Institute e-Austria in Timisoara

IeAT Report Series

Experiments on the local load balancing algorithms Part 1

Ştefan Măruşter

2004

Experiments on the local load balancing algorithms; part 1

Ştefan Măruşter

Institute e-Austria Timisoara
West University of Timişoara, Romania
maruster@info.uvt.ro

Abstract. In this paper the influence of the amount of load which can be transferred between processors on the convergence and stability of the common static and dynamic load balancing algorithms are studied.

AMS Subject Classification: G.1.0., G.2.2., C.2.3.
Keywords and phrases: load balancing, stability.

1 Introduction

The load balancing problem arise as a natural matter in parallel computer architecture. A number of models and algorithms have been proposed for to solve it; especially so called *local* algorithms was studied, in which it is assumed that does not exists a central processor that coordinate the load balancing and that every processor contributes to perform it. The main goal of these algorithms is to control the processing in a parallel computer (or in a cluster of computers) such that the performance of the system be improved. The algorithms attempt to distribute the total load among the processors of the system as evenly as possible, so that to ensure that certain processors are not over-loaded while others are left idle. Briefly, the central idea of a local load balancing algorithm is to transfer some load from the highly loaded processor to its less loaded neighbors.

There exist two main classes of local algorithms [1].

(1) *Static algorithms.* The total load is available at the start of processing and no new load is added or existing load is removed. The static algorithms distribute the total load amongst processors before initiating the processing. From different reasons, new imbalance can appear soon after the processing was started, so that the load balancing algorithm must periodically called.

(2) *Dynamic algorithms.* The load is dynamically balanced. It is allowed at the beginning of each time (round) to introduce load at some processors (corresponding to the new jobs) and remove load from others (corresponding to jobs that have finished). Subsequently, the algorithm transfers loads across edges to tray to maintains the balance. Of course, the dynamic load balancing is more corresponding to real case and more challenging than the static version. Note that the variable topology of the network is also interprets as a characteristics of dynamic algorithms [2]

The static load balancing problem have been chiefly studied and various algorithms have been proposed. Conditions on the amount of the load which is transported and on the delay in transferring process, are usually imposed. The common results for static case state that load in every processor tends to the average of the total initial load [3]. In [4] the *single-port* model and *multi-port* model were analyzed and tight time bounds and imbalance bounds have been established in the both cases. It is supposed that in one unit of time at most one task (token) can be transmitted across an edge of the graph in each direction.

The effect of time delays on the stability of dynamic load balancing algorithms has been considered in [5], [6], [7], both for linear and for nonlinear cases. A processor receives the information concerning the level of the load of the neighborings processors delayed by a finite amount of time and then it uses this information to compute its local estimate of the average load in the network. The transfer itself is delayed as well. The stability of the processors loads is established provided that the both delays are bounded.

A new framework for dynamic load balancing in which the jobs traffic is modelled by an adversary have been introduced in [1] and further developed in [8]. The adversary controls the arrivals and removals of loads in the process. If the height of load in any processor and at any time is kept close to the average height of all processors , it says that the algorithm is stable against the given adversary. The main condition for stability is the *cut* condition which limits the increase of load in any subset of processors.

In [9] a static load balancing algorithm have been considered in the frame of discrete event systems. Suitable Lyapunov function is constructed for stability analysis.

Local balancing algorithms restricted to some particular network (asynchronous ring and a network consisting in two servers) have been studied in [10] and [11]

Usually, one considers that a single unit of load (a token) is transmitted across each edge in a round of time. If it supposed that a certain amount of load may transferred between two processors, then one of the key in the performance (rate of convergence, stability, etc.) of load balancing algorithms is just this amount of load. In this paper the influence of the amount of load which can be transferred between processors on the convergence and stability of the common static and dynamic load balancing algorithms are studied.

2 The model

The network is described by a connected undirected graph $G = (V, E)$, where E is a finite set (the vertex) representing the processors, $Card(V) = n$, and $E \subseteq V \times V$ (the edges) is the set of arcs connecting the processors. There exists a general task which must be executed by network; the processors are cooperating for to achieve this goal. We shall suppose that the general task is shared in small subtasks (tokens) and that any token can be executed by any processor. The tokens are spreading over processors; we shall suppose that the load (the tokens) which is keeps by a processor may be described by a continuous variable.

Let $x_i(t)$ be the level of load (the number of tokens) of the processor i at the time t. Each processor in the network sends its level of load to all its neighbors. A neighbor processor j receives this information form processor i delayed by a finite amount of time, τ_{ij}, that is, it receives $x_j(t-\tau_{ij})$. These information about load levels of others processors are used by each processor to compute the amount of load which is transferred to its neighbors. Further, the load sent by the processor i is received by processor j with the same delay.

Let a_t denote the average level of load per processor in the system at the beginning of time t. We say that a load balancing algorithm is stable if there exists constant B such that $|x_i(t) - a_t| \leq B$ for all processors and time t. For a set $S \subseteq V$, let $e(S)$ denote the set of edges with exactly one end in S, and let $\delta_t(S)$ be the net increase in level in set S due to the addition and removal of tokens in time t (note that $\delta_t(S)$ can be negative). We say that the dynamics of the system satisfies *cut* condition [8] if

$$|\delta_t(S) - |S|(a_{t+1} - a_t)| \leq |e(S)|.$$

Let $N_i \subseteq V$ denote the neighbors of the processor i, $N_i = \{j|(i,j) \in E\}$.

The following algorithm uses a weight factor c_i for each processor i which control the amount of tokens transferred from the processor i to every its neighbors. Let $\delta_{ij}(t) = x_i(t) - x_j(t-\tau_{ij})$ denote the total amount which processor i would transfers to processors j if no weight factor is used.

The algorithm:

```
At each time $t$ and for each processor
    for all $j \in N_i$
        If $\delta_{ij}(t) > 0$ then send $c_i\delta_{ij}$ tokens from $i$ to $j$.
```

3 Implementation on non-parallel computers

The implementation on a non-parallel computer is motivated for the possibilities of the delays control; it can simply assign an integer values to each delay τ_{ij} and suitable organize the processing. For instance, if all processors are executing the same task (say an iterative scheme for linear or nonlinear equations) in a synchronous mode, it can simulate this by building a random ordering in covering the cycle.

To each processor i it is assigned a vector x_i which memorizes the successive values of the numbers of tokens in that processor as time progress, that is the values $x_i(0), x_i(1), ...$

Static load balancing without delays

Algorithm SLBOD

```
    $x_i(0) := x0_i, \quad i = 1, ..., n;$
    For $t = 0, 1, ...$
        For $i \in [1, n]$ in random mode
```

$$x_i(t+1) := x_i(t) + \sum_{j\in N_i} c_j(x_j(t) - x_i(t)). \tag{1}$$

The values $x0_i, \quad i = 1, ..., n$, represent the initial loads of processors and let $\sum x0_i = L$ be the total initial load. This initial load is conserved, that is $\sum x_i(t) = L$ for all t. The conservation property result simply from the equation

$$\sum_{i=1}^{n} \sum_{j\in N_i} c_j(x_j(t) - x_i(t)) = 0. \tag{2}$$

The random execution of load balancing means, in fact, the random order in which iteration formula (1) is executed with respect the index i.

Static load balancing with delays

Initially, all components of state vectors are setting with values zero. In this way, a load sent from a processor i to processor j at the time t and which will reach the processor j at time $t + \tau_{ij}$, may be accumulate in the position $x_i(t + \tau_{ij})$. Let τ_m be the maximum values of delays, $\tau_m = max\tau_{ij}$. The components $x_i(0), ..., x_i(\tau_m)$ must be setting with the initials values in initialization step also. Note that the comparison between the levels of two neighboring processors is supposed delayed with the same value, that is $x_i(t)$ must be compared with $x_j(t - \tau_{ij})$. Let be $\delta_{ij}(t) := x_i(t) - x_i(t) - x_j(t - \tau_{ij})$.

Algorithm SLBWD

```
    For $i = 1, ..., n$ and $fort = 0, ..., \tau_m$
        $x_i(t) := x0_i(t);$
    For $t = \tau_m + 1, ...$
        For $i = 1, ..., n$
            $x_i(t+1) := x_i(t+1) + x_i(t);$
        For $i \in [1, n]$ in random mode
            for $j \in N_i$
                if $\delta_{ij} > 0$
```

$$x_i(t+1) := x_i(t+1) - c_j\delta_{ij}(t); \tag{3}$$

$$x_j(t + \tau ij) := x_j(t + \tau ij) + c_j\delta_{ij}. \tag{4}$$

The kernel of the SLBWD are the iterative formulas (3), (4), in which the principle of the algorithm is performed. If the processor i is highly loaded in comparison with its neighbor j, that is $\delta_{ij}(t) := x_i(t) - x_i(t) - x_j(t - \tau_{ij}) > 0$, then a portion of its load, controlled by c_j is transferred to processor j.

Dynamic load balancing with delays

Suppose that the arrivals and removals of load to the processors are performed with a specific , constant amount of load for each processor. This means that the load variation is linear in time. Such dynamics can be simulated by adding in the algorithm a new step of the form $x_i(t) := x_i(t) + \Delta l_i$ at the beginning or of the form $x_i(t+1) := x_i(t+1) + \Delta l_i$ at the end of the cycle for i in SLBWD algorithm.

References

1. Muthukrishnan, M., Rajaraman, R, An adversial model for distributed dynamic load balaning, Proc. ACM Symp. on Parallel Algorithms and Architecture, 1998.
2. Aiello, W., Awerbuch, B., Maggs, B., Rao,S., Approximate load balancing on dynamic and asynchronous networks, Proc. ACM Symp. on Theory of Computing, 1993.
3. Bertsekas, D.P., Tsitsiklis, J.N., Parallel and distributed computation: Numerical Methods, MIT, Athena Scientific, Belmont, Massachusetts, 1997.
4. Bhaskar Ghosh, et. al., Tight analyses of the local load balancing algorithms, from papper list of home page.
5. Abdallah, C.T., et.al., Load balancing instabilies due to time delays in parallel computation, Proceedings of the 3th IFAC Conference on Time Delay Systems, Dec. 2001, Santa Fe, NM.
6. Abdallah, C.T., et.al., The effect of time delays in the stability of load balancing algorithms for parallel computation, from papper list of home page.
7. Birdwell, J.D, et. al., The effect of time delays in the stability of load balancing algorithms for parallel computations, from papper list of home page.
8. Anshelevich,E., Kempe, D., Keinberg, J., Stability of load balancing algorithms indynamic adversarial systems
9. Burges, K.L., Passino, K.M., Stability analysis of load balancing systems, Int. Journal of Control, Vol. 61, No. 2 (1995), pp. 357-393
10. Gehrke, J.E., Plaxton, C.G., Rajaraman, R., Rapid convergence of a load balancing algorithm for asynchronous rings
11. Kleinberg, J.M., A lower bound for two servrs balancing algorithms