

# Dynamic Load Balancing Design and Modeling in MPIAB

Srinivas Mandalapu

Dept. of Computer science

Southern Illinois University, Carbondale

smanda@cs.siu.edu

## ABSTRACT

**This paper presents a design and a model to describe the dynamic load balancing methodology used in MPIAB (Message Passing Interface – Agent Based), an agent based architecture for parallel programming. The design of the dynamic load balancing used in MPIAB provides automatic determination of the efficient hosts even while the task is being executed such that the overall system-wide performance is optimized. The design is achieved by monitoring the load on the host in short intervals of time and providing a mechanism for maintaining the connections with other agents with fewer overheads. The model is composed of functional differential equations. The queue length of tasks at each host plays a major part in modeling. The variables (the number and size of queue, etc.) in the model are proved to remain nonnegative, which is in agreement with the physical background of the variables.**

## Keywords

**Dynamic load balancing, agents, Parallel processing, MPI, Distributed systems.**

## 1. INTRODUCTION

Parallel and distributed computing is the process that aims at breaking down a computation into components, which can then be solved independently on an interconnection network. The technology allows the researchers to solve compute-intensive problems in lesser time using inexpensive commercial-off-the-shelves computing nodes.

Unfortunately, distributed applications often face the problem of load imbalance in a heterogeneous environment. Load imbalance can degrade the application performance greatly. A high degree of reliability and system utilization can be achieved if computers evenly share the network load. Thus, an efficient resource allocation technique should be incorporated in a distributed system in such a way that the transparent and fair distribution of the load can be achieved.

Load balancing can be defined as follows: *Given a collection of tasks comprising a computation and a set of computers on which these tasks may be executed, find the mapping of that tasks to computers that results in each computer having a proximally equal amount of work.* Dynamic load balancing shifts workloads to adapt to changes in environment. The technique assumes very little prior knowledge about the system and applications.

Ideally load balancing should be performed as frequently as possible to track load changes as closely as possible. However,

load-balancing costs make this impractical. First there is overhead associated with collecting the information to make a load balancing decision and increasing the load balancing frequency might make the overhead unacceptable. Second there is overhead associated with moving work, which means that it is impractical to trace load changes that happen very quickly and trying to do so will result in unnecessary overhead. These two sources of overhead place an upper limit on the frequency of load balancing [1].

MPIAB (Message passing Interface – Agent Based) is agent-based architecture for parallel processing [2]. MPIAB is designed to model standard MPI library using Java and addresses two major problems faced by the standard MPI and PVM models. Firstly, these models do not offer automatic node selection for participating computers, so load balancing of the network cannot be incorporated. Secondly, they are not fully platform independent; therefore, interoperability between different implementations is not easily achieved.

The objective of the dynamic load balancing technique in MPIAB is to execute the submitted task code on the selected under-loaded nodes in the network till the completion of the task. Under-loaded nodes are selected before the execution and their under-loaded status is maintained throughout the execution. The design methodology is to have a threshold value and whenever the load on the system exceeds the threshold value and the load continues to exceed the threshold value for a specified amount of time another under-loaded node is selected and execution is transferred to that node. The frequency of the checking the load on the nodes and maintenance of the connections while moving to another node are also considered.

In this paper, the functional differential equations, adapted from the model in [5], are used to describe the agent-based load balancing process with main focus on the queue length. Questions to be addressed here are (1) what will be the effect of random tasks' service time and (2) what will be the effect of random tasks arrival time in the dynamic behavior of load balancing. The variables (the number and size of queue, etc.) in the model are proved to remain nonnegative, which is in agreement with the physical background of the variables.

In the following sections, the dynamic load balancing design and its model is presented. Section 2 mainly discusses the related work and section 3 describes the design. Section 4 gives the model and section 5 deals with the theoretical proofs. Finally section 6 summarizes the paper.

## 2. RELATED WORK

Load balancing techniques have been predominantly applied to client-server based architectures in the past. More recently with the advent of mobile agent paradigm, load balancing is being adapted to this field as well [3]. A mobile agent is an autonomous software agent capable of moving across the network to perform a desired task. The use of mobile agents for dynamic scheduling is an effective choice. This is because of the inherent strengths of this such as improvement in latency and bandwidth over client-server applications as well as reduction of vulnerability due to network disconnection. There is a crucial distinction between the two approaches. Instead of having stationary load balancing methods, which require communication with potential hosts in the network, the mobile agent technique subsumes the concept of mobility and autonomy without central supervision.

A methodology for load balancing in MPIAB has been proposed by Meha et al [4]. The design objective of this methodology is to utilize a combination of deterministic and heuristic methods to select the prospective under-loaded nodes on which a parallel task code can be executed. A list of prospective nodes is selected depending on the task code complexity and system resource usage. These short listed nodes are given as an input to heuristic mechanism, which uses a reward penalty technique to determine the final selection of the hosts on which the task is most optimally executed.

As discussed above once the final selection of hosts is made there is no further check whether the hosts in which the task is being executed are overloaded or not. The host may be overloaded while executing the task due to some other applications, and at this moment we have to move the execution of the task to some other under-loaded host. The design proposed in this paper handles these situations which is an enhancement to [4].

## 3. DYNAMIC LOAD BALANCING

The proposed design exploits real time system resource utilization till the completion of the task with minimum amount of overhead in communication between the agents.

The hosts which are selected in [4] and they are given as input to dynamic load balancing in which we keep on monitoring the loads on the hosts and when we get a host which has fewer loads than the present execution, then we move the execution to that node. This process is done till the completion of the execution of the task code.

Figure 1 illustrates the flowchart of the dynamic load balancing.

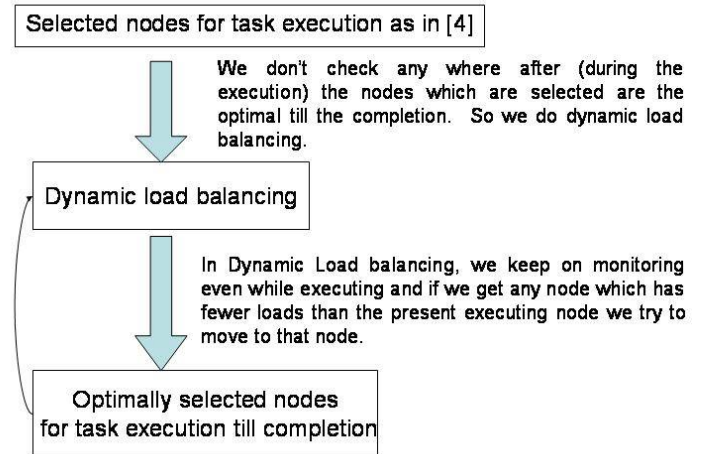


Figure 1 : Dynamic load balancing approach

Main points, which are considered in the design, are:

1. Calculating the function for getting the system load and choosing a correct threshold value.
2. How the agents are moved maintaining the connectivity with other agents with least overhead.
3. Calculating the frequency of the load balancing checks

### 3.1 Function for system load

The three factors, which are considered to get the current system load on the processor, are:

#### 3.1.a. Processor

##### i. Processor time:

This is the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the duration of the idle thread is active in the sample interval, and subtracting that time from interval duration.

##### ii. Processor Queue Length

This is the number of threads in the processor queue. There is a single queue for processor time even on computers with multiple processors. A sustained processor queue of less than 10 threads per processor is normally acceptable, dependent of the workload.

#### 3.1.b. Memory

##### i. Available Mbytes:

This is the amount of physical memory available to processes running on the computer, in Megabytes, rather than bytes as reported in Memory\Available Bytes. It is calculated by adding the amount of space on the Zeroed, Free, and Stand by memory lists.

##### ii. Pages/sec

This is the rate at which pages are read from or written to disk to resolve hard page faults. This counter is a primary indicator of the kinds of faults that cause system-wide delays.

### 3.1.c. Physical disk

#### i. Avg. Disk Queue Length

This is the average number of both read and writes requests that were queued for the selected disk during the sample interval.

Load is calculated as:

$Load\_value = c1 * (\text{processor\_time/processor queue length}) + c2 * (\text{available Mbytes} + \text{Pages/sec}) + c3 * \text{avg.disk queue length}$

Here  $c1, c2, c3$  are weights of the parameters and they add up to 1.

$$\text{Percent\_Load} = 100 * (1 - D / (D + Load\_value))$$

Here  $D$  determines which load value should be associated with which percentage. Choosing  $D = 50$  means that 128 is 60% load, 256 is 80%, 512 is 90%, and so on.

### 3.2 Maintaining the connection:

**Message Buffering and Forwarding Service:** This service provides a persistent area to buffer and forward messages for agents in transient. One or more message relay agent (MRA) provides this service.

Every MRA agent has the transient table, location table and message table.

Transient table has two attributes:

1. Agent Name: A unique name that identifies the agent in transient.
2. Target Address: The network address where the agent is migrating.

The location table for agents has four attributes:

1. Agent Name: A unique name that identifies each agent.
2. Physical Address: The network address where the agent is currently executing.
3. MRA Agent Name: The name of the agents' MRA service provider.
4. Agent State: The agent mobility state. This attribute is used by the middleware to decide how to route messages. The state attribute takes one of these values: stationary, in transient, arrival, and off-line.

The message relay agent is responsible for storing messages. The MRA agent buffers messages for agents in transient. The MRA agent is equipped with a special registry to maintain message information. The registry keeps track of the message table.

The message table has five attributes:

1. Agent name: A unique name that identifies the recipient of the message.
2. Message ID: A unique serial message identifier used for indexing and text retrieval.
3. Message envelope: The message sender and recipient envelope. Essentially this attribute represents the message header.

4. Message contents: The message body represented by the actual text of the message.

5. Time stamp: Message arrival date and time.

Buffering a message is triggered by an event that is present in the Manager agent (MA) after the agent informs the MA that it is in transient. At arrival the MA instructs the MRA agent to deliver its messages or it may retrieve the messages itself. The MRA reassembles the message from the message envelope and the message content fields and routes the message to the recipient.

### 3.3 Frequency

After seeing the overhead for calculating the load on the system and the overhead for maintaining the communication, the frequency is determined.

## 4. MODEL

Each task is carried by a task agent (TA). The TA's are distributed to the hosts using the load balancing mechanism present in the MPIAB [4], which uses the services of resource agent (RA) present in it. A queue is maintained at the hosts and TA's have local information about the queue lengths on networks, but they do not have the global knowledge on the state and shape of network.

### 4.1 Description

Task agents can leave or join the queue at the service host, or form a new queue if there are no other agents there. After joining a queue, a single task agent can also leave the queue and moves to other hosts. Therefore, agents' behavior can be decomposed into two elements: leaving and queuing. Local factors, such as initial conditions, strategies for leaving and queuing will determine the global dynamic behavior of the system. The following mechanism shows the formation of queues at the hosts.

---

```
00 Phase I:
01           Initial work of MPIAB is completed.
02           Queues are maintained at each node.
03 START:
04           JoinQueue (TA).
05           if (queue_length > 1)
06               goto Phase II.
07           else
08               goto EXIT.
09 Phase II:
10           If (LB mechanism == transfer)
11               LeaveQueue (TA).
12               queue_length = queue_length -1.
13           goto START.
14 EXIT:
15           FinalQueue (TA).
16 END.
```

---

Algorithm for Queue formation in DLB.

As shown in the above algorithm, initially the MPIAB will assign the task agents to the least busy hosts, which are available in the network. Queues of TA are maintained at each host and the servicing of the task agents starts. If the queue length at the hosts is more than one then phase II is started. In this we again check the load on the host, which is done by the load balancing mechanism, provided by MPIAB and if it asks for the transfer of the TA, the TA leaves the queue and joins the other host. This process continues until all the task agents finish their execution.

#### 4.2 Assumptions

- Agents follow the load balancing strategies of leaving and queuing.
- On networks, every host can generate tasks (agents) and supply the service which is needed by task agents.
- During the initial period  $[0, \tau]$  where  $\tau$  is the maximum time taken to distribute the TA's, agents either queue in the various nodes or waiting to join the queue.

#### 4.3 Dynamic Model

Let 'y' denote the number of TA, which are yet to be assigned to a node,  $y \geq 0$ . Let 'y<sub>s</sub>' denote the number of queues at the nodes whose size is s,  $y_s \geq 0$ . Let 'S' be the total number of task agents generated.

$$y(t) + \sum_{s=1}^m s y_s(t) = S, t \in [0, \tau]$$

During the initial period  $[0, \tau]$ , that is before completely assigning the tasks, the task agents, which are yet to be assigned,  $y(t)$  and the task agents which are queued at different nodes  $s y_s(t)$  gives us the total number of agents present in the network.

Before going into the model let us see a sample scenario focusing on how the queue length of size one is affected by the dynamic load balancing algorithm. This can be maintained in the following ways:

- TA joins the empty queue
- TA leaves a node whose queue length is two
- A node having a queue length of two completes the execution of the task
- TA joins the node of queue length one
- A new load balancing task arrives and produces a TA

These can be modeled as

$$\frac{dy_1(t)}{dt} = \lambda y(t - \tau) + d_2 y_2(t) - c_1 y(t - \tau) y_1(t) + J_1(t - \tau, t) - F_1(t - \tau, t) \quad (1)$$

$$\frac{dy_s(t)}{dt} = d_{s+1} y_{s+1}(t) + c_{s-1}(t - \tau) y_{s-1}(t) - d_s y_s(t) - c_s y(t - \tau) y_s(t) + J_s(t - \tau, t) - F_s(t - \tau, t) \quad (2)$$

where

$$\begin{aligned} \tau > 0, 0 < \lambda < 1, \\ 0 < c_s < 1, 0 < d_s < 1, \\ y_s \geq 0, y \geq 0, \end{aligned}$$

$\frac{dy_s(t)}{dt}$  is the change rate of queue length 's'

Parameter  $\lambda$  is the rate at which a task agent will be assigned to an idle node and forms a new team of size one, therefore we have  $\lambda < 1$ .

'c<sub>s</sub>' is the rate of joining team of 's'  $0 < c_s < 1$ .

'd<sub>s</sub>' is the rate of leaving team 's'  $0 < d_s < 1$ .

In the equation (1), the change rate of queue lengths of size one is described.

Each term is denoted as follows:

$\lambda y(t - \tau)$  - task agent joins an idle node and forms a queue of size one at time t.

$d_2 y_2(t)$  - queue length of size two at time t becomes a team of size one after an agent's leaving.

$c_1 y(t - \tau) y_1(t)$  - queue length of size one becomes two at time t after a task agent comes and join a queue of length one.

$J_1(t - \tau, t)$  - counting process which denotes the number of external task arrival at that host at the interval  $[t - \tau, t]$ . To capture any possible burst-like characteristics in the external-task arrivals the process  $J_i(\cdot, \cdot)$  is a compound Poisson process. That is,  $J_i(t - \tau, t) = \sum_{k: t - \tau < t_k \leq t} H_k$ , where  $t_k$  are the arrival times of job requests (which arrive according to a Poisson process with rate  $\lambda_i$ ) and  $H_k$  ( $k = 1, 2, \dots$ ) is an integer-valued random variable describing the number of tasks associated with the  $k^{\text{th}}$  job request.

$F_1(t - \tau, t)$  - is a Poisson process with rate  $\mu_1$  describing the random number of tasks finishing in the interval  $[t - \tau, t]$ .

Equation (2) describes the same as equation (1) but it is general representation of equation (1).

## 5. PROOFS

In this section two things are considered. One is the proof for the positive ness of the size of the queue length and number of task agents and the second one is the stability of the dynamic load balancing mechanism when there is no more arrival of the external tasks.

For the simplicity of the proofs let us assume the maximum size of the queue is two. So the model becomes

$$\frac{dy_1(t)}{dt} = \lambda y(t - \tau) + dy_2(t) - cy(t - \tau)y_1(t) + J_1(t - \tau, t) - F_1(t - \tau, t) \quad - (1)$$

$$\frac{dy_2(t)}{dt} = cy(t - \tau)y_1(t) - dy_2(t) + J_s(t - \tau, t) - F_s(t - \tau, t) \quad - (2)$$

Positive ness of the variables:

Proof: The variables which are in consideration are  $y_1(t)$  and  $y_2(t)$ . As we can see from the above equation the two negative terms are for the change in the queue length of one to two and the servicing the agents. These can never be greater than the combination of the arrival of the external jobs, formation of a queue at the idle node and the queue length of size two becoming one because the agent left due to load balancing mechanism. This gives us that the change rate of queue length of size one is positive and which strongly implies that the variable  $y_1(t)$  is greater than zero all the times.

Arrival of Steady state:

Proof: It can only possible when there are no external tasks on the hosts. If that is the case then the dynamic load balancing at some point allocates the task agents equally among the hosts depending on their load. Once the servicing of the task agents starts the model reaches the steady state.

## 6. SUMMARY

Load balancing is an important consideration for parallel computing. The use of mobile agents in MPIAB architecture provides the advantages of the reduction of bandwidth and latency compared to the client-server paradigm. The design framework for dynamic load balancing of MPIAB has been proposed with the view of optimization of the system performance by executing the task codes on the nodes, which are maximally under-loaded. The proposed load balancing methodology deployed in MPIAB is an enhancement to the one proposed in [4]. A detailed explanation of the model to establish the close relationship with agent-based load balancing mechanism is given. The positive ness that has shown here is important to the effectiveness of the model.

Some open problems:

In Design:

- The calculation of the load on the system is not exactly accurate.
- The frequency of the load checking is not predetermined

In Model:

- Positive ness of the variables is not proven strictly
- Arrival of steady state is not modeled when the task agents keep on changing the queues.

As the future extension of the project the theoretical proofs which have been given can be backed up with numerical simulations taking different conditions into consideration.

## REFERENCES

- [1] Bruce Siegell and Peter Steenkiste, "Automatic Generation of Parallel Programs with Dynamic Load Balancing", Proceedings of the Third International Symposium on High-Performance Distributed Computing, IEEE, San Francisco, August 1994
- [2] Shahram Rahimi, Ajay Narayanan, Meha Sabharwal, "MPIAB: A Novel Agent Architecture for Parallel Processing", Proceedings of The IEEE International Conference on Intelligent Agent Technology (IEEE-IAT 2003), Halifax, Canada, pp. 554-557, 2003.
- [3] Sasa Desic, Darko Huljenic "Agent based Load balancing with component distribution capability" International Symposium on Cluster Computing and the Grid (CCGRID'02), May 21-24, 2002.
- [4] Meha Sabharwal, Shahram rahimi, Raheel Ahmad, Ajay Narayanan "A Deterministic - Heuristic load balancing methodology for MPIAB", Proceedings of the 19th International Conference on Computers and Their Applications, Seattle, WA.
- [5] Yuanshi Wang, Jiming Liu, Xiaolong Jin "Modeling Agent-Based Load Balancing with Time Delays", IEEE/WIC International Conference on Intelligent Agent Technology, October 13 - 17, 2003 Halifax, Canada.