Dynamic and Randomized Load Distribution in Arbitrary Networks

J.Gaber and B.Toursel

L.I.F.L., Université des Sciences et Technologies de Lille1
59655 Villeneuve d'Ascq cedex -France-
{gaber,toursel}@lifl.lifl.fr

Abstract. We present the analysis of a randomized load distribution algorithm that dynamically embed arbitrary trees in a distributed network with an arbitrary topology. We model a load distribution algorithm by an associeted Markov chain and we show that the randomized load distribution algorithm spreads any M-node tree in a distributed network with N vertices with load $O(\frac{M}{N}+\delta(\varepsilon))$, where $\delta(\varepsilon)$ depends on the convergence rate of the associated Markov chain. Experimental results obtained by the implementation of these load distribution algorithms, to validate our framework, on the two-dimensional mesh of the massively parallel computer MasPar MP-1 with 16,384 processors, and on a network of workstations are also given.

1 Introduction

We model the load distribution problem that dynamically maintains evolving an arbitrary tree in a distributed network as an on-line embedding problem. These algorithms are dynamic in the sense that the tree may starts as one node and grows by dynamically spawning children. The nodes are incrementally embedded as they are spawned.

Bhatt and Cai present in [1,2] a randomized algorithm that dynamically embeds an arbitrary M-node binary tree on a N processor binary hypercube with dilation $O(\log\log N)$ and load $O(M/N+1)$. Leighton and al. present in [3,4] two randomized algorithms for the hypercube and the butterfly. The first algorithm achieves, with high probability, a load $O(M/N+\log N)$ and respectively dilation 1 for the hypercube and dilation 2 for the butterfly. The second algorithm achieves, for the hypecube, a dilation $O(1)$ and load $O(M/N+1)$ with high probability. Kequin Li presents in [5] an optimal randomized algorithm that achieves dilation 1 and load $O(M/N)$ in linear arryas and rings. The algorithm concerns a model of random trees called the *reproduction tree* model [5]. The load distribution algorithm that we describe here work for every arbitrary tree (binary or not) on a *general network*. The analysis use mathematical tools derived from both the Markov chain theory and the numerical analysis of matrix iterative schemes.

2 The Problem

Consider the following needed notations. Let P_j the jth vertex of the host graph and k a given integer. We define, for each vertex P_j, the set

$$\mathcal{V}(P_j) = \{P_{n(j,1)}, P_{n(j,2)}, ..., P_{n(j,k)}\}.$$

This set defines a *logical neighbourhood* for P_j. $n(j, \delta)$ refers to the δ-neighbour of P_j. The terms *logical neighbour* denotes that an element of $\mathcal{V}(P_j)$ is not necessarily closed to P_j in the distributede network.

Consider the behavior of the following mapping algorithm. At any instant in time, any task allocated to some vertex u that does not have k children can spawn a new child task. The newly spawned children must be placed on vertices with satisfying the following conditions as proposed by the paradigm of Bhatt and Cai: (1) without foreknow how the tree will grow in the future, and (2) without accessing any global information and (3) once a task is placed on a particular vertex, it cannot be reallocated subsequently. Hence, the process migration is disallowed and the placement decision must be implemented within the network in a distributed manner, and locally without any global information.

The mapping algorithm that we will describe is randomized and operates as follows. The children of any parent node v initially in a vertex P_j are randomly and uniformly placed on distinct vertices of the set $\mathcal{V}(P_j)$. The probability that a vertex in $\mathcal{V}(P_j)$ is chosen is $1/k$. At the start, the root is placed on an initial vertex which can be fixed or randomly choosen. The randomness is crucial to the success of the algorithm as it will be seen in its analysis.

3 Analysis

As was mentionned, as each node is spawned as a child of a node v initially in P_j, it is assigned a random vertex in the set $\mathcal{V}(P_j)$. The probability that any particular vertex is chosen is $1/k$. As our process has the property that the choice of a vertex destination at any step ℓ depends only on the father's node, i.e., depends only on the state $\ell - 1$, we have thus a Markov's chain whose state space is the set of the N vertices. We construct its transition probability matrix $A = (a_{ij})_{0 \leq i,j \leq N}$ where

$$a_{ij} = \begin{cases} \frac{1}{k} \text{ if } P_j \in \mathcal{V}(P_i) \\ 0 \text{ otherwise} \end{cases}$$

A node distribution can be considered to be a probability distribution and the mapping of the newly spawned nodes is the computation of a one-step transition probabilities. Let p_ℓ be an N vector where any entry $p_\ell(i)$ is the proportion of objects in state i at time ℓ. We denote by p_0 the initial probability distribution. At the state ℓ, we have $p_\ell = p_0 A^\ell$, $\forall \ell \geq 1$ We know that if the matrix A is regular, then the Markov chain has stationary transition probabilities since $\lim_{\ell \to \infty} A^\ell$ exists. This means that for large values of ℓ, the probability of being in state i after ℓ transitions is approximatively equal to $\frac{1}{N}$ (the ith entry

of $\widetilde{p}$) no matter what the initial state was. In other words, as a consequence of the asymptotic behavior of a such Markov process, the distribution converges to uniform distribution which allocates the same amount of nodes to each vertex for very large M. Let p_ℓ be the probability distribution at step ℓ. We have

$$p_\ell = p_{\ell-1}A \quad \text{et} \quad \widetilde{p} = \lim_{\ell\to\infty} p_\ell = \left(\tfrac{1}{N}, \tfrac{1}{N}, ..., \tfrac{1}{N}\right)$$

Let $c(v)$ be the number of outcomes where vertex v is selected. We consider that the Markov chain reaches the stationary phase beginning from $t(\varepsilon)$ (e.g., we choose the smallest t such that each entries of p_t is equal to $1/N + \varepsilon$ where ε goes to 0).

Denote by $k^{(\ell)}$ the number of spawned nodes at any step ℓ. We have $M = \sum_{\ell=0}^{\bar{\ell}} k^{(\ell)}$, with $\bar{\ell} \in [t, \infty[$.

We have $c(v) = \sum_{\ell=0}^{\bar{\ell}} p_\ell(v)k^{(\ell)}$ which become [6] $\underbrace{\sum_{\ell=0}^{t(\varepsilon)}(p_\ell(v) - \widetilde{p}_\ell(v))k^{(\ell)}}_{\alpha(v)} + \frac{M}{N}$

we obtain the vector $\alpha = \sum_{\ell=0}^{t(\varepsilon)}(p_\ell - \widetilde{p})k^{(\ell)}$. We need now to bound α. As stated by Cybenko in [7], if γ is subdominant modulus of A, then we have $\|\underbrace{p_0A^\ell}_{p_\ell} - \widetilde{p}\|^2 \leq \gamma^{2\ell}\|p_0 - \widetilde{p}\|^2$

and thus $\|p_\ell - \widetilde{p}\| \leq \gamma^\ell\|p_0 - \widetilde{p}\|$ and $\|\alpha\| = \|\sum_{\ell=0}^{t(\varepsilon)} k^{(\ell)}(p_\ell - \widetilde{p})\| \leq \sum_{\ell=0}^{t(\varepsilon)} k^{(\ell)}\|p_\ell - \widetilde{p}\| \leq$

$\sum_{\ell=0}^{t(\varepsilon)} k^{(\ell)}\gamma^\ell\|p_0 - \widetilde{p}\|$ thus $\|\alpha\| \leq \|p_0 - \widetilde{p}\| \sum_{\ell=0}^{t(\varepsilon)} k^\ell\gamma^\ell$

Note that we have use the fact that the number of spawned nodes $k^{(\ell)}$ at any step ℓ is at most k^ℓ. We obtain $\|\alpha\| \leq \|p_0 - \widetilde{p}\|\frac{(k\gamma)^{t(\varepsilon)+1} - 1}{k\gamma - 1}$

Recall that $\|.\|_\infty$ of a vector denotes its maximum entry. Denote by $\delta(\varepsilon) = \|\alpha\|_\infty$ the maximum entry of α. Since $p_0 - \widetilde{p} = \left(1 - \frac{1}{N}, -\frac{1}{N}, ...\right)$, we have $\|p_0 - \widetilde{p}\|_\infty = 1 - \frac{1}{N}$

and

$$\delta(\varepsilon) \leq (1 - \frac{1}{N})\frac{1 - (k\gamma)^{t(\varepsilon)+1}}{1 - k\gamma} \tag{1}$$

In view of (1), for small γ, we obtain a small value $\delta(\varepsilon)$. Recall that $\gamma \in]-1, 1]$. For $\gamma = \frac{1}{k} + a$. We compute the Taylor series expansion of $\delta(\varepsilon)$ at $\frac{1}{k}$. This reveals that $\delta(\varepsilon) \leq t(\varepsilon) + 1$, where $t(\varepsilon)$ is the rate of convergence of A (note that $t(\varepsilon)$ is bounded).

With the above analysis we obtain the following theorem.

Theorem 1. *Given a randomized mapping algorithm of a process graph that is an arbitrary dynamic k-ary tree on any arbitrary topology. If the mapping's*

stochastic matrix associated with the logical neighbourhood is regular then the number of nodes mapped on a single vertex of the network is $O(\frac{M}{N} + \delta(\varepsilon))$, *where* $\delta(\varepsilon) = (1 - \frac{1}{N})\frac{(k\gamma)^{t(\varepsilon)+1} - 1}{k\gamma - 1}$, γ *is the subdominant eigenvalue of the mapping's stochastic matrix,* $t(\varepsilon)$ *the number of steps necessary to converge within* ε, M *is the number of nodes and* N *the number of vertices of the network.*

4 Experimentations

We ran a parallel implementation of an algorithm wherein an arbitrary tree grows during the course of the computation (as in branch-and-bound serach, divide-and-conquer or game tree evaluation), on a 128 × 128 mesh. The following table gives the experimental results in terms of the maximum load obtained when the randomized algorithm is (resp. is not) used to balance load (each line shows the result in terms of the maximum load obtained when we embed an arbitrary tree. The first column gives the total nodes number of this tree).

Number of nodes	load without randomized algorithm	load with randomized algorithm	ideal load
3123	93.00	1.17	1
55791	219.00	4.59	3.41
65135	4.00	4.00	3.98
99325	16.00	6.82	6.06
204383	64.00	13.43	12.47
731923	64.00	45.86	44.67
1640123	256.00	102.59	100.11

We ran, on a network of 8 stations with a multithreaded environement, a parallel implementation of an algorithm wherein an arbitrary binary tree grow during the course of the computation. The following table gives the effective load of each station obtained with and without the randomized algorithm. The value ToTal denotes the total nodes number of the embeded arbitrary tree.

5 Conclusion

Theorem 1 establishes that for a given mapping function, a simple randomized load distribution algorithm as described in section 2 maintains dynamically evolving an arbitrary tree on a general distributed network with a load $O(\frac{M}{N} + \delta(\varepsilon))$, where $\delta(\varepsilon)$ depends on the mapping function. This implies that we can easily compare mapping functions just by computing eigenvalues of the associated adjancy matrix.

N=8			
Numéro Station	load without randomized algorithm	load with randomized algorithm	ideal load *ToTal/N*
0	4171	16106	17271.50
1	4012	16164	17271.50
2	7302	17050	17271.50
3	7289	16012	17271.50
4	3605	16627	17271.50
5	3412	16597	17271.50
6	51722	20394	17271.50
7	56659	19222	17271.50
ToTal	138172.00		

Acknowledgment

We are grateful to F.T.Leighton for the helpful comments and K.Li for helpful discussions. We are also grateful to F.Chung[8], S.Bhatt and G.Cybenko for providing us a helpful papers and suggestions.

References

1. S.N. Bhatt and J.-Y. Cai. Talk a walk, grow a tree. *in Proc. 29th Annual IEEE Symposuim on Foundations of Computer, Science, IEEE CS, Washington, DC, pp. 469-478*, 1988.
2. S.Bhatt and J-Y.Cai. Taking random walks to grow trees in hypercubes. *Journal of the ACM*, 40(3):741–764, July 1993.
3. F.T. Leighton, M.J. Newman, A.G. Ranade, and E.J. Schwabe. Dynamic tree embeddings in butterflies and hypercubes. *Siam Journal on computing*, 21(4):639–654, August 1992.
4. F.T. Leighton. *Introduction to parallel algorithms and architectures.* Morgan Kauffmann Publishers., 1992. Traduit en Français par P.Fraignaud et E.Fleury, International Thomson publishing France, 1995.
5. K. Li. Analysis of randomized load distribution for reproduction trees in linear arrays and rings. *Proc. 11th Annual International Symposium on High Performance Computers, Winnipeg, Manitoba, Canada (10-12), July*, 1997.
6. J. Gaber. Plongement et manipulations d'arbres dans les architectures distribuées. *Thèse LIFL*, Janvier 1998.
7. G. Cybenko. Dynamic load balancing for distributed memory architecture. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.
8. Fan.R.K. Chung. *Spectral Graph Theory.* AMS., 1997.