bisection of the network, low dimensional cubes (k-ary n-cubes with n small) provide larger throughput than high dimensional cubes (k-ary n-cubes with n large). We believe that future very large-scale multicomputers should provide network bandwidths that can meet the traffic generated by nonlocal communication, if they are to support a wide variety of parallel algorithms, not restricted to ones with high communication locality. Dally [1991] proposes a design of such network architectures.

6 Conclusions

An asymptotic analysis of the load balance of distributed hash tables was conducted, and it was found that, with a constant load factor, $m = \omega(p \log^2 p)$ is a sufficient rate of growth of table size m to balance the load as the number of processors p grows. Communication overheads on multicomputers was also briefly discussed. In the case of mesh multicomputers, unless the network channel bandwidth grows sufficiently as p grows, the network will eventually become a performance bottleneck.

Because of the rather high overheads in encoding and decoding message packets on the part of the processing node, small- to medium-scale multicomputers may not generate enough message traffic to make contention—or, even communication latency—a performance bottleneck [Nakajima and Ichiyoshi 1990, Chittor and Enbody 1990]. But, the bottleneck is bound to show itself in very large-scale multicomputers.

The probabilistic analysis in this paper is fairly general and can be applied to similar load balance problems, such as parallel A^* search with distributed OPEN lists [Kumar *et al.* 1988, Huang and Davis 1988, Manzini 1990]. Kruskal and Weiss [1985] studied parallel runtimes when independent subtasks are allocated on processors, with an (rather restrictive) assumption that the distribution of subtask running times is one with increasing failure rate (IFR). Their analysis was also asymptotic as the number of subtasks and processors becomes large. This paper differs from their study mainly in that (1) the IFR assumption does not hold for the distribution of hash operation costs, and (2) asymptotic (super-)isoefficiency is investigated.

References

[Chittor and Enbody 1990] S. Chittor and R. Enbody. Performance evaluation of mesh-connected wormhole-routed networks for interprocessor communication in multicomputers. In *Proceedings of Supercomputing Conference*, 1990, pp. 647–656.

[Coffman and Lueker 1991] E. G. Coffman, Jr. and G. S. Lueker. *Probabilistic Analysis of Packing and Partitioning Algorithm.* John Wiley & Sons, 1991.

[Dally 1990] W. Dally. Network and processor architecture for message-driven computers. In R. Suaya and G. Birtwistle, editors, *VLSI and Parallel Computation*, chapter 3. Morgan Kauffman Publishers, 1990.

[Dally 1991] W. J. Dally. Express cubes. *IEEE Transactions on Computers*, Vol. 40, No. 9 (1991), pp. 1016–1023.

[Gupta and Kumar 1990] A. Gupta and V. Kumar. On the scalability of FFT on parallel computers. In *Proceedings of the Frontiers 90 Conference on Massively Parallel Computation*, 1990.

[Huang and Davis 1988] S. Huang and L. S. Davis. Parallel iterative A* search: An admissible distributed heuristic search algorithm. In *Proceedings of IJCAI-89*, 1989, pp. 23–29.

[Kimura and Ichiyoshi 1991] K. Kimura and N. Ichiyoshi. Scalability analysis of static load balancing under unpredictable subproblem sizes. ICOT Technical Report, ICOT, 1991. To appear.

[Knuth 1973] D. E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, 1973.

[Kolchin *et al.* 1978] V. F. Kolchin, B. A. Sevast'yanov, and V. P. Chistyakov. *Random Allocations.* V. H. Winston & Sons, 1978.

[Kruskal and Weiss 1985] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 10 (1985), pp. 1001–1016.

[Kumar *et al.* 1988] V. Kumar, K. Ramesh, and V. N. Rao. Parallel best-first search of state space graphs: A summary of results. In *Proceedings of AAAI-88*, 1988, pp. 122–127.

[Kumar and Rao 1987] V. Kumar and V. N. Rao. Parallel depth-first search, part II. *International Journal of Parallel Programming*, Vol. 16, No. 6 (1987), pp. 501–519.

[Manzini 1990] G. Manzini. Probabilistic performance analysis of heuristic search using parallel hash tables. 1990 (Written while the author was at MIT).

[Nakajima and Ichiyoshi 1990] K. Nakajima and N. Ichiyoshi. Evaluation of inter-processor communication in the KL1 implementation on the Multi-PSI. In *Proceedings of ICPP'90*, 1990, pp. 613–614.

[Singh *et al.* 1990] V. Singh, V. Kumar, G. Agha, and C. Tomlinson. Scalability of parallel sorting on mesh multicomputers. Technical Report TR-90-45, Institute of Technology, University of Minnesota, 1990.

[Vitter and Flajolet 1990] J. S. Vitter and P. Flajolet. Average-case analysis of algorithms and data structures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, chapter 9. The MIT Press/Elsevier, 1990.