1 Introduction

In a distributed computing system (DCS) where processing nodes are connected by a local area network, it is very likely that some nodes have higher transient workload than others. To improve system throughput, a dynamic load distribution (LD) algorithm uses current system load information to smooth out such workload imbalance. This is accomplished by relocating application tasks from busy nodes to lightly-loaded (or idle) nodes.

Most earlier studies on dynamic LD algorithms are targeted for homogeneous systems. In a homogeneous system, processing nodes are not only functionally equivalent, but also have identical (or very close) processing speeds.

In recent years, heterogeneous systems are being built with inter-operable software and hardware components.

Menasce and Almeida classified heterogeneity into two categories:

- Throughput Heterogeneity
  Processing nodes in the system are functionally identical but vary in processing speeds. With this type of heterogeneity, surplus workload of a busy node can be distributed to any other node in the system.

- Functional Heterogeneity
  At least a subset of nodes in the system are functionally inequivalent. Such nodes either run different softwares or have different architectures and incompatible instruction sets. With functional heterogeneity, processing nodes can share its workload with only a subset (possibly empty) of nodes in the system.
It has been shown that LD algorithms designed for homogeneous systems may not provide satisfactory performance in heterogeneous systems. In recent years, LD algorithms have been designed specifically for heterogeneous systems.

However, such algorithms are mostly based on single task assignments. That is, at most one task is transferred from a sender to a receiver during each negotiation session. Because of the differences in processing speeds and the amount of spare processing capacities, the maximum amount of workload that a receiver can accept without exceeding its own capacity varies tremendously. Single task assignments are too conservative when applied to heterogeneous systems. The performance gain that can be achieved is hindered.

In this paper, we propose a class of adaptive LD algorithms designed for heterogeneous systems exhibiting throughput heterogeneity.

The rest of the paper is organized as follows.

Section 2 on page 2 discusses basic concepts and some related work. Section 3 on page 3 presents the system model and assumptions. Section 4 on page 4 is the conclusion.

2 Background Study

Casavant and Kuhl proposed a classification of LD algorithms. With static algorithms, task assignment decisions are made a priori during compilation time and are to remain unchanged during runtime. In contrast, dynamic algorithms use current system load information for run-time assignments of tasks to processing nodes.

An early work on static algorithms by Stone used a graph theoretic Max Flow/Min Cut algorithm to find an assignment solution which minimizes total execution and communication costs. His work was continued by V. M. Lo where task execution concurrency is improved by introducing interference costs to Stone’s model. The static assignment problem was formulated as state-space search. Heuristic-based solutions were proposed.

The queuing theoretic approach is also commonly used in static algorithms. Ni and Hwang derived a probabilistically optimal task assignment algorithm for a system showing both functional and throughput heterogeneity. However, their work was limited to exponential task service demand patterns. The neglect of task transfer overhead and communication de-
lays also limited the practical value of their algorithm.

It is now commonly agreed that despite higher run-time complexity, dynamic algorithms can potentially provide better performance than static algorithms.

The major components of a dynamic LD algorithm are:

1. the location policy, referring to the strategy used to pair up tasks senders and receivers;
2. the information policy, referring to the strategy used to disseminate load information among processing nodes; and
3. the transfer policy, referring to the strategy used to determine whether task transfer activities are needed by a node. We discuss these three policies below.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Concerning Node pairing.</td>
</tr>
<tr>
<td>Information</td>
<td>Concerning load information.</td>
</tr>
<tr>
<td>Transfer</td>
<td>Concerning Node activity</td>
</tr>
</tbody>
</table>

3 System Model

We assume an asynchronous distributed system with no shared memory support. Message passing is the only communication mechanism available between processing nodes. We also assume that the network is fault free and strongly connected. Messages are received in the order sent.

3.1 Modeling Processing Node Heterogeneity

The figure, we assume that nodes differ only in processing speeds, thus an application task can run in any node in the system. The heterogeneity of a system is characterized by the various processing speeds that the constituent nodes have. Nodes with identical (or very close) processing speeds are grouped into a type.
3.2 Task Class Specification

Application tasks are categorized into classes. Each task class is represented by a 4-tuple.

Service demand is the processing requirement of a class task. The unit service demand (one SD) represents the amount of computation that needs one second to complete if executed in a particular node type selected for calibration purpose. The time needed to complete one SD in other node types can be found easily with the relative processing speeds matrix.

\[
\int_{0}^{\infty} \frac{Y_o(t)dt}{t} = -\frac{1}{\pi} \left(\ln \frac{x}{2}\right)^2 - \frac{2\theta}{\pi}
\]

3.3 Processing Node Model

The processing node model is shown in figure. Each node consists of three tasks queues: Local Queue, Remote Queue, and Preemption Queue.

When a task arrives locally to a node, it joins the Local Queue at the tail and waits to be served by the CPU in FIFO discipline. Tasks in the Local Queue are candidates for remote transfer if the node becomes busy. A LD algorithm can select task(s) in the Local Queue for remote transfer in any arbitrary order it finds appropriate (i.e. not necessarily FCFS).

4 Conclusion

We proposed a class of adaptive load distribution algorithms, referred to as algorithms. Algorithms are designed for heterogeneous distributed systems showing throughput heterogeneity and subject to multiple task classes having varying service demands. The core components of algorithms are batch tasks assignments, the GR Protocol, and the symmetrically-initiated location policy.
You may want to learn more other LaTeX features by reading [1] and [2].

References
