Distributed Computing System for Parallel Processing

Pallavi S. Shendekar
Department of Information Technology
Sipna College of Engineering & Technology
Amravati, India

Vijay S. Gulhane
Department of Information Technology
Sipna College of Engineering & Technology
Amravati, India

ABSTRACT
Distributed computing is a form of parallel computing, but parallel computing is most commonly used to describe program parts running concurrently on multiple processors in the same computer. Both types of processing require dividing a program into parts that can run simultaneously, but distributed programs often must deal with assorted environments, network links of varying latencies, and unpredictable failures in the network or the computers. In distributed computing a program is divide into parts that run simultaneously on multiple computers communicating over a network. There are many different types of distributed computing systems and many challenges to overcome in successfully designing one. The main goal of this paper is to connect users and resources in a transparent, open, and scalable way. Ideally this arrangement is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems.

Keywords
Distributed computing, parallel processing, data conversion, distributed programming.

1. INTRODUCTION
The word distributed in terms such as "distributed system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area. The system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel.[12] Distributed computing deals with hardware and software systems containing more than one processing element or storage element, concurrent processes, or multiple programs, running under a loosely or tightly controlled administration. In distributed computing a program is divide into parts that run simultaneously on multiple computers communicating over a network. In parallel computing, all processors may have access to a shared memory to exchange information between processors whereas in distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors. The application of several processors to a single task is an older idea with a relatively large literature. The advent of very large-scale integrated technology has made testing the idea realistic, and the fact that single processor systems are impending their maximum performance level has made it crucial. We shall show, however, that victorious use of parallel processing imposes rigorous performance necessities on algorithms, software, and architecture. The so-called asynchronous systems which use a few tightly coupled high-speed processors are a natural development from high-speed single-processor systems. In fact, systems with two to four processors will soon be available (for example, the Cray X-MP, the Cray-2, and the Control Data System 2XX).

To estimate the speedup of a tightly coupled system on a single application, we use a model of parallel computation introduced by Ware. We define a as the fraction of work in the application that can be processed in parallel. Then we make a simplifying assumption of a two-state machine; that is, at any instant either all processors are operating or only one processor is operating. Consider the condition user having 10000 document to process and each having large data in this case project need to employee the system which will transfer processing over the network system and save output on the server or main system. So the proposed system will aimed at parallel processing of provided task.

In Distributed Computing approach, it is followed to assign a job to a processor if it is idle. The focus is now on how to optimize re-sources to decrease the energy consumption by volumes of computing equipments to deal with green and sustainability issues[10]. Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely-coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

2. LITERATURE SURVEY
Various algorithm and models have been proposed, mostly heuristic in nature, as the optimal solution often requires future knowledge and is computationally intensive. The most widely approach for studying DLB algorithms is analytic modeling and simulation. For analytic modeling, the computer system is modeled as a queuing network with job arrivals and their resource consumptions following certain probabilistic patterns. Queuing network solution techniques are used to compute performance measures [2, 9, 8, 7] Due to limitations of the solution techniques, simulation is often resorted to for approximate solutions [5, 4]. Some of the-source-initiated DLB algorithms are by Eager. [8, 7, 6]

Task partitioning is proposed by Deelman et al[11]. It partitions a workflow into multiple sub-workflows which are executed sequentially. Rather than mapping the entire workflow on Grids, allocates resources to tasks in one sub-workflow at a time. Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed

onto a circuit board or made up of loosely coupled devices and cables. At a higher level it is necessary to interconnect processes running on those CPUs with some sort of communication system.

2.1 DISTRIBUTED PROGRAMMING ARCHITECTURES

Distributed programming typically falls into one of several basic architectures or categories: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, or tight coupling [1].

2.1.1 Client-server

Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.

2.1.2 3-tier architecture

Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.

2.1.3 N-tier architecture

N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.

2.1.4 Tightly coupled (clustered)

Tightly coupled refers typically to a cluster of machines that closely work together, running a shared process in parallel. The task is subdivided in parts that are made individually by each one and then put back together to make the final result.

2.1.5Peer-to-peer

Peer-to-peer is an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.

2.1.6 Space based

Space based refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

2.2 Different Transparency

Our goals in designing a distributed file system are to present certain degrees of transparency to the user and the system:

2.2.1 Access transparency

Clients are unaware that files are distributed and can access them in the same way as local files are accessed.

2.2.2 Location transparency

A consistent name space exists encompassing local as well as remote files. The name of a file does not give it location.

2.2.3 Concurrency transparency

All clients have the same view of the state of the file system.This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.

2.2.4 Failure transparency

The client and client programs should operate correctly after a server failure.

2.2.5 Heterogeneity

File service should be provided across different hardware and operating system platforms.

2.2.6 Scalability

The file system should work well in small environment (1 machine, a dozen machines) and also scale gracefully to huge ones (hundreds through tens of thousands of systems).

2.2.7 Replication transparency

To support scalability, we may wish to replicate files across multiple servers. Clients should be unaware of this.

2.2.8 Migration transparency

Files should be able to move around without the client's knowledge.

2.2.9 Support fine-grained distribution of data

To optimize performance, we may wish to locate individual objects near the processes that use them.

2.2.10 Tolerance for network partitioning

The entire network or certain segments of it may be unavailable to a client during certain periods (e.g. disconnected operation of a laptop). The file system should be tolerant of this.

3. PROPOSED ARCHITECTURES

In the proposed architectures the main factors are the designing the distributed system and parallel processing through a specific problem domain. Here the problem domain is known and well defined, the environment in which the system run is also well defined. Basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a master/slave relationship. Alternatively,"database centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.

3.1 Working Modules

Figure1 shows the basic block diagram of the paper. Actual task processing needs a series of steps to be performed. These series of processes are simultaneously executed on different client machines. Basically here we are distributing the no. of files on to the network through the shared database.

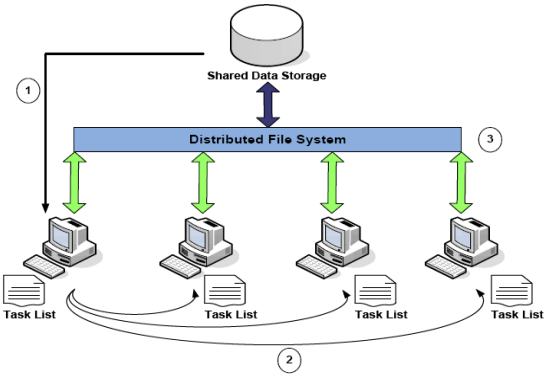

Fig. 1- System Architecture

3.1.1 Master / Slave System

Parallel processing system built using server / client technology. Where master server act as a process manager system. In this system whenever network initialize or the first system user started in the network will check for active server in the network if no server running found then it will become host or master server and other were become slave system. This feature can be dynamic or static which means user can disable or enable this feature.

3.1.2 Task (Conversion)

This is the main input to the parallel processing system. First user need to define the task to perform the specified operation. In this system it will consider a task of batch file format conversion. Data can be passed to the client to process or it can be placed on shared data storage location from where all clients will fetch data to process.

3.1.3 Task Assigning

Once the user provides input task on master server then master will analyze the task and divide it in to proper task and create its task list for client. Once all clients get the task list to process it will start processing. As proposed system will work on shared data storage it will reduce network processing and network traffic by removing data transfer processing

3.1.4 Parallel Processing

After master distributes the task over the distributed network all clients will process simultaneously and send acknowledgement to the master server. Master server will also process the task and at the same time will check for the client processing status and monitor it on the screen.

3.1.5 Process Failure Detection System

The system will also manage failure of the client system at run time. Consider a condition if any of the client get failed due to any reason the remaining task should be processed by other system in the network. Master server will take care of this. It will continuously get acknowledgment from client time to time after each task completion. Once any client's connection get closed server will check work remain by specific client and then again divide this task and pass it to other client and client will process it.

3.1.6 Distributed File System

System sends data to client for processing but it will increase system overhead. So in this paper data to processed will kept on shared storage and accessed using distributed file system so that network protocol processing can be reduced.

The main task is to start the server on specific port in listen mode now server is ready to get the request from the client. Now client can make request to the server by providing server address and the server port detail. Client send connect request. Server will get the connection request same as we get the ring on mobile for connection. After this server can accept or reject the connection and server accepting the request, a connection link gets established between server and client. Now server can send the data to the client and client get the data arrival ACK, after this client can read the data .Same thing happened with server and communication goes on. Finally any one of both can close the connection.

4. REQUIREMENT ANALYSIS

Operating System: Windows XP
Development Tool: C# (.Net)
Database: SQL Server 2005

5. CONCLUSION

The proposed system is best for batch or mass execution. This parallel processing distributed computing Model can reduce over-heads and it makes the proper utilization of multiple systems rather than implementing supercomputing processor. This system reduces the risk of failure as it can use normal lower configuration PC system to complete the task and even input task is not dependent on the single system. But If not planned properly, a distributed system can decrease the overall reliability of computations if the unavailability of a node can cause disruption of the other nodes.

6. REFERENCES

[1] Gupta R., Chaube A.R., Singh S. (2011) International Journal of Advanced Research in Computer Science and Software Engineering,.

[2] Wang Y. and Morris R. (1985) *IEEE Trans. Computing*, 34(3), 204-217.

[3] Stone H.S. (1978) *IEEE Trans. Software Eng.*, 4(3).

[4] Stone H.S. (1977) *IEEE Trans of Software Engineering*, 3(1), 95-93.

[5] Miron Livny, Myron Melman (1982) *The Computer Network Performance Symposium*, 47-55.

[6] Hsu C.H. and Liu J.W. (1986) The 6th International Confer-ence on Distributed Computing Systems, 216-223.

[7] Derek L. Eager, Edward D. Lazowska, John Zahorjan (1986) *IEEE Transactions on Software Engineering*, 12(5), 662-675.

[8] Eager D.L., Lazowska E.D. and Zahorjan J. (1986) *Perfor-mance Evaluation*, 6(1), 53-68.

[9] Chow Y.C. and Kohler W. (1979) *IEEE Transactions on Com-puters*, 28, 334-361.

[10] Joshi E. International Journal of Computer Applications, 1(18), 0975 - 8887.

[11] Deelman E. et al. (2004) *European Across Grids Conference*, 11-20 .

[12] Lynch,Nancy A (1996),Distributed Algorithm, Morgan Kaufmann, ISBN 978-1-58488-564-1.