Abstract: — In the present scenario the differences amongst collecting, transporting, storing and processing information are quickly disappearing. As our ability to gather, process and distribute information grows the demand for even more sophisticated information processing system grows much faster. Distributed Computing is an optimal solution to fulfill these ever-changing demands. Distributed Computing Systems (DCS) consist of loosely coupled processors which communicate with one another only by passing messages and which do not have common memory. The DCS are increasingly drawing attention, yet have a meaning that is not understood. The term ‘Distributed Computing System’ is used to describe system with multiple processors. However, the term has different meaning to different systems because processors can be interconnected in many ways for assorted reasons. All practical real-time scheduling algorithms in distributed computing systems present a trade-off between their computational complexity and performance. In real-time systems, tasks should be performed correctly and timely. Finding minimal schedule in distributed systems with real-time constraints is shown to be NP-hard. Although some optimal algorithms have been employed in uni-processor systems, they fail when they are applied in distributed computing environment. The practical scheduling algorithms in real-time systems have not deterministic response time. Deterministic timing behavior is an important parameter for system robustness analysis. The intrinsic uncertainty in dynamic real-time systems increases the difficulties of scheduling problem. In this paper we alleviate these difficulties, and shown that there is a need to develop a scheduling approach to arrange real-time periodic and non-periodic tasks in heterogeneous distributed systems. Static and dynamic optimal scheduling algorithms fail with non-critical overload. In contrast, a balanced approach is required to be developed which may balance task loads of the processors successfully while consider starvation prevention and fairness which cause higher priority tasks to have higher running probability.)

Keywords: EC, ITCC, DCS, CC

1. INTRODUCTION

1.1 Review of Previous Work: Before presenting the review of tasks allocation models, some definitions and assumptions that are used in the various models are given below:

1.2 Definitions and Assumptions
In Distributed Computing System, a distributed program is a set of computational tasks  \( T = \{t_1, t_2, t_3, \ldots, t_m\} \) which collectively form a common goal. An assignment of each task to processors can be defined by a function \( f \), from the set of tasks or modules to processors \( f: T \rightarrow P \). The goal is to find an optimal assignment that minimizes the cost function under certain constraints. The salient features of various tasks allocation models available in the literature are presented along with the definitions and assumptions considered therein.

a) Distributed Program Graph
An application which is partitioned into a set of interdependent tasks is represented by a graph consisting of a set of nodes connected by edges. The nodes correspond to tasks and the edges connecting the nodes indicate the data exchanged between the tasks. This graph is known as a distributed program graph.

b) Execution Cost (EC)
The execution Cost \( e_{ij} \) (where \( 1 \leq i \leq m, 1 \leq j \leq n \)) of each task \( t_i \) depends on the capabilities of the processor \( p_j \) to which it is assigned. If a task is not executable on some processor due to absence of some resources, the \( e_{ij} \) of that task on that processor is taken to be infinite.

c) Inter Task Communication Cost (ITCC)
The ITCC \( c_{ik} \) of the interacting tasks \( t_i \) and \( t_k \) is incurred due to the data units exchanged between them during the process of execution if they are not co-resident. Whenever groups of tasks or clusters are assigned to the same processor, the ITCC between them is zero. In most of the tasks allocation models, the ITCC is assumed to be proportional to the value of the data exchanged between the processors.
d) Completion Cost
Completion cost (CC) is the maximum value of the sum of EC and ITCC for all the processors in a DCS.

e) Load Balancing
Load balancing is a desirable requirement to assign each processor with equal load to prevent the situation in which some processors are heavily loaded while others are ideal.

1.3 Assumptions
Several assumptions have been made by the researchers to keep their algorithms reasonable in size while defining the problem.

- The program is assumed to be collection of “m” tasks to be executed on a set of “n” processors, which have different processing capabilities.
- A task may be portion of an executable code or a data file. The number of tasks to be allocated is more than the number of processors (m > n), as normally is the case in the real life distributed processing environment.
- It is assumed that the EC of a task on each processor is known, if a task is not executable on any of the processor due absence of some resources. The EC of that task on that processor is taken to be (∞) infinite.
- Once a task has completed execution on a processor, the processor stores the output data of the task in its local memory, if the data is needed by some another task being computed on the same processor, it reads the data from the local memory. The overhead incurred by this is negligible, so for all practical purposes taken as zero. Whenever a group of tasks is assigned to the same processor, the ITCC between them is zero.

The strategies of task allocation on a parallel of distributed system may be done in any of the following ways:

a) Static Allocation: In static allocation when a task is assigned to processor, it remains there while the characteristic of the computation change, a new assignment must be computed. The phrase “characteristics of the computation” means the ratios of the times that a program spends in various parts of the program. Thus, in a static allocation, one is interested in finding the assignment pattern that holds for the life time of a program, and result in the optimum value of the measure of effectiveness [3-10].

b) Dynamic Allocation: In order to make the best use of resources in a distributed system, it is essential to reassign modules or tasks dynamically during program execution, so as to the advantage of changes in the local reference patterns of the program. Although the dynamic allocation has potential performance advantages, Static allocation is easier to realize and less complex to operate [14-18].

A program that runs in a DCS is called a distributed program, and distributed programming is the process of writing such programs [1]. Distribution of resource in distributed system is seen as a way to improve system throughput and availability. An important resource for distributed system is user’s program that consists of a set of tasks. Response time is the important parameter for measuring performance for DCS. A distributed system is designed for solving some specific real time applications. The system is required to finish a certain task within a specific time limit [2]. Several other methods have been reported in the literature, such as, Integer programming [19, 21], Branch and bound technique [22-23], Matrix reduction technique [7], reliability evaluation to deal with various design and allocation issues in a DCS by [24-27].

2. IDENTIFICATION OF RESEARCH PROBLEM

When systems become large, the scheduling problems are not linear; there is often a qualitative change in complexity, and some things that are trivial to deal with in a network of only a few machines and principals, suddenly become a big deal. Over the last 35 years, computer science researchers have built many distributed systems and studied issues such as concurrency, failure recovery and scheduling. The theory is also supplemented by growing body of experience from industry, commerce and government. These issues are central to the design of effective and optimal systems, but are often handled rather badly.

Nowadays, usage of real-time distributed systems is dramatically increasing, unfortunately, less is known about how to schedule them. Optimal scheduling of real-time tasks on distributed systems is known to be computationally intractable for large task sets. Any practical scheduling algorithm in distributed systems presents a trade-off between performance and computational complexity. The performance of a scheduling algorithm is measured in terms of additional processor required to be added at a schedule without deadline violations as compared to optimal algorithm.

Scheduling real time systems involves allocation of resources and CPU-time to tasks in such a way that certain performance requirements are met. In real-time systems scheduling plays a more critical role than non-real-time systems because in these systems having the right answer too late is as bad as not having it at all. Having more computational complexity in these systems is a more serious issue than in real-time systems. Such a system must react to the requests within a fixed amount of time which is called deadline.
Schedulable of periodic, preemptive, soft real-time tasks on uniprocessor systems is well understood; simple and efficient algorithms are available and widely used. Nevertheless, for distributed systems these algorithms are not promising. Meeting the deadlines of real-time tasks in a distributed system requires a scheduling algorithm that determines, for each task in the system, on which processor it must be executed, and in which order with respect to the other tasks, it must start its execution.

In the global scheme, processors repeatedly execute the highest priority tasks available for execution. It has been shown that using this approach with common priority assignment schemes such as rate monotonic (RM) and earliest deadline first (EDF) can give rise to arbitrarily low processor utilization. One especially straightforward method to achieve this is the modeling of these parameters through fuzzy logic.

3. EXPECTED IMPACT ON ACADEMIC AND INDUSTRY

With the advancement of modern society, basic essential services (utilities) are commonly provided such that everyone can easily obtain access to them. Today, utility services, such as water, electricity, gas, and telephony are deemed necessary for fulfilling daily life routines. These utility services are accessed so frequently that they need to be available whenever the consumer requires them at any time. Consumers are then able to pay service providers based on their usage of these utility services. Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered. Several computing paradigms have promised to deliver this utility computing (Distributed computing) and from which businesses and users are able to access applications from anywhere in the world on demand. Thus, the computing world is rapidly transforming towards developing advancement in distributed computing for millions to consume as a service, rather than to run on their individual computers.

Homogeneous Distributed computing is an extension of paradigm wherein the capabilities of business applications are exposed as sophisticated services that can be accessed over a network. Distributed service providers are incentivized by the profits to be made by charging consumers for accessing these services. Consumers, such as enterprises, are attracted by the opportunity for reducing or eliminating costs associated with “in-house” provision of these services. In addition, enterprise service consumers with global operations require faster response time, and thus save time by distributing workload requests to multiple Clouds in various locations at the same time. This creates the need for establishing a computing atmosphere for dynamically interconnecting and provisioning Clouds from multiple domains within and across enterprises. There are many challenges involved in creating such Clouds and Cloud interconnections.

Distributed computing is by no means a new idea: many concepts that underlie today’s grid systems predate even the Internet. Back in the mid-1960s, for example, Fernando Corbato, the father of time-sharing operating systems, described the then revolutionary Multics system as a “computing utility.” The banking and airline industries have run sophisticated distributed systems for decades. We, however, approached the problem from a different perspective, one defined by the needs of scientific research communities. In our experience, the often-extreme requirements and only partially controlled chaos of scientific investigation can be strong drivers for innovation. It seems no coincidence that the World Wide Web was invented by Tim Berners-Lee, a computer scientist whose work with high-energy physicists inspired him to create a universal system for sharing information.

Meanwhile Distributed computing researchers are tackling the next set of challenges: How can we manage large, distributed infrastructures so that they deliver reliable service in the face of failures? How can we enable users to exploit the availability of on-demand resources and services? How must grid concepts and technologies evolve as the number and power of computers rise by orders of magnitude? The answers will emerge from both research and practical experience and will surely draw on ideas being pioneered in the related fields of autonomic, ubiquitous and peer-to-peer computing [see “The Worldwide Computer,” by John Kubiatołowicz and David P. Anderson; Scientific American, March 2002]. These disciplines are all converging on the same vision of the computing environment of tomorrow.

4. TASK ASSIGNMENT PROBLEM AND DEFINITION

The processors are heterogeneous, i.e., the execution cost of a task depends on the processor on which it is executed. Let P be the set of n processors in the heterogeneous computing system, T be the set of m tasks to be assigned to the processors, where it is assumed that M>N, assign (allocate) each of the M modules to one of the N processors in such a manner that the IPC time is minimized and the processing load is balanced. ETC = \{x_{ij}\} m x n be the expected time to compute matrix where x_{ij} denotes the execution cost of task i on processor p, and G = (T ,E) be the TIG, where E is the set of edges representing the communication between tasks. Each edge \((i, j ) \in E\) is associated with a communication cost c_{ij}, which incurs only when tasks i and j are assigned to different processors. The processors are heterogeneous in the sense that there is no special structure in the ETC matrix.
4.1 Parameter of different load balancing algorithm

Execution:

a) Cost: The execution cost $ec_{ij}$ is the amount of the work to be performed by the executing task $t_i$ on the processor $p_j$ during the $k^{th}$ Phase. Where $1 \leq i \leq m, 1 \leq j \leq n$ of each task $t_i$ depends on the processor $p_j$ to which it is assigned and the work to be performed by each of tasks of that processor $p_j$. The processing execution cost of the tasks on all the processors is given in the form of Execution Cost Matrix (ECM) of order m x n. The Execution Cost of a given assignment on each processor is calculated by the following equation.

$$RT(A_{alloc}) = \max_{1 \leq j \leq n} \{PEC(A_{alloc})_{j} + IPC(A_{alloc})_{j}\}$$

d) Residence Cost
Residence cost $r_{kj}$ (where $1 \leq i \leq M$ & $1 \leq j \leq N$) is the amount of time spent by the residing task (other than executing task) $t_i$ on the processor $p_i$ in $k_{th}$ phase. These costs may come from the use of storage.

Assign the residing tasks $t_i (g = 1, 2... M, g =/= s)$ to processors $p_h$ at which time is Minimum say TRC $(g) k$

Calculate:

$$TRC_k = \sum_{i=1}^{M} TRC(i)_{k}$$

e) Reallocation cost
Reallocation cost $rel_{ik}$ is the cost of reassigning the $i_{th}$ task from one processor to another processor at the end of $k_{th}$ phase. When an allocated task is shifted from one processor to another processor during the next phase then reallocation cost is mentioned at the end of each phase. An amount of relocation cost for reassigning each task from one processor to the others at the end of the phases.

f) Data Transfer Rate
Data Transfer Rate $d_{ik}$ is per unit cost i.e. data exchanged between tasks $t_i$ and $t_k$ during the program execution. The speeds with which data can be transmit from one device to another.

$$PEC(f) = \sum_{j=1}^{n} ec_{ij}x_{ij}, i = 1, 2, 3, 4, 5, 6, ... m$$

5. COMPARATIVE ANALYSIS OF LOAD BALANCING ALGORITHMS based on different parameter

Based on different parameter we Collect Analysis of Different load balancing algorithm in distributed Computing system which is shown in table form [37]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ADAPTIVE LOAD SHARING ALGORITHM</th>
<th>MAXIMALLY LINKED MODULES ALGORITHM</th>
<th>NEAREST-NEIGHBOR LOAD BALANCING</th>
<th>MAEKAWA'S ALGORITHM</th>
<th>A* ALGORITHM</th>
<th>PARALLEL GENETIC ALGORITHM</th>
<th>DETERMINISTIC MODEL GENETIC ALGORITHM</th>
<th>MULTIHEURISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
<td>Static</td>
<td>Static</td>
<td>Static</td>
<td>Static</td>
<td>Static</td>
<td>dynamic</td>
<td>dynamic</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>
6. CONCLUSION

Load balancing in distributed systems is the most thrust area in research today as the demand of heterogeneous computing due to the wide use of internet. In this paper it is clear that based on various parameters every load balancing algorithm behaves in different manner. More efficient load balancing algorithm more is the performance of the computing system.

We have enumerated the facilities provided by load balancing algorithms. Finally, from given comparative table of different load balancing algorithm, there exists no absolutely perfect balancing algorithm but one can use depending on the need. The comparative study not only provides an insight view of the load balancing algorithms, but also offers practical guidelines to researchers in designing efficient load balancing algorithms for distributed computing systems:

REFERENCES


[33] Sagar Gulati, K. Bhatia and P. K. Yadav “Execution Time and Failure Rate based model for Reliability Optimization in Distributed Systems” IRACST.

[34] Wireless Communications (IJCNCW), ISSN: 2250-3501 Vol. 4, No. 5, October 2014.


