KARPAGAM ACADEMY OF HIGHER EDUCATION
Coimbatore-641021.
(For the candidates admitted from 2017 onwards)
Department of CS,CA & IT

SUBJECT : GRID COMPUTING
SEMESTER : II
SUBJECT CODE : 17CSP205B **CLASS : I M.Sc.CS**

SCOPE

The main objective of the course is to portray the recent trends in the field of Grid computing and creation and management of Internet-based utility computing infrastructure**.**

OBJECTIVES

- Provide a good understanding of the concepts standards and protocols in Grid computing
- To perform analysis design and implementation of ARC grid computing model.

UNIT- I

Introduction: Cluster to Grid Computing – Cluster Computing Models – Grid Models – Mobile Grid Models – Applications. Parset: System-independent Parallel Programming on Distributed Systems –introduction – Semantics of the Parset Construct – Expressing Parallelism through Parsets – Implementing Parsets on a Loosely Coupled Distributed System

UNIT- II

Anonymous Remote Computing Model: Issues in Parallel Computing on Interconnected Workstations – Existing Distributed Programming Approaches – The ARC Model of Computation – The Two-tired ARC Language Constructs – Implementation. Integrating Task Parallelism with Data Parallelism: A Model for Integrating Task Parallelism into Data Parallel Programming Platforms – Integration of the Model into ARC – Design and Implementation – Applications - Performance Analysis

UNIT- III

Anonymous Remote Computing and Communication Model: Location – Independent Inter-task Communication with DP – DP Model of Iterative Grid Computations – Design and Implementation of Distributed Pipes. Parallel Programming Model on CORBA: Notion of Concurrency – System Support –Implementation and Performance

UNIT- IV

Sneha-Samuham Grid Computing Model: A Parallel Computing Model over Grids – Design and Implementation – Performance studies. Introducing Mobility into Anonymous Remote Computing and Communication Model – Issues in Mobile clusters and Parallel Computing on Mobile Clusters – Moset Overview – Computation Model – Implementation and Performance

UNIT- V

Distributed Simulated Annealing Algorithms for Job Shop Scheduling - Implementation. Parallel Simulated Annealing Algorithms - Simulated Annealing (SA) Technique – Clustering Algorithm for Simulated Annealing (SA) – Combination of Genetic Algorithm and Simulated Annealing (SA) Algorithm - Implementation. Epilogue : DOS Grid: Vision of Mobile Grids - Mobile Grid Monitoring System – Healthcare Application Scenario.

SUGGESTED READINGS

TEXT BOOK

1. Janakiram, D. (2009). Grid Computing – A Research Monograph. New Delhi: TataMcGraw Hill Publishing Company Limited.

REFERENCES

1. Frederic Magoules.(2009). Fundamentals of Grid Computing. Taylor and Francis.
2. Prabhu, C.S.R. (2008). Grid and Cluster Computing New Delhi:Prentice Hall of India
3. Jie Pan. 2009. Introduction to Grid Computing. Taylor and Francis. CRC Press.

WEB SITES

1. http://cseweb.ucsd.edu/classes/sp00/cse225/notes/fran/introweb.html
2. http://www.wisegeek.com/what-is-grid-computing.htm
3. http://www.cs.kent.edu/~farrell/grid06/lectures/index.html

ESE MARKS ALLOCATION

1	**Section A** 20 x 1 = 20	20
2	**Section B** 5 x 6 = 30	30
3	**Section A** 1 x 10 = 10	10
	TOTAL	60

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act, 1956)

Coimbatore – 641 021.

LECTURE PLAN
DEPARTMENT OF COMPUTER SCIENCE

STAFF NAME: Dr.S.VENI
SUBJECT NAME: GRID COMPUTING SUB.CODE:17CSP205B
SEMESTER: II CLASS: I M.Sc (CS)

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
		UNIT-I	
1	1	Introduction - Cluster to Grid Computing	T1:1
2	1	Cluster Computing Models	T1:2
3	1	Grid Models	T1:3
4	1	Mobile Grid Models	T1:3
5	1	Applications	T1:5
6	1	Parset	T1:7
7	1	System Independent Parallel Programming on Distributed Systems-Introduction	T1:8
8	1	(Contd..)System Independent Parallel Programming on Distributed Systems-Introduction	T1:8
9	1	Semantics of the Parset Construct	T1:10, w1
10	1	Expressing Parallelisim through Parsets	T1:17
11	1	Implementing Parset on a loosely Coupled Distributed system	T1:33
12	1	Recapitulation and Discussion of Important Questions	
	Total No of Hours Planned For Unit 1=12		

UNIT-II			
1	1	Anonymous Remote Computing Model - Issues in Parallel Computing on Interconnected workstations	T1:34
2	1	Existing Distributed Programming Approaches	T1:40
3	1	The ARC Model of Computation	T1:43,w1
4	1	The Two-Tiered ARC Language Constructs	T1:52
5	1	Implementation	T1:65
6	1	Integrating Task Parallelism with Data Parallelism	T1:80
7	1	A Model for Integrating Task Parallelism into Data Parallel Programming Platforms	T1:85
8	1	Integration of the Model into ARC	T1:87
9	1	Design and Implementation	R2:109
10	1	Application	T1:101
11	1	Performance Analysis	T1:113
12	1	Recapitulation and Discussion of Important Questions	
	Total No of Hours Planned For Unit II=12		
UNIT-III			
1	1	Anonymous Remote Computing and Communication Model	T1:119,w2
2	1	Location - Independent Inter Task Communication with DP	T1:121
3	1	DP Model of Interactive Grid Computation	T1:124
4	1	Design and Implementation of Distributed Pipes	T1:129
5	1	(Contd..)Design and Implementation of Distributed Pipes	T1:129
6	1	Parallel Programming Model on CORBA	T1:155, R1:143
7	1	(Contd..) Parallel Programming model on CORBA	T1:155,

8	1	Notion of Concurrency	T1:160
9	1	System Support	T1:163
10	1	Implementation	T1:191
11	1	Performance	T1:194
12	1	Recapitulation and Discussion of Important Questions	
	Total No of Hours Planned For Unit III=12		
		UNIT-IV	
1	1	Sneha-Samuham Grid Computing Model	T1:210
2	1	A Parallel Computing Model over Grids	T1:212,w2
3	1	Design and Implementation	T1:218
4	1	Performance Studies	T1:221
5	1	Introduction Mobility into Anonymous Remote Computing and Communication Model	T1:230
6	1	(Contd..) Introduction Mobility into Anonymous Remote Computing and Communication Model	T1:232
7	1	Issues in Mobile Cluster and Parallel Computing on Mobile Clusters	T1:233
8	1	(Contd..) Issues in Mobile Cluster and Parallel Computing on Mobile Clusters	T1:235
9	1	Moset Overview	T1:237, R2 : 202
10	1	Computational Model	T1:240
11	1	Sneha-Samuham Grid Computing Model	T1:245
12	1	Recapitulation And Discussion Of Important Questions	
	Total No of Hours Planned For Unit IV=12		
		UNIT-V	
1	1	Distributed Simulated Analyzing Algorithms For Scheduling	T1:255

2	1	Distributed Algorithm For Job Shop Scheduling	T1:260,R2:231
3	1	Implementation Results And A Observation	
4	1	Parallel Simulated Amealing Algorithms	T1:278
5	1	Simulated Annealing (SA) Technique, Clustering Algorithm For Simulated Annealing	T1:279,W2
6	1	Combination Of Genetic Algorithm And Simulated Annealing Algorithm Implementation Of Algorithms	T1:281
7	1	Epilogue: Dos Grid - Vision Of Mobile Grid	T1:292
8	1	Dos Grid, Mobile Grid Monitoring System, Healthcare Applications Scenario	T1:294
9	1	Recapitulation and Discussion of important Questions	
10	1	Discussion of Previous ESE Question Papers.	
11	1	Discussion of Previous ESE Question Papers.	
12	1	Discussion of Previous ESE Question Papers.	
	Total No of Hours Planned for Unit V=12		
Total Planned Hours	**60**		

TEXT BOOK

1. Janakiram, D. 2009. Grid Computing – A Research Monograph. New Delhi: TataMcGraw Hill Publishing Company Limited.

REFERENCES

1. Frederic Magoules.(2009). Fundamentals of Grid Computing. Taylor and Francis.
2. Prabhu, C.S.R. (2008). Grid and Cluster Computing New Delhi:Prentice Hall of India
3. Jie Pan. 2009. Introduction to Grid Computing. Taylor and Francis. CRC Press.

WEBSITES

1. http://cseweb.ucsd.edu/classes/sp00/cse225/notes/fran/introweb.html
2. http://www.wisegeek.com/what-is-grid-computing.htm
3. http://www.cs.kent.edu/~farrell/grid06/lectures/index.html

UNIT-I

SYLLABUS

Introduction: Cluster to Grid Computing – Cluster Computing Models – Grid Models – Mobile Grid Models – Applications. Parset: System-independent Parallel Programming on Distributed Systems –introduction – Semantics of the Parset Construct – Expressing Parallelism through Parsets – Implementing Parsets on a Loosely Coupled Distributed System

INTRODUCTION

Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach a common goal. The **grid** can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Although a grid can be dedicated to a specialized application, it is more common that a single grid will be used for a variety of different purposes. Grids are often constructed with the aid of general-purpose grid software libraries known as middleware.

Grid size can vary by a considerable amount. Grids are a form of distributed computing whereby a "super virtual computer" is composed of many networked loosely coupled computers acting together to perform very large tasks. Furthermore, "distributed" or "grid" computing, in general, is a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network

interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

CLUSTER COMPUTING MODELS

The Anonymous remote computing (ARC) paradigm is proposed to address the issues specific to parallel programming on workstation systems. Arc differs from the conventional communicating process model as its treats a program as one single entity consisting of several loosely coupled remote instruction blocks instead of treating it as a collection of process. The Arc approach results in transparency in both distribution and heterogeneity. At the same time, it provides fault tolerance and load adaptability to parallel programs on workstations. Arc is developed in a two-tiered architecture consisting of high level language constructs and low level Arc primitives.

Arc is pure data parallel approach and assumes that there is no inter-task communication. Programs which are tied up with specific machines will not be resilient to the changing conditions of a netwok of workstations(NOW).The distributed pipes(DP) models enables location independent inter-communication among process across machines. This approach enables the migration of communicating parallel tasks according to runtime conditions. A transparent programming model for a parallel solution to iterative grid computations (IGC) using DP is also proposed.

An engineering problem, namely, the steady state Equilibrium problem is studied over the model. the performance analysis shows the speedup due to parallel execution and scaled down memory requirements .Both Arc and ARCC(Anonymous remote computing and communication) use low level network programming for implementation .In order to raise the abstraction level the use of middleware for cluster computing was explored and we built P-COBRA .Existing models for parallel programming over

common object request broker architecture(COBRA) do not address issues specific to parallel programming over NOWs. . P- CORBA, a model for parallel programming over CORBA addresses these issues. The transmission and distribution of computing power of a NOW are facilitated by P-CORBA. The main contribution of the work is to bring a notion of concurrency into CORBA. . A detailed performance comparison of the model is made with a widely used parallel programming tool, namely Message-Passing Interface (MPI).

GRID MODELS

The Sneha-Samuham grid computing model is an attempt to provide an adaptive parallel computing support over computational grids for solving computation-intensive applications. Unlike other grid computing models, Sneha-Samuham provides task-splitting capabilities, wherein the given task is split according to the computational capabilities of the nodes participating in the computation. Aggregating resources in Sneha-Samuham is as simple as making friends by using an instant messenger. The runtime environment of Sneha-Samuham executes a task efficiently by sharing the task among the participating machines, depending on their computation capability, which is measured by using a Grid Computation Capacity Factor (GCCF). The Sneha-Samuham grid computing model has been implemented over a nationwide grid. The model has been evaluated by using neutron shielding simulation application. The results show that it achieves almost linear speed-up. A comparison with MPI shows that Sneha-Samuham outperforms MPI, especially when machines with varying GCCFs comprise the grid. Currently, scientific applications that are purely data parallel and coarse-grained can benefit from Sneha-Samuham.

MOBILE GRID MODELS

Advances of technology in terms of cellular communications and the increasing computing power of the mobile systems have made it convenient for people to use mobile systems more than static systems. This has seen the greater use of mobile devices in personal and distributed computing, thus making the computing power ubiquitous.

The combination of wireless communication and cluster computing in many applications has led to the integration of these two technologies to emerge as a Mobile Cluster Computing (MCC) paradigm. This has made parallel computing feasible on mobile clusters, by making use of the idle processing power of the static and mobile nodes that form the cluster. In order to realize such a system for parallel computing, various issues such as connectivity, architecture and operating system heterogeneities, timeliness issues, load fluctuations in machines, machine availability variations, and failures in workstations and network connectivities need to be handled.

Moset, an Anonymous Remote Mobile Cluster Computing (ARMCC) paradigm is being proposed to handle these issues. Moset provides transparency to the mobility of nodes, distribution of computing resources, and to heterogeneity of wired and wireless networks. The model has been verified and validated by implementing a distributed image rendering algorithm over a simulated mobile cluster model.

Advancement in technology has enabled mobile devices to become information and service providers by complementing or replacing static hosts. Such mobile resources are highly essential

for on-field applications that require advanced collaboration and computing. This creates a need for the merging of mobile and grid technologies, leading to a mobile grid paradigm. The key idea in building the mobile grid is to integrate the computational, data and service grids. Thus a mobile device from anywhere and at any time, can harness computing power, and the required resources and services seamlessly. Simultaneously, the device could also be providing location-sensitive data to the grid. We have designed and prototyped a middleware for a mobile grid that transparently manages and bridges the requirement of the mobile users and the actual providers.

APPLICATIONS

Simulated Annealing (SA) has been considered a good tool for complex non-linear optimization problems. The technique has been widely applied to a variety of problems. However, a major disadvantage of the technique is that it is extremely slow and hence unsuitable for complex optimization problems such as scheduling. There are many attempts to develop parallel versions of the algorithm. Many of these algorithms are problem-dependent in nature. We present two general algorithms for SA. The algorithms have been applied to the Job Shop Scheduling Problem (JSS) and the Traveling Salesman Problem (TSP), and it has been observed that it is possible to achieve super-linear speed-ups using the algorithm.

Job Shop Scheduling USS) belongs to the class of NP-hard problems. There are a number of algorithms in the literature for finding near optimal solution for the JSS problem. Many of these algorithms exploit problem specific information and hence, are less general. However, simulated annealing algorithm for JSS is general and produces

better results when compared to other algorithms. But one of the main drawbacks is that the execution time is high. This makes the algorithm inapplicable to large scale problems. One possible approach is to develop distributed algorithms for JSS using simulated annealing. Three different algorithms have been developed, namely Temperature Modifier, Locking Edges and Modified Locking Edges algorithms.

DOS Grid: Vision of Mobile Grids

We finally present our grand vision of building a mobile grid as an integration of computation, data and service grids. The mobile grid enables any device to access or provide the required computing power, information or other services from or to the grid. The data considered also includes lower-level data collected aggregated from the sensor devices. The mobile grid requires monitoring data for a variety of tasks such as fault detection performance analysis, performance tuning, performance prediction, and scheduling. The requirements and essential services are outlined that must be provided by a mobile grid monitoring system. We also present its realization as a peer-to-peer overlay over a distributed shared object space. The proposed mobile grid model visualizes the architecture as a distributed shared object space, wherein all the participating mobile devices are modelled as surrogate objects which reside on the wired network. We illustrate the mobile grid through a mobile health care application.

PARSET: SYSTEM-INDEPENDENT PARALLEL PROGRAMMING ON DISTRIBUTED SYSTEMS

During the last decade, a significant amount of interest has been shown in the development of parallel programs on loosely coupled distributed systems. An example of such a system is a set of powerful multi-programmed workstations connected through a

local area network. Several mechanisms for performing inter-process communication, synchronization, mutual exclusion and remote accession have been proposed to cope up with the challenges arising out of the distributed nature of these systems. Some examples of these mechanisms are client-server communication, message-passing multiple programs, and Remote Procedure Calls (RPC).

A programmer can start his processes on heavily loaded nodes, thereby causing severe load imbalances, resulting in underutilization of the network. This can also adversely affect the performance of other programs running in the network. A key property of distributed systems is that they are open-ended. Various system parameters like node configuration and node availability keep changing over a period of time. In such cases, the programs need to adapt themselves dynamically to the changing system configurations.

Thus, there is no clear separation between the programmer's concerns and the system's concerns in the present approaches to distributed programming. It has become necessary to provide high-level language constructs which can do this task. These constructs should be provided with adequate Iow-level runtime support which can achieve the separation between the system's concerns and the programmer's concerns. These language constructs can be provided as extensions to existing programming languages. The programmer can specify his coarse parallel blocks within his program by making use of these high-level language constructs. These language constructs are suitably translated and

handled by the low-level system mechanisms. The various advantages of using such constructs are listed below.

Advantages from the System's Point of View

If the selection of nodes for performing computation is made at the programming level, the programmer can write programs which can generate heavy load imbalances in the system. For example, P4 gives the choice of node selection to the user. At the time of node selection, the user may not be in a position to predict the actual load on the selected nodes at the time of execution.

Also, the programs may not make use of dynamically changing loads on the machines due to its rigid process configuration. For example, a program in PVM may create a fixed number of processes. In such a case, the program will not be able to utilize the additional capacity in the system if some nodes become lightly loaded when the program actually starts executing.

With appropriate high-level language constructs, the programmer can only express the willingness of parallel execution. With this, the programs need not be modeled as a pre-configured collection of processes. This provides maximum flexibility to the system to make effective use of the available resources.

Advantages from the Programmer's Point of View

(i) The user can be relieved of the burden of creating processes and performing explicit communication and synchronization among them.

(ii) The number of available nodes and their interconnection pattern vary from one distributed system to the other, and

also from time to time in a single distributed system. The load on the machines frequently keeps changing. In such a case, if a program is written as a collection of a fixed number of processes, it cannot make use of the dynamically changing loads in the system. By using the high-level language constructs, the programs can be written in a system-independent fashion.

(iii) Several programming errors, which occur during programming inter-process communication, synchronization, termination etc., can be avoided by programming with such language constructs.

In this chapter, we present a language construct called parset and the low-level runtime support for implementing it. Parsets can be used for expressing coarse grain parallelism on distributed systems. The parset construct consists of a data structure and a set of functions which operate on this data structure. The construct has been specially designed for capturing several kinds of coarse grain parallelism occurring in distributed systems. The use of parsets relieves the programmer of the burden of handling the remote processes, inter-process communication, remote procedure calls, etc. A low-level distributed parset kernel creates sub-tasks, locates suitable remote nodes, and gets the code executed on the remote nodes. This makes the programs that are written using parsets, scalable over varying system parameters.· Thus, parsets draw a clear distinction between the system's concerns and the programmer's concerns.

SEMANTICS OF THE PARSET CONSTRUCT

In this section, we first describe the parset data structure with the basic operations which manipulate the data structure. When functions receive parsets as their arguments, they derive special meanings. The function semantics on parsets are explained

subsequently. Finally in this section, a special case of parset called indexparset is described.

The Parset Data Structure

Parset is a set type of data structure. The elements of a parset can be:

- Basic data types,
- Untyped,
- Functions.

When the elements of a parset belong to a basic data type, it is called a simple parset. When the elements are untyped, it is an untyped parset. A function parset has functions as its elements. Simple parsets can be used to express the SPMD kind of parallelism. This is discussed in detail in Section 2.3.1. Untyped parsets find applications in expressing MPMD kind of parallelism using polymorphic functions [4]. Function parsets are the most general ones, and by using them, it is possible to express the MPMD parallelism in a general way.

A parset is kept logically ordered on the basis of the entry of its elements on a first-come-first-served basis. Cardinality is an attribute associated with a parset. The cardinality of an empty parset is zero. A typed parset declares the type of the elements held by the parset. For example,

parset P of int;

declares a parset P of elements of type integer.

The operations that can be performed on parsets are insert(), flush(), get(), delete(), getcard() and setcard(), The functions that can be executed on parsets have their arguments tagged as RO (read-only), RW (read-write) or WO (write-only). This scheme is very similar to the Ada language approach which places the reserved keywords IN, OUT, and INOUT before the arguments.
The semantics of these tags are as follows:

WO: The argument is only modified inside 'the function but not read.

RO The argument is passed to the function only for reading.

RW : The argument is read as well as written to inside the function ..

When functions taking parsets as arguments are executed, the arguments are locked in a proper mode. For example, if an argument is tagged as RO, it is locked in read mode. This tagging scheme allows the exploitation of parallelism in control flow. When two functions are sequenced one after the other in a program they can be run in parallel if the earlier function does not lock the arguments needed for execution of the second function. For example, if two functions take the same argument tagged as RO, they can be executed in parallel. On the other hand, if the argument is tagged as RW, unless one function releases the locks on the argument, the other function cannot start execution. The

The operations which manipulate the parsets are as follows:

insert (WO P, RO i, WO order)

Inserts an element i in the parset P as its last element. The cardinality of P increases by 1 after the insertion operation. The argument order returns the order of the inserted element. P is tagged as WO here as the function writes an element into the parset but does not read any element from it.

flush (WO ~)

Flushes the parset P. After flushing, the cardinality of P becomes zero.

delete (WO P, RO, n)

Deletes the nth element from the parset P. The delete operation leaves the order of the parset intact.

get (RO P, RO n, WO element)

The argument element returns the nth element of the parset P.
getcard (RO P, WO c)

The argument c returns the cardinality of the parset P.

The above operations provide the means for manipulating the elements of a parset. When functions are called on parsets, they acquire special meanings.

Function Semantics on Parsets

When a parset is passed as an argument to a function, three possibilities exist for the execution of the function. The function can execute in parallel on each element of the parset. This is the first type of execution. The function can also execute sequentially on each element of the parset one after the other. This is the second type of execution. In the third type of execution, the function takes the parset as a simple argument for further processing within the function. In order to differentiate between these three function types, the two keywords par and seq are used. We now illustrate the three types of function calls with simple examples.

par function call:

Example: par process (RO P);

The function processl) is applied to each element of P in parallel. Each activation proceeds asynchronously. If a function has more than one parset as its arguments, then

the cardinalities of all of them must be the same. This will enable a particular function activation to pick up the corresponding element from each parset. The par function call exploits the data parallelism expressed in a parset.

seq function call:

Example: seq print (RO P);

The function printO is applied to each element of P sequentially in the order of the elements.

Ordinary function call:

Example: myprint (RO P);

Here no keyword is prefixed to the function call. Hence this is treated as an ordinary function call, and the parset P is passed just as a plain argument to it.

As an example, the following can be the description of myprinto:

```
Function myprint (RO P) {
 seq print (P);
                    }
```

A special function called 'myid' is provided to identify the element of the parset on which the present function activation is operating. This function returns the order of the element of the parset on which a par or a seq function call is operating.

Defining the Functions which Execute on a Parset

When a par or a seq function is called with a parset as its argument, each activation of that function receives one element of the parset, Hence, the function is defined for one element of the parset. On the other hand, a function, which takes a parset as a plain argument as in the case of an ordinary function call, declares its argument type the same

as the parset type itself. The following example illustrates the difference. In this example, addl) concatenates strings and howMany tells the cardinality of a collection of strings.

```
Function MassConcat () {

        parset    P  of   string       "Work" ,    "Think" ,
        "Speak" ;
        parset    Q  of   string       "hard" I    "deep" I
        "truth" ;
        parset    R  <f   string;
        int    n;

par add (P,' Q, R);
hownany (R, n);

Function add (string RO x, string RO y,
string WO z)
{
concat (x, y, z);
}

Function howMany (parset RO WO n)
{

getcard (strset, n);
}
```

Each activation of function addt) binds an element of parset P to argument x, and the corresponding element of parset Q to y. The returned argument z is bound to the corresponding element of R. The cardinality of the parset P and Q must be the same in this case.

Concurrent Execution of Multiple Function Calls on Parsets

There can be situations wherein a parset, which is an output parameter of one function call, becomes an input, parameter in a subsequent parset function call. This offers additional possibilities of concurrent execution. This is explained in the following example:

```
Function Encourage () {
parset E of employee, A of assessment, R
of reward;
par assess (RO B,.WO .A)';
par encourage (RO A, wo R);
}
```

In the function Encourage 0 as given above, the employee records are assessed so as to encourage the employees by offering them suitable rewards. The output of assess 0, which is parset A, is an input to Encourage (). As soon as the function assess () finishes with anyone of its multiple activations, the WO lock on the corresponding element of the parset is released. After the release of the lock, the next function Encourage 0 acquires the RO lock for that particular element of the parset. Once the lock is acquired, the function can start its execution. Thus, multiple

functions can execute concurrently providing additional parallelism in control flow.

Indexparset

The indexparset is a special case of parset. It holds no elements but carries an index. The index can be seen as cardinality of the indexparset. A function call on an indexparset is activated index number times. Hence the indexparsets can be used when multiple activations of the same function are required. An indexparset can be declared as:

indexparset I;

Only three operations, namely setcard(), getcard(), and flush (), are performed on an indexparset. The last two are the same as described in Section 2.2.1 on parsets. Operation setcard sets the cardinality of the indexparset I to a given value c and is defined as:

setcard (WO 1, RO c);

When an indexparset becomes an argument to a par or a seq function call, each function activation receives an integer which represents the order of that particular activation. The following example demonstrates the use of an indexparset. It collects the status of distributed resources in a parset called StatusSet.

```
Function CollectStatus {
indexparset I;
parset StatusSet of int;
setcard (I, 10);
par myread (I, StatusSet);

        / * The activations of myread collect the status of  10 resources */
}

Function myread (int RO ResourceId, int WO s)

/ * ResourceId = current function activatior number * /
ReadStatus (ResourceId, status);
S = Status;
}
```

EXPRESSING PARALLELISM THROUGH PARSETS

Parsets can be employed for expressing both SPMD and MPMD kinds of parallelism. This is discussed in the following sections.

Expressing the SPMD Parallelism through Simple Parsets

The SPMD parallelism can be expressed by using simple parsets with par function calls. When a par function is called on a simple parset, the function executes in parallel on different elements of the parset. The simple parsets can be created in two ways. An empty parset can be declared initially and the elements can be inserted into the parset explicitly by insert calls. The other method of creating simple parsets is to convert an array of elements into a parset with a grain control mechanism. These two methods of

creating simple parsets are explained in Sections 2.3.1.1 and 2.3.1.2, respectively.

Creation of Simple Parsets using Explicit Insert () Calls

First a simple parset of the desired data type is declared. Multiple data belonging to the same data type can now be added to the parset using the insert 0 function. Then a function can be executed on this parset by a par function call. In order to express the SPMD parallelism in the processing of arrays, one may create a parset and explicitly insert the array elements into the parset with this method. An easy way to convert an array into a parset is through the grain control mechanism, which is specially designed for this purpose.

Conversion of Arrays into Simple Parsets by the Grain Control Mechanism

Through this mechanism, one can indicate the granularity that is desired to build such a parset out of an array. The mechanism consists of two constructs, namely granularity, which is a metatype, and a function CrackArrayO. The granularity works as a metatype in the sense that its value is a data type.

As an example, we may specify a granularity of int[100} to convert an array of type int[1000} into a parset. The parset will have ten elements, each of type int[100} as specified by granularity. It is possible to covert a multi-dimensional array into a parset by cracking the array in any dimension. For example, we may specify a granularity of int[25}[100} to convert a row major array of type

int[100}[100} into a parset. In this case, the parset will have four elements, each of type int[25}[100].

We take the following image transformation example to demonstrate the use of this mechanism. In this example, the array named A is converted into a parset. The array A consists of 1000 elements. Each element of the parset is constructed by combining 100 elements of the array. Thus the parset will have ten elements in it.

```
Function ProcessArray ()
intA [1000];

granularity 9 = int [100];
parset P, Q of g;
CrackArray (A, P, g);
par transform (P, Q);
par plot (Q) ;
flush (P);
}
```

Converting a data structure like an array into a group of several grains of specific granularity becomes possible with the metatype 'granularity: The target parset is declared as a simple parset of the same data type as that of the granularity. The array is converted into a parset by using the function CrackArray (). The function takes three arguments: the source array, the target parset .handle and the granularity. When the function returns, the parset handle corresponds to the new parset that is built out of this array.

After the conversion, there are two ways to access the array.

One is by manipulating the new parset handle, and the other is by directly using the array name. The array name refers to the copy of the array which is local. The parset handle refers to the copy of the array which may be scattered in the network. Now if both the handles are allowed to manipulate the array, it will create inconsistencies between these two copies. Hence till the parset handle is active, the array must be accessed only through the parset handle. However, a flush 0 call on the parset handle has the special function of deactivating the handle and storing back the new values of the array into its local copy. After the flush, the array can be referenced in the normal way.

By setting the granularity, the user can set an upper bound on the number of processors that can be utilized by the underlying parset kernel for an execution. For example, for a single-dimensional array of size 1000, the granularity may be set to 100 or 250, thereby creating parsets of size 10 or 4. Thus a varying degree of parallel execution on a parset can be obtained by using granularity.

The grain control mechanism thus achieves two goals. First, it allows an array to be treated directly as a parset without the need for multiple insert () calls. Secondly, it can control the degree of parallel execution of a function on an array

Dynamic Specification of Granularity

When a granularity variable is declared as an array, the dimensions of the array can be specified during runtime. The following example demonstrates this:

```
Function ProcessArray () {
int A [1000];
granularity g = int [];
parset P, Q of g;
int dim;
dim = 100;
CrackArray (A, P, g [dim]);
par transf0rm (P, Q);
par plot (Q);
flush (P);
}
```

It can be noted that the value of the granularity variable has been declared as an integer array of single dimension without mentioning its dimension value. The dimension of 100 has been specified during runtime in this example.

Expressing MPMD Parallelism through Parsets

The MPMD parallelism can be expressed with parsets by using two mechanisms. The first is to use polymorphic functions with untyped parsets. It offers a limited way of expressing MPMD parallelism, and can be readily implemented in a language which provides function polymorphism. The other mechanism is a general one and uses function parsets. These mechanisms are described in the following sub-sections.

MPMD Parallelism through Polymorphic Functions

Untyped parsets can be used for expressing MPMD parallelism. In a typed parset, only the elements of the specified type can be inserted, whereas in untyped parsets, elements of any type can be inserted. An untyped parser declaration does not mention the type of its elements. As an example, an untyped parset Q can be declared as:

parset Q;

In this case, a parset is created as a collection of data belonging to different types. A polymorphic function identifies the type of each element and executes the required code on it. For example, with a function call such as par process (Q), various activations of the function processt) may receive arguments of different types. In this way, we can obtain the concurrent execution of different codes on different data.

But this mechanism cannot fully capture the MPMD parallelism for the following reason. Always the same piece of code is executed on two different elements of a parset if they are of the same type. So we cannot execute different codes on the elements of the same type by this mechanism. By using the function parsets, this limitation can be overcome.

MPMD Parallelism with Function Parsets

A function parset can hold a collection of functions. Another parset is used to hold the corresponding argument set. The

cardinalities of these two parsets must be the same. The function parset can now be invoked on the corresponding argument parset to achieve the general MPMD parallelism. The following example shows the structure of such a program:

```
Function MPMD-Prog-Structure () {
int d11, d12, d3;
char d2;
parset F of func = {f1 (int, int), f2 ( char), f3 ( in t) } ;
parset Dl = {dll, d2, d3};
parset D2 = {d12, NIL, NIL};
par (F) (Dl, D2);
Function fl (int RO dll, int RO d12) {
}
Function f2 (char RO d2) {
}
Function f3 (int RO d3) {
}
```

In the above example, function f1 0 will take its first argument from the first element of parset D 1, and the second argument from the first element of parset D2. Similarly, the functions f20 and f30 pick up their arguments from D1 and D2.

IMPLEMENTING PARSETS ON A LOOSELY COUPLED DISTRIBUTED SYSTEM

We discuss an implementation of the parset constructs on a network of workstations consisting of Sun 3/50 and Sun 3/60s, running

SunOS version 4.0.3. The environment supports the Network File System (NFS). TCP/IP has been used for inter-process communication. The parset constructs are provided as extensions to the C language. A parset pre-processor translates the parset constructs into C code. An overview of the implementation is now given followed by the description of various components. We studied the performance of an application using the implementation.

Overview of the Implementation

The heart of the implementation consists in a distributed parset kernel. The kernel is divided into resident and volatile parts. Each parset has an associated process called P-Process to maintain and manipulate the elements of the parset. A par function call on a parset is executed with the help of separate processes called

E-Processes. P-Processes and E-Processes form the volatile kerneL P-Processes reside on the same node where the user program resides. E-Processes reside on different nodes to exploit the parallelism.

The resident kernel consists of daemon processes which are started during the boot-up time on the machines that are willing to participate in the execution of programs that use parsets. The copy of the resident kernel is obtained by nodes from a designated node using the NFS. In this way, both diskless as well as diskful machines can obtain the resident kerneL The resident kernel manages the P-Processes and the E-Processes. It provides an interface to user programs to create the parset processes and to execute various functions on it.

The Parset Preprocessor

Figure 2.1 shows the functionality of the parset preprocessor. The high-level user program is provided with clean parset constructs, as described in the earlier sections, which hide the underlying distributed implementation. The parset preprocessor then translates the user program into a low-level C code, which makes calls to the resident kernel and the parset processes.

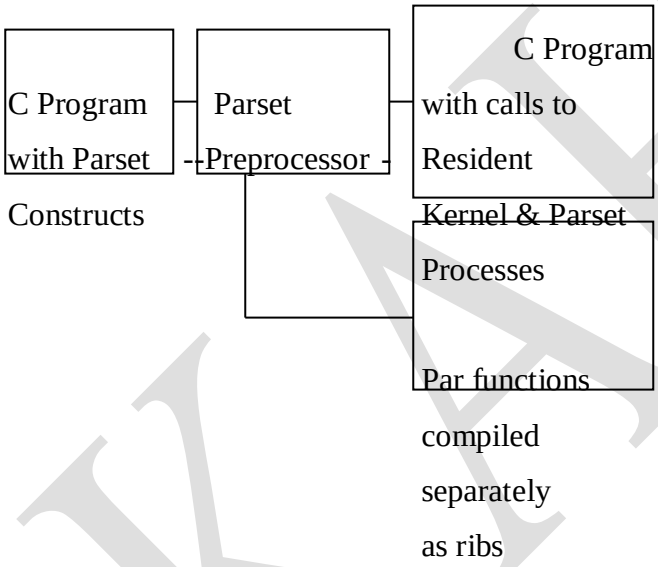


FIG. 2.1 The Parset Preprocessor`

The parset preprocessor identifies the par function calls separately as Remote Instruction Blocks (RIBS). An RIB contains a code which can be migrated at runtime to a remote node. The migration of the RIB code is the migration of the passive RIBs and not active processes or tasks. The RIB is given control for actual

execution at the remote node. RIBs are named and compiled separately by the preprocessor.

Remote Instruction Blocks (RIB)

The RIB facility was developed during the course of this implementation. The facility is similar to Remote Evaluation (REV) [7] developed by Stamos and Gifford. With the RIB primitives, as opposed to the RPC primitives [6], we can migrate a code to be executed to a remote site at runtime and get it executed there.

The parset preprocessor names and compiles .the RIBs separately. Whenever a block has to be executed on a remote node, its name and address is made known to the remote node. The remote node can access the RIBs by using the NFS, and can execute it as and when required.

The Distributed Parset Kernel

The parset kernel is distributed over the network. It consists of a resident and a volatile part. The resident part of the kernel is always present on all the nodes that participate in distributed execution. The volatile part is dynamically created on selective nodes depending upon the requirements. Figure 2.2 shows the organization of various components in the implementation. The functionalities of each are described in the following sections:

The Resident Kernel

The resident kernel performs the low-level system-dependent tasks which deal with the primitives for RIBs, creation and termination

of P-Processes and E-Processes, etc. It is responsible for selecting the lightly loaded nodes for execution and managing the distributed execution. Each local resident kernel contacts other resident kernels and fmds about the lightly loaded nodes. Achieving the program scalability is also a function of the kernel.

The resident kernel provides an interface to low-level user programs. The interface consists of the calls for registration and de-registration of a user process, creation and destruction of parset processes, and notifications of function execution on parsets. When an executing user program declares a parset variable, the resident kernel spawns a parset process called P-Process. This P-Process manages the parset on the node where the user program runs. Any further manipulations performed by the user program on this parset with Insert(), get(), delete(), getcar() and flush() calls, are directed to the corresponding P-Process bypassing the resident kernel.

During the remote execution of a function in a par call, the resident kernel creates new E- Processes, or locates free executor processes depending on their availability on the nodes suitable for the execution. For a particular function execution, the resident kernel helps P-Processes and the corresponding E-Processes to synchronize. After the synchronization, P-Processes send the required parset elements to E-Processes.

In order to improve the execution efficiency, the kernel can combine several elements in a parset together to form a super-

grain. It can be noted that the super-grain formation is different from the grain control mechanism. The latter is a mechanism meant for the user whereas the former is used by the kernel, making it transparent to the user. The resident kernel guides the volatile kernel for the super-grain formation.

The Volatile Kernel

The volatile part of the kernel consists of P-Processes and E-Processes, as mentioned earlier. For a function execution, the P-Processes which correspond to parsets involved in the execution, perform the required inter-process communication with the allotted E-Processes. In this fashion, all P-Processes and E-Processes proceed concurrently. An E-Process can be allocated for other incoming execution requests after the current request is completely processed. Similarly, a P-Process is derailed by the resident kernel when it receives a destroy () call.

As discussed earlier, a P-Process manages one single parset variable. A parset variable is a collection of multiple grains (elements). When a par or a seq function takes a parset variable as its argument, it specifies the parset variable as an RO, WO, or RW variable depending on its usage inside the function. At the time of actual execution, the elements of the parset have to be locked according to these specifications in order to exploit the concurrency as discussed in Section 2.2. In case of multiple parallel activations of a function, each activation operates on a different grain. Hence each activation can proceed independently as long as it can obtain the required locks.

At first sight, it appears that in the place of WO locks, one may use RW locks, thereby eliminating the need for additional WO locks. This is based on the fact that along with a write permission, there is no harm in granting a read permission also. But this is not so in reality because when an argument gets a read lock, it has to be moved to the remote site where the function is executing. But a WO argument need not be moved to the site of the remote function, since the function does not read the value of this argument. Thus these three distinct locks help in minimizing the communication overhead.

With an execution request of a par or a seq function, a parset process first establishes contact with the remote E-Process through the resident kernel. Then it continues to set the locks on the grains. As and when a lock is set, S the following action is taken corresponding to each lock:

RO: 1. Send a copy of the grain to the remote E-Process.
2. Release the lock on the grain.

WO: 1. Receive the grain value from the remote E-Process.
2.Update the grain value in the parset.
3.Release the lock.

RW: 1. Send a copy of the grain to the remote E-Process.
2.Receive the grain value from the remote E-Process.
3.Update the grain value in the parset.
4.Release the lock.

A Case Study

In this section, we discuss a simple case study of the image transformation problem encountered in computer graphics. Two examples of such transformations are rotation and dragging. Figure 2.3 shows the test program for image transformation using parsets. The program was run on a network of Sun workstations. Since the transformation problem was an SPMD type of problem, we used the techniques presented in Section 2.3.1.2.

The following observations can be made with respect to the case study program:

The program only expresses the parallelism present in solving the graphics rotation problem without any explicit use of system-dependent primitives.

```
define TotalPoints 20000
#define GrainSize 400
typedef struct {int x, y;} point;
typedef point grain [GrainSize);
Main:
point image [TotalPoints);
granularity G = grain;
parset P of G;
read-image (image);
CrackArray (image, P, G);
par transform (P);
flush (P);
```

EndMain

Function: transform (grain RW image-
grain) {
int i;
for (i = 0; i <. GrainSize; i++)
image-grain = rotate (image-grain);

}

The Image Transformation Problem

- The user can control the size of the grain by specifying the grain size in the program.
- The function transforml) will be compiled separately as an RIB so that it can be migrated to a remote node for execution. The system will make appropriate choice of nodes and the mechanism is completely transparent to the user program.
- The program is easy to understand and to debug.
- The program can be executed on a distributed system supporting the parset construct and hence can be ported easily to other systems.
- In the case of node failures, the kernel can reassign the sub-tasks which remain unevaluated to other nodes without the involvement of the user.

POSSIBLE QUESTIONS(2 Marks)

1. What is Parset Data Structure?
2. Define Index Parset.
3. Define SPMD Parallelisim.
4. What is RIB?
5. Distinguish between Resident Kernel and Volatile Kernel.
6. What is Parset Preprocessor?
7. Write a note on Distributed Parset Kernel.
8. What are the advantages from System's Point of View?
9. Define Mobile Grid Model.
10. List the applications of Grid Model.

POSSIBLE QUESTIONS(6 Marks)

1. Brief about Cluster Computing Models and Grid Models.
2. Write a note on expressing SPMD parallelism through simple parsets.
3. What are the advantages in System's and Programmer's Point of view in parallel programming on distributed Systems.
4. Write a note on expressing MPMD parallelism through simple parsets.
5. Explain Function semantics on Parsets.
6. Write a note on Parset Preprocessor and Remote Instruction Block.
7. Write about Parset Data structure.
8. Explain the functionalities of Distributed Parset Kernel.
9. Write short notes on Parset Data Structure and Indexparset.
10. Discuss about Implementing Parsets on a Loosely Coupled Distributed System.

Karpagam Academy of Higher Education

Department of CS, CA & IT

Subject: Grid Computing (17CSP205B)

Batch : 2017-2019 **Class: I M.Sc CS**

Objective Type Questions

UNIT I

S.	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	KEY
1	The ARC approach results in transparancy in both	Distribution and programming	Distribution and heterogencity	Programming and computing	Programming and processing	Distribution and heterogenc
2	The performance analysis shows the speed-up due to……. and scaled down	Parallel execution	Parallel transmission	Predefined execution	Network selection	Parallel execution
3	ARCC stands for	Anonymous remote computing & communicatio	Anonymous remote computation &	Anonymous remote comparing & communicati	Anonymous retrieve computing &	Anonymous remote comparing &
4	The sneha samuham grid computing model has been implemented over	Local wide grid	World wide grid	Inter wide grid	Nation wide grid	Nation wide grid
5	The combination of wireless communication and cluster computing is	Mobile cluster computing	Mobile cluster combination	Mobile communicati on cluster	Mobile computing cluster	Mobile cluster computing
6	The key idea in building the mobile grid is to	Data and device	Data and service	Mobile and service	Test and service	Data and service
7	…….. belongs to the class of NP-hard	Job shop scheduling	Job scale scheduling	Job shop sharing	Job share scheduling	Job shop scheduling
8	The mobile grid requires monitoring data for a variety of tasks such as	Fault protection	Fault precumption	Fault detection	Fault reduction	Fault detection
9	A key property of distributed systems is that they are	Open-closed	System-ended	open-property	open-ended	open-ended
10	The parset constinct consists of a data structure and a set of function which operates	Data structure	Data blocker	Block structure	Data set	Data structure
11	When the elements of a parset belongs to a basic datatype.it is called a	Function parset	Simple parset	Sample parset	Basic parset	Simple parset

12	A parset is kept logically ordered on the basic of the enty of its element on a…. basics.	Second-come-first-served	First-come-second-served	First-come-first-served	Second-come-second-served	First-come-first-served
13	The function can execute in parallel on each element of the	Keywords	Parset	Element	Process	Parset
14	The …….function call exploits the data parallelism expressed in a	Par	Print	Seq	My print	Par
15	The cardinality of the parset ……..must be the same in this case.	P and R	Q and K	R and Q	P and Q	P and Q
16	The……..parallelism can be expressed by using simple parsets with par	SPMD	SMPD	SMDP	SMTP	SPMD
17	The mechanism consists of two constructors,	Glarity	Graduality	Granularity	Granulity	Granularity
18	The array is converted into a parset by using the	Crack array()	Target array()	Copy array()	Sparse array()	Crack array()
19	NFS stands for	Network frame system	Nework function system	Network file selection	Network file system	Network file system
20	The parset kernel is distributed over the	Connection	Network	Function	Places	Network
21	A Parset is created as a collection of _____ belonging to different	Process	Data	Function	code	Data
22	A_____Function identifies the type of each element and executes the required code on it.	polymorphic	par	seq	Parset	seq
23	A_____ can hold a collection of functions.	Untyped Parsets	Parset	Function Parset	Typed Parsets	Function Parset
24	The _____ of these Two Parsets must be same.	Cardinalities	Granularity	Graincontrol	Datastructure	Cardinalities
25	Expansion of NFS_____	Network File station	Network Function	Nested File System	Network File System	Network File station
26	_____ has been used for inter-Process communication.	SMTP	TCP/IP	TCP	FTP	TCP/IP
27	The_____is divided into resident and Volatille	Parset	Kernel	Program	Parfunction	Kernel

28	_____ reside on different nodes where the user program resides.	P-Process	E-Process	Inter-Process	Outer_Process.	P-Process
29	_____reside on different nodes to exploit the Parallelism.	P-Process	E-Process	Inter-Process	Outer-Process	E-Process
30	The _____Kernel consist of dameon Process.	Parset	Distributed	Resident	Volatile	Resident
31	The Parset _____then translstes the user Program into a	Process	Processor	processing	Preprocessor	Preprocessor
32	Expansion of RIB _____	Remote Instruction Blocks	Remote Information Blocks	Remote Instruction Based	Remote Instruction Based	Remote Instruction Blocks
33	Expansion of REV _____	Remote Evaluate	Remote Evaluation	Remote Evaluate version	Remote Evaluation version	Remote Evaluation
34	The _____Parset is dynamically created on selective nodes depending upon the requirements.	Volatile	Resident	Local Volatile	Local Resident	Local Volatile
35	The _____Provides an interface to low level user Programs.	Volatile	Resident Kernel	Kernel	Parset	Resident Kernel
36	P-Process is Detailed by the resident Kernal when it recievs a_____call.	insert()	get()	getchar()	destroy().	destroy().
37	The _____Problem was an SPMD type of	Transforms	Rotation	transformation	Dragging	transformation
38	Expansion of PVM_____	Parallel Virtual	Process Virtual	Private Virtual	Public Virtual	Parallel Virtual
39	The function _____will be compiled separately as an RIB.	Insert()	flush()	transform()	get().	transform()
40	With an execution request of a _____function,a Parset Process first establishes contact with the remote E-Process.	par	a seq	a Par or a seq	a par and a seq	a Par or a seq
41	_____is a net by interconnected several networks on resources.	Cluster	Grid	Parallel computing	ARC	Grid

42	_____ is a interconnected workstations and attractive propostion due to the rapid growth in speeds of interconnection	Grid	ARC	Clusters	Parallel computing.	Parallel computing.
43	The application of several computer to a single problem at a same time is called_____	Parallel computing	Grid model	Grid computing	Cluster	Grid computing
44	AEC stands for_____	Anonymous Remote computing	Automatic Remote computing	Anonymous Remote control	Anonymous Remote client	Anonymous Remote computing
45	_____ is a cluster computing model.	CORBA	NOW	ARC	DP	ARC
46	CORBA stands for a._____	Common object request broker architecture	common object request broker access	control object request broker	common object required broker access	Common object request broker
47	NOW stands for _____	Network of workstations	Network of work	Node of workstations	Network object	Network of
48	DP stands for_____	Distributed process	Distributed pipes	Distance pipes	Documented pipes	Distributed pipes
49	IGC stands for_____	Interactive Grid Computation	Interface Grid Computing	Interactive Grid Control	Interactive Grid Communicati	Interactive Grid Computati
50	_____treats several loosely coupied resources as a single entity	DP	COBRA	ARC	CORBA	ARC
51	The transparent programmability of communications parallel task in a _____	Network of workstation	Distributed pipes	Grid model	CORBA	Network of workstation
52	_____do not address issues specific to parallel programming.	ARC	ARCC	P-CORBA	CORBA	CORBA

CLASS: I M.SC CS **COURSE NAME: GRID COMPUTING**

COURSE CODE: 17CSP205B **UNIT: II(ARC)** **BATCH-2017-2019**

UNIT-II

SYLLABUS

Anonymous Remote Computing Model: Issues in Parallel Computing on Interconnected Workstations – Existing Distributed Programming Approaches – The ARC Model of Computation – The Two-tired ARC Language Constructs – Implementation. Integrating Task Parallelism with Data Parallelism: A Model for Integrating Task Parallelism into Data Parallel Programming Platforms – Integration of the Model into ARC – Design and Implementation – Applications – Performance Analysis

ANONYMOUS REMOTE COMPUTING MODEL

It basically two-tiered architecture. At the lower layer, the primitives which are built over a kernel provide the basic support for anonymous remote computing while at the upper layer; various easy to use high-level programming language constructs are supported. Workstation cluster and scientific computing in academic are becoming increasingly popular The processing power of the workstation has witnessed tremendous growth resulting in clusters of workstations. Parallel programming on workstation systems is a relatively new field and has been an attractive proposition ever since it was used . Start the process on those nodes.

ISSUES IN PARALLEL COMPUTING ON INTERCONNECTED WORKSTATIONS

Parallel computing on tightly-coupled distributed systems has so far been widely popular. several key issues distinguish parallel computing on work station clusters. resent advantages in communication technology and processor technology and processor technology make parallel programming on loosely coupled distributed systems of tightly-coupled massively parallel architecture .Several key is distributed work station cluster from that network

It follows four categories:

i) Changing loads on the nodes of the network

A distributed system consisting of inter connected workstation massively parallel experiences a wide fluctuation of loads on individual nodes. a considerable amount of unused computing capacity is always present in the network. A program is said to be load adaptive if adapts to the changing load in the system. A programmer may not able to use the knowledge of load fluctuation in the program unless an adequate language support is provided. It can challenge tasks. A program can be load adaptive it changing load in the system. Using parallel virtual machine. A load on a particular machine increases during execution process on that node would suffer.

ii) Changing node availability on the network

Distributed systems are characterized by nodes that keep going down and coming up over a period of time. it only three nodes available. A program at a given instance of execution may have made use of five nodes while in another instance. A sub-task ,a node or a link might crash and may again come up before the completion of the execution of the program.

iii) Difference in processor speeds and network speeds

It consists of inter connected workstations are processor speeds and network speeds. The communication overhead in class of distributed systems is fairly high. It only for grain parallelism. Selecting appropriate grain sizes during runtimes becomes important as a consequence of this variation. Communication overheads play a role of speed ups of parallel tasks on these systems. Heterogeneous cluster processors have different speeds.

iv) Heterogeneity in architecture and operating systems

It basically interconnected workstations with distributed file system and standardization of software.NFS are common uses. The binary executables files are not compatibles between architectures, using several difficulties in parallel programming on

workstations clusters in messages passing or shared memory abstractions. The architectural differences between workstations are more difficult to handle than operating system differences. the binary executable files are not compatibles between architectures.

EXISTING DISTRIBUTED PROGRAMMING APPROACHES

COP approaches to programming on workstation systems can be grouped with some amount of overlap between the various categories. That are intended for providing distributed services. Some of them have been proposed for parallel programming other are intended for providing distributed services. we analysis the suitability of these approaches from the parallel programming point of view. This model is many rise to many difficulties while programming on this system.

i) Bare Socket Programming

The most primitive constructs for inter-process communication between workstations are sockets. Communication is at the level of un typed bytes streams. The selection of nodes, tacking of failures, load adaptability and heterogeneity at the level of the executable code. Such as the selection of nodes tacking of failures load adaptability and heterogeneity at the level of the executable code have to be handled explicitly by the program. XRD is provided in order to handle.

ii) Remote Procedure Calls (RPCs)

Parallel programs are written with RPC in combination with lightweight threads. it based on RPC mechanism to heterogeneous environments. These attempts largely focus on the issue of design of the stub generator for RPC systems. it is most appropriate for providing distributed services rather than writing parallel programs on workstation clusters. The programming on loosely coupled distributed systems. It is more than appropriate for providing clusters.

iii) Remote Execution

A remote execution facility for migrating code fragments to remote servers for execution. The REV has been developed for providing migration of executable code to servers. it only access on client-server communication. it workstations due to its lack of adequate support for heterogeneity, load adaptability and faculty tolerance. The executable code is clint nodes.

iv) Message Passing Abstractions

These are the higher level abstractions to are the concept of typed data communications is introduced and un typed byte can be retained .It provides support for process migration. process migration-based solutions are useful for adapting to load variations on homogeneous systems only based on massage parallel virtual machine ,its provides support for process migration.

v) Distributed Shared Memory Abstractions

They range from bare distributed shared memory support such as the ether system. shared memory abstractions appear to provides a much easier programming. once the number of processes is fixed by a program, it utilized the additional computing power that becomes available during runtimes.

vi) Object-Oriented Abstractions

Several object-based and object-oriented paradigms such as emerald, processes are replaced by objects. the communication in this case is inter object rather than inter-process.the COP model is still preserved but at the level of objects .the various drew backs of these abstractions in the context of workstation systems have been discussed. The nodes in the network remain anonymous and a run time system starts the required processes on remote nodes

THE ARC MODEL OF COMPUTATION

It basically model of computation is designed to meet two goals as clear separation between programmers concerns and systems concerns and account for heterogeneity fault

tolerance, load adaptability and processor availability. the original of the conventional COP model of parallel programming on workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. To execute remain anonymous to the program. At the time multiple programs can be generating RIBs and multiple anonymous emote participants can be joining or leaving the ARC system. An RIB can future generates new RIBs.

Comparing ARC and COP models

i)Task Synchronous Systems

A task synchronous system provides synchronization between computing entities at the level. Two computing entities are synchronized at initiation time or the completion entities. Task can be deposited in a tuple space and decoupled processes pick up the deposited tasks.

ii) Call Synchronous Systems

Two computing entities communicate by calling methods defined by each other, that entity calls a methods action to be blocks for result to return RPC.

The Distributed processes models are call synchronous COP systems. mechanism is used either a future mechanism is used or a call back mechanism is implemented to obtain the return value of a call. The latency tolerance mechanism of CHARM++ ,future-based call asynchronous system

iii) Call Asynchronous Systems

It based on object oriented programming systems are call asynchronous systems. It based on non-blocking fashion. Using the latency tolerance mechanism. Method calls return immediately in a non-blocking fashion. Either a future mechanism is used or a call back mechanism is implemented to obtain the return value of a call.

iv) Message Synchronous Systems

Two computing entities exchange messages such as arbitrary data values. message communication is blocking. Communicating sequential processes CSP is based on synchronous COP system.

v) Message asynchronous systems

Two computing entities exchange messages such as arbitrary data of time .message passing mechanisms such as PVM and MPI are COP system. this type of synchronization can be achieved by shared memory as well as message passing techniques. A message can arrive at any time, message is usually typed and the receiving process knows the type of the incoming message.

vi)The ARC approaches

ARC extends the computing entities in the horizontal domain to RIBs, ARC can be classified as a task synchronous system. The vertical domain ARC can be classified as a task synchronous system. The synchronization is provided with the help of synchronizers.

RIBs and Synchronizers

RIP are the code segments that can be executed at anonymous nodes.RIP is achieved through synchronizers .RIBs are the code segments that can be executed at anonymous nodes. synchronization between independently executing RIBs is achieved through synchronizers

i) Remote Instruction Blocks

Open - tenderness: the target machine for execution is unspecified. the source code of the RIB needs to be migrated. As a consequence of this prosperity ,heterogeneity and local adaptability are supported. the target machine is unspecified .as a consequence of this property.

Fault tolerance: it basically two-tired architecture using high level program used. C or C ++ The failure of an anonymous node, the RIB may be executed on a different node or

in the worst case execution on the host node can be guaranteed. The choice of a scheme of fault tolerance depends upon the semantics of the high-level ARC constructs.

ii) Synchronizers

Synchronizers are mechanisms that are needed to achieve synchronization between independently executing RIBs. it based sync variables and lock specifications are provided. The first type of synchronizer, the SYNC variable to assure that the execution sequence of the associated RIB is completed. Functional languages have used similar variables. A function begins arguments.

iii) Dynamically Growing and Shrinking Trees

It is represented as dynamic growing and shrinking trees .It is not supported by RIBs. It is a very challenging task and needs further research. as RIBs complete execution the tree shrinks. RIBs are generated during execution a shrinking tree may again start growing. Achieving different types of RIB communication rather than task synchronization is a very challenging task and needs further research.

iv) Benefits of the ARC Model

The ARC model based on RIBs contained in a program instead of a COP that communicates explicitly. This approach grand's distribution transparency to parallel programs. ARC also provides heterogeneity transparency.

Abstracting load and speed will be available for a parallel program to execute on un evenly loaded heterogeneous machines.

THE TWO TIERED ARC LANGUAGE CONSTRUCT

The two tiered ARC language constructs the two-tiered ARC language constructs. The lower layer provides primitives to support system such as the creation of fault tolerance ,load sensing and source code migration. It based on fault tolerance, load sensing and source

code migration. The upper layer consists of higher level language extensions that advocate ARC programming methodology

i) The Evolution of RIB

This is operating system for load balancing ,this motivation is parallel programming it is basic building blocks of an APC program. An RIB can be submitted, executed and manipulated with help of the ARC kernel interface calls. RPC is perhaps the most popular remote execution paradigm used to build distribution applications. Many specific RPC protocols have been developed. The client makes calls simultaneously to multiple servers for loading files.

ii) The Design Of The Lower Layer Arc Interface

Obtain a lock on an anonymous computing unit, pack the arguments to the RIB, pack the arguments to the RIB, post the RIB on the lock obtained ,the results of the RIB execution start-up and close-down primitives, primitive support for RIBs, primitives for packaging primitives for RIB posting ,primitives for parameter setting, horse power factor and asynchronous intimations, the horse power factor(HPF)primitive, the asynchronous intimation primitive

The upper layer ARC constructs

The primitives explained above are built over the ARC kernel. the ARC function call models .a high level ARC language paradigm may use these primitives in various ways to provide easy-to-easy ARC language constructs.ARC paradigms for object-oriented programming respectively.

Blocking and Non-blocking ARC Calls

The user can tag the functions as belonging to one of the blocking or the non-blocking call classes. The non-blocking version of the ARC function call needs a synchronizer

It may be executing on a remote anonymous node .at a later point of time ,the result of the non-blocking call can be obtained by waiting on the corresponding synchronizer. It is possible to provide appropriate higher level ARC language semantics to suit the higher level requirements.

IMPLEMENTATION

The anonymous remote computing mechanism is provided as an extension to C language. It is implemented on a heterogeneous as an extension to C language. A distributed ARC kernel is spread over the workstations that participate in anonymous remote computing. It is local activity and global activity It consists of these primitive calls.

A distributed ARC kernel is spread over the workstations that particular in anonymous remote computing. The entire domain is partitioned into three logical clusters of multiple local variables.

The System Coordinator

There is only a single system coordinator in a given domain of logically grouped clusters. The system coordinator's functions are to manage locks, route RIBs and maintain migration history. It only functions as a policeman controlling and routing the traffic of ARC calls

Lock management

A machine wants to improve its utilization ,it registers with the system coordinator through local coordinator. A lock request arrives, a statistic-daemon is contacted to access the current load information and the normalized speed of the machine, the HPF is computed and returned. A lock request improve its utilization or to share its work with an anonymous remote node.

Routing RIBs

An RIP posting is routed to the corresponding free local coordinator for execution. An RIP posting consists of the arguments data- packet, the source code or compiled code, it is appropriate and the make directives for various architecture.RIP binaries are prepared for the target machine. The results of the RIPs are sent back to the corresponding local coordinators.

Maintaining Migration History

A history of resent migrations is maintained .A machine may received an RIB task belonging to an earlier posted source code. The binaries available in the file system that of the cluster repetitive migration and recompilation are avoid by using the history of resent migrations across clusters

The Local Coordinator

The local coordinator runs on a machine that participates in the ARC system either to improve its utilization or to share its work with an anonymous remote node. Any ARC communication to or form the local processes is achieved through the local coordinator are described in the sub-sections. Any ARC communication from the local variable.

Improve Local Utilizations

An online command can be executed that directs the local coordinator to obtain a particular amount of work from the ARC system .this generates an asynchronous intimation that travels to the remote programs. the results are forwarded to the local coordinator. A task executor may also generate new RIBs.

Accept RIBs From Local Processes

RIBs are initially generated by user processes. later the RIPs themselves may generate new RIPs .A new RIB may be generated upon the receipt of an asynchronous intimation. The local coordinator provides the intimation to a program by using a signal. The results of remotes RIBs arrive, they are queued up in a result queue. A synchronizer in the program requires a particular result, it is queue is searches.

The RIBs

RIBs are extracted from the program by using an ARC compiler .An RIB code is generated b using a generic RIB process that accepts arbitrary arguments and posts a result .Dynamic RIBs do not die immediately and can further accept a new set of arguments. A specialized RIB is prepared from the generic RIB by integrating the required code fragments into it. The processing power of the workstation has witnessed tremendous growth resulting in clusters of workstations. Parallel programming on workstation systems is a relatively new field and has been an attractive proposition ever since it was used.

Time-Outs and Fault Tolerance

The code that maintains the time-outs is integrated with the code that generates RIBs. A timer signal ticks at regular intervals. . It based on fault tolerance, load sensing and source code migration. The upper layer consists of nodes. The time-out parameter in the lower layer ARC primitives is specified in terms of this interval. . A program is said to be load adaptive if adapts to the changing load in the system. A programmer may not able to use the knowledge of load fluctuation in the program unless an adequate language support is provided. the value is time-out and retries parameters permit a re-execution ,the RIB may be resent to another suitable remote node.

Security and Portability

Security in ARC implementation is built over the security provided by operating systems over ARC is implemented. The system coordinator accepts connections only from local coordinators on trusted clients. the original of the conventional COP model of parallel programming on workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. A remote local coordinator accepting RIPs arriving from anonymous runs with specific user permissions especially created for the purpose of ARC computations. portability to ARC programs nodes primitives are made portables.

INTEGRATING TASK PARALLELISM WITH DATA PARALLELISM

Data parallelism refers to the simultaneous execution of the same instruction stream on different data elements. Several programming platforms target the exploitation of data parallelism. Control parallelism refers to the simultaneous execution of different instruction streams. This is also referred to as task parallelism or functional parallelism. Some of the tasks that constitute the problem may have to honor precedence relationships amongst themselves. The Control parallelism with precedence's. It is the parallel execution of distinct computational phases that exploit a problem control parallelism. This kind of parallelism is important for various reasons.

In Multi disciplinary applications there is an increased interest in parallel multi-disciplinary applications it different scientific disciplines and may be implemented for parallel computation. The air shed model is a grand challenge application that characterizes the formation of air pollution as the interaction between wind and reactions among various chemical species

Complex simulations: Most of the complex simulations developed by scientists and engineers have potential task and data parallelism. a data parallel platform would be able to exploit the control parallelism.

Real time requirements : it is characterized by their strict latency time throughput requirement. Task parallelism lets the programmer explicitly partition resources among the application modules to meet such requirements.

Performance Task parallelism allows the programmer to enhance locality and performance by executing different components of a problem concurrently on disjoint sets of nodes. It also allows the programmer to specify computation schedules that compiler

Problem characteristics many problems can benefit from a mixed approach with parallel coordination layer integrating multiple data parallel co ordinations. Some problems admit both data and task parallel solutions

Task and data parallelism are complementary than competing programming models. Many problems exhibit a certain amount of both data parallelism and control parallelism. It is desirable for a parallel program to exploit both data and task parallelism inherent.

A MODEL FOR INTEGRATING TASK PARALLELISM INTO DATA PARALLEL PROGRAMMING PLATFORM

Expectations from an integrated platform a high level parallel programming platform stem from the nature of applications utilized the platform. The requirements come from the desired expressivity of the application, possible transparency in programming, exploitation of parallelism

Impressibility in order to exploit parallelism in an application the program must express potential parallelism execution units. An elegant impressibility scheme should reflect the parallel units, data parallel units and precedence among the tasks in the program.

Transparency it is desirable to relieve the programmer from details relating to underlying network programming. This results in the programmer concentrating on his application domain itself. Network programming information coded in the application, a major portion of the program.

Performance system level optimizations by the parallel programming platform can improve the performance of applications. The system can achieve load balancing for the application, further enhancing performance. The run-time scheduling decisions by the system.

Other desirable Properties of the system include fault resilience, facult resilience, fault tolerance, accounting for heterogeneity in machine architecture and operating system and portability of application.

The expressible that can be provided is influenced by the nature and organization into converted

Programming model

The model aims at a parallel programming platform that permits expressibility for task and data parallelism so that both can be exploited. A large number of existing data parallel programming platforms for NOWs, it would be useful to formulate the problem as integrating task platforms. A block could be a high level characterization of a data parallel module in accordance with the principles of the underlying data parallel platform.

Events of interest signify the completion of one or more tasks meets the pre conditions for another task that is writing to be executed. This takes care of the probility factor in the order of completion of tasks that constitute the program.

Other crops up during the integration parallel sub division of divisible tasks. The underlying data parallel model could not be sub-dividing a data parallel platform could be a significant event in the proposed integrated model the system has to initiate the user process an event of its interest occurs.

It because necessary to integrate to the existing system the notion of the notion of a task as a collection of sub tasks. The program expressibility of the model reflects the task parallel blocks and precedence relationships in the task graph.

Task begin and task end are introduced to demarcate the blocks in the block structured code. Another construct is provided to specify the pre conditions of the tasks.

Program Structure and Translation Of A Task Graph

A sample task graph and its block-structured code are illustrate the expressibility by model .the translation of a given task graph into program structure favored by the model. Also it illustrates the translate of a given task graph into the program structure favored by the model.

The outline program expressibility of a task graph in the model. At the level Task1 and Task 2 could be executed in a control parallel fashion at the beginning of the run itself. This is evident from their task begin constructs it is not followed by the construct on finish

Separation of System's and Programmers Concerns

Task and data parallelism are complementary than competing programming models. Many problems exhibit a certain amount of both data parallelism and control parallelism. It is desirable for a parallel program to exploit both data and task parallelism inherent.

In Multi disciplinary applications there is an increased interest in parallel multi-disciplinary applications it different scientific disciplines and may be implemented for parallel computation. The air shed model is a grand challenge application that characterizes the formation of air pollution as the interaction between wind and reactions among various chemical species

INTEGRATION OF THE MODEL INTO ARC

ARC model of computation

Is a code fragment that can be migrated into a convenient anonymous remote node at runtime node at any mechanism of process creation or inter-task communication at the programming language level. The nodes at RIBs need to be executed remain anonymous .The lower layer provides primitives to support system such as the creation of fault tolerance ,load sensing and source code migration. It based on fault tolerance, load sensing and source code migration. The upper layer consists of higher level language extensions that advocate ARC programming methodology.

The local coordinator runs on a machine that participates in the ARC system either to improve its utilization or to share its work with an anonymous remote node. Any ARC communication to or form the local processes is achieved through the local coordinator are described in the sub-sections.

An online command can be executed that directs the local coordinator to obtain a particular amount of work from the ARC system .this generates an asynchronous intimation that travels to the remote programs. The results are forwarded to the local coordinator. A task executor may also generate new RIBs.

It is implemented on a heterogeneous as an extension to C language. A distributed ARC kernel is spread over the workstations that participate in anonymous remote computing. It is local activity and global activity it consists of these primitive calls.

ARC implementation is built over the security provided by operating systems over ARC is implemented. The system coordinator accepts connections only from local coordinators on trusted clients. The original of the conventional COP model of parallel programming on workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. A remote local coordinator accepting RIPs arriving from anonymous runs with specific user permissions especially created for the purpose of ARC computations. Portability to ARC programs nodes primitives are made portables.

Outline of ARC Runtime Support

A distributed ARC kernel is spread over the workstations that particular in anonymous remote computing. The entire domain is partitioned into three logical clusters of multiple local variables.

One of the machines in the pool is selected to run daemon co-ordinates the all of the machines participate in the pool. This is termed system co-coordinator .An RIB is a code fragment that can be migrated into a convenient anonymous remote node at runtimes for execution.

This call is used to obtain information about various machines available in the system and their loads. The parameter to this system the number of machines required by the program. The return value is a structure the values to identify the system . The parameter that calls supported.

Outlines of ARC runtime support Performance system level optimizations by the parallel programming platform can improve the performance of applications. The system can achieve load balancing for the application, further enhancing performance. The run-time scheduling decisions by the system.

The integrated platform A program is said to be load adaptive if adapts to the changing load in the system. A programmer may not able to use the knowledge of load fluctuation in the program unless an adequate language support is provided. the value is time-out and retries parameters permit a re-execution ,the RIB may be resent to another suitable remote node.

A sample block in the integrated platform programmers concerns and systems concerns and account for heterogeneity fault tolerance, load adaptability and processor availability. The original of the conventional COP model of parallel programming on workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. To execute remain anonymous to the program. At the time multiple programs can be generating RIBs and multiple anonymous emote participants

DESIGN AND IMPLEMENTATION

The parser for by the user into the coordination of the pool of workstation and functional library support to avail system services are the elements of the system

Parser

The Parser for program submitted by the user into the final coordination. The original of the conventional model of parallel programming on workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. To execute remain anonymous to the program. At the time multiple programs can be generating RIBs and multiple anonymous emote participants can be joining or leaving the system

Local coordinator

It basically inter connected workstations with distributed file system and standardization of software are common uses. The binary executables files are not compatibles between architectures, using several difficulties in parallel programming on

workstations clusters in messages passing or shared memory abstractions. The architectural differences between workstations are more difficult to handle than operating system differences. The binary executable files are not compatibles between architectures. Using common NFS network.

System coordinator

These are intended for providing distributed services. Some of them have been proposed for parallel programming other are intended for providing distributed services. we analysis the suitability of these approaches from the parallel programming point of view. This model is many rises to many difficulties while programming on this system.

The original of the conventional COP model of parallel programming on workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. .this generates an asynchronous intimation that travels to the remote programs. The results are forwarded to the local coordinator. . A program is said to be load adaptive if adapts to the changing load in the system. A programmer may not able to use the knowledge of load fluctuation in the program unless an adequate language support is provided

APPLICATIONS

Applications with coarse grain control parallelism or coarse grain data parallelism or both are the target of our platform. to use high-level programming language constructs are supported. workstation cluster and scientific computing in academic are becoming increasingly popular The processing power of the workstation has witnessed tremendous growth resulting in clusters of workstations. parallel programming on workstation systems is a relatively new field and has been an attractive proposition ever since it was used.

Several key issues distinguish parallel computing on work station clusters. Resent advantages in communication technology and processor technology and processor technology make parallel programming on loosely coupled distributed systems of tightly-coupled massively parallel architecture.

A considerable amount of unused computing capacity is always present in the network. A program is said to be load adaptive if adapts to the changing load in the system. A programmer may not able to use the knowledge of load fluctuation in the program unless an adequate language support is provided. It can challenge tasks. A program can be load adaptive it changing load in the system.

It based on object oriented programming systems are call asynchronous systems. It based on non-blocking fashion. Method calls return immediately in a non-blocking fashion. Either a future mechanism is used or a call back mechanism is implemented to obtain the return value of a call.

Two computing entities exchange messages such as arbitrary data values. Message communication is blocking. Communicating sequential processes is based on synchronous system.

The lower layer provides primitives to support system such as the creation of fault tolerance, load sensing and source code migration. It based on fault tolerance, load sensing and source code migration.

The requirements come from the desired expressibilty of the application, possible transparency in programming, exploitation of parallelism.

PERFORMANCE ANALYSIS

This presents performance related aspects of the work. The test bed for the experiments consists of a immediately and can further accept a new set of arguments. A specialized RIB is prepared from the generic RIB by integrating the required code fragments into it. The processing power of the workstation has witnessed tremendous growth resulting in clusters of workstations. Parallel programming on workstation systems is a relatively new field and has been an attractive proposition.

A spite of a task division policy based on run time load conditions the completion time of the tasks the original of the conventional COP model of parallel programming on

workstation systems can be traced to the process based multi-programmed machine with various inter-process communication mechanisms. A remote local coordinator accepting RIPs arriving from anonymous runs with specific user permissions especially created for the purpose of ARC computations.

The problem is a case the task and data parallelism is complementary. The control A programmer may not able to use the knowledge of load fluctuation in the program unless an adequate language support is provided. It can challenge tasks. A program can be load adaptive it changing load in the system. A load on a particular machine in the analysis.

The Data parallelism refers to the simultaneous execution of the same instruction stream on different data elements. post the RIB on the lock obtained ,the results of the RIB execution start-up and close-down primitives, primitive support for RIBs, primitives for packaging primitives for RIB posting ,primitives for parameter setting, horse power factor and asynchronous intimations.

It can be seen that exploitation of both task and data parallelism, six nodes are utilized for parallel execution before request arrives, a statistic-daemon is contacted to access the current load information and the normalized speed of the machine, the HPF is computed and returned. A lock request improve its utilization size of four minutes is reached.

POSSIBLE QUESTIONS(2 Marks)

1. List the issues in Parallel Computing on Interconnected Workstations.
2. Define ARC.
3. What are the two properties of RIB?
4. What is Fault Tolerance?
5. Write two benefits of ARC Model.
6. Define HPF.

7. Write the difference between Blocking and Non Blocking ARC Calls.
8. What is System Coordinator?
9. What is Local Coordinator?
10. Write a note on Lock Management.

POSSIBLE QUESTIONS(6 Marks)

1. Discuss the issues in Parallel Computing on Interconnected Workstations.
2. Explain the design of Upper Layer ARC Constructs.
3. Describe the existing Distributed Programming Approaches.
4. Discuss the implementation of Two Tired ARC Language Constructs.
5. Differentiate between ARC and COP models.
6. Discuss the model for integrating Task Parallelism into Data Parallel Programming Platforms.
7. Brief about RIBs and Synchronizers.
8. Describe ARC Model of Computation and ARC Runtime Support.
9. Explain the design of Lower Layer ARC Interface.
10. Discuss the performance analysis of Control Parallelism and Data Parallelism in ARC.

Karpagam Academy of Higher Education

Department of CS, CA & IT

Subject: Grid Computing(17CSP205B)

Batch : 2017-2019 **Class: I M.Sc CS**

Objective Type Questions

UNIT II						
S.NO	**QUESTIONS**	**OPTION 1**	**OPTION 2**	**OPTION 3**	**OPTION 4**	**KEY**
1	PVM stands for_____	Parallel virtual machine	Program virtual machine	Primitive virtual machine	Process virtual machine	Parallel virtual machine
2	RPC stands for_____	Remote program call	Remote procedure call	Root process call	Remote procedure computing	Remote procedure call
3	COP stands for_____	Computing of process	Collection of program	Collection of processes	Computing of processes	Collection of processes
4	A _____ facility allows for migrating code fragments to remote servers for	Remote execution	Program process	Remote function evaluation	Parallel computing	Remote execution
5	MPA stands for_____	Message passing assembly	Message passing abstraction	Message program absolute	Message program abstraction	Message passing abstraction
6	MPVM stands for_____	Message program virtual machine	Machine parallel virtual mode	Message parallel virtual machine	Message process virtual machine	Message parallel virtual machine
7	_____are provided for handling variation in loads & speeds	Primitives	Parallel computing	Remote instruction block	Grid computing	Primitives
8	The runtime system known as the _____ system decides the nods on	MPA	ARC	RIB	RPC	ARC
9	RIB stands for_____	Remote instruction block	Remote institute block	Remote index block	Remote identify block	Remote instruction
10	_____ extends the computing entities in the horizontal domain	RFE	RIB	ARC	HPF	ARC
11	DSMS stands for_____	Distributed shared memories	Distribution state memories	Domain system memories	Domain shared memories	Distributed shared memories

12	_____ are mechanisms that are needed to achieve synchronization between	Parallel computing	Procedure call	Primitive	Synchronizers	Synchronizers
13	REF stands for_____	Remote free evaluation	Remote function	Remote function	Remote factor evaluation	Remote function
14	ARC provides an abstraction called_____	Horse power factor	Remote function call	Collection of process	Parallel process	Horse power factor
15	If the anonymous node is heterogeneous the source code of the ____ needed to be	RFE	HPF	RIB	ARC	RIB
16	The distributed object oriented programming systems are	Call asynchronous system	Producer call	Remote execution	Noise power factor	Call asynchronous system
17	_____ is implemented to obtain the return value of a	Primitive	Synchronizers	Call back mechanism	Parallel virtual machine	Call back mechanism
18	FDDI stands for_____	Function distributed domain interface	Free distributed data	Fiber distributed data interface	Function divided data interface	Fiber distributed data interface
19	_____ types of COP are available in distribution	9	7	6	4	6
20	The …….primitive is used to set the control parameters for a	SETPARAM ()	obtain result ()	post RIB ()	open data pack()	SETPARAM ()
21	The primitive …….is used to obtain the results of an earlier Rib posted for	post Rid()	set param ()	OBTAIN RESULT ()	open data pack()	OBTAIN RESULT ()
22	the time out value may be incremented with the help the help	SETPARAM()	obtain result()	post Rib()	open data pack()	SETPARAM( )
23	if an anonyms node cannot be contacted due to its unavailability or due to the time-	REM-FAILURE	failure-rem	post Rib()	insert data pack()	REM-FAILURE

24	the…….primitive is used to post on RIB to yhe ARC kernel which can allocate the RIB to an earlier	obtain result ()	POST RIB()	set param()	open data pack()	POST RIB()
25	the…….primitive returns a new data-pack handler in which an artitary number of arguments can be	OPEN DATA PACK()	set param() (	post Rib()	obtain result()	OPEN DATA PACK()
26	a continuous packet is prepared for all the inserted arguments by calling the	obtain lock()	open data pack()	close-down()	CLOCK DATA PACK()	CLOCK DATA PACK()
27	the ARC ……..primitive unlinks the program	CLASS DOWN()	close data pack()	open data pack()	obtain lock()	CLASS DOWN()
28	the stacks values……is returned if the time out is reached but the	WORM-IN- PROGRESS	rem-failure	work progress	infinity	WORM-IN- PROGRESS
29	the abbservation for HDF is…….	rem-failure	HORSE POWER FACTOR	work progress	infinity	HORSE POWER FACTOR
30	the …… primitive is used to secure a lock on an anonyms node & obtain the HPF for the	close data pack()	close-down ()	OBTAIN LOCK	set param()	OBTAIN LOCK
31	a call to the …… primitive returns a value TRUE if an synchronyms	INTIMATION RECEIVED()	obtain lock()	load adaptability	post RIB()	INTIMATIO N RECEIVED()
32	…….is a major concern for ARC	post Rib	intimate received	work-in- progress	LOAD ADAPTABILITY	LOAD ADAPTABILITY
33	The……..function call is an example of an low level non-	POST RIB()	obtain lock()	intimation received	close-down	POST RIB()
34	The…….. mechanism is provided as an extension to c	Arc	set param()	obtain lock()	post Rib	Arc
35	RIB are extracted from the programs by using an…..compiler	obtain lock()	set param()	ARC	post Rib	ARC
36	The abbservation for HPU is……	HORSE POWER UTILIZATION	horse power utilize	horses power utilization	home power utilization	HORSE POWER UTILIZATIO

37	the default value for time out & retries can be chosen	zero&one	INFINITY&ZERO	null	null&infinity	INFINITY&ZERO
38	the……figure is used for deciding on execution in case of a	infinity	close down	TIME OUT	ARC	TIME OUT
39	ARC model was implemented on a network of……..	HETEROGENOUS WORKSTATION	homogenous workstation	hetro workstation	homo workstation	HETEROGENOUS WORKSTATION
40	XDR stands	External data	External	External Data	External	External data
41	MPMD stands	Multiple	multiple	Message	Message	Multiple
42	_____is a set type of	Collection of	NOW	DP	Parset	Parset
43	When the Elements of	Simple Parset	Function	Cluster	untyped Parset.	Simple Parset
44	_____consist of a	procedure call	Parset	ARC	Server	Parset
45	____refers to the simulataneous execution of same	Data parallelisim	Control parallelisim	parallelisim	Control parallelisim	Data parallelisim
46	Data Parallelisim refers to the simulataneous	data stream	instruction stream	Control parallelisim	ARC	instruction stream
47	MPI stands for_____	Message Passing	Message	Mobile	Mobile Passing	Message
48	Sharing the task	GCI	MPI	CORBA	Grid	Grid
49	The Combination of	Mobile Cluster	Cluster	GridComputi	ARC	Mobile
50	MCC stands for	Mobile cluster	Message	Mobile	Mobile	Mobile
51	_____is a good tool	Simulated	Mobile	Distributed	JSS	Simulated
52	_____Belongs to the	Travelling	Simulated	Job Shop	ARC.	Job Shop

UNIT-III

SYLLABUS

Anonymous Remote Computing and Communication Model: Location – Independent Inter-task Communication with DP – DP Model of Iterative Grid Computations – Design and Implementation of Distributed Pipes. Parallel Programming Model on CORBA: Notion of Concurrency – System Support – Implementation and Performance

INTRODUCTION

The effective parallel solution of problems on NOWs requires the runtime selection of nodes. The granularity of individual subtasks may have to be deferred until runtime for load balancing. Dynamic schemes may be needed to make the programs resilient to changing conditions. Support for inter-task communication high-level parallel programming platforms which support runtime and dynamic policies, leads to several issues. Some of these issues are as follows:

In ARC, the node to which a sub-task is migrated is decided at runtime. The nodes remain anonymous to the user program which initiates the migration. Thus, a sub-task would be unaware of the location of other sub-tasks in order to communicate with them.

Another issue pertains to the dynamic schemes employed by the platforms. The system support may migrate an already running task, reshuffle the load allotted to individual sub-tasks, etc. Such dynamic policies detach processes from specific nodes. Hence, communication primitives which assume the location of processes are unsuitable.

Transparent inter-task communication will facilitate a number of application domains to exploit the parallel computing power of clusters of workstations. Some of these application domains are as follows:

Iterative grid computations comprise a large class of engineering applications. When the domain of computation of an iterative grid computation problem is divided, the sub-domains will need to exchange their boundary values. Grid computations are used to solve problems, such as elliptical partial differential equations by finite differences [8].

Parallel solutions of the class of sub-optimal algorithms like simulated annealing are discussed in [9]. The problem partitioning adopted requires the sub-tasks to exchange their intermediate results.

Some problems can be partitioned for parallel solution as a network of filters. Networks of filters can be used to solve a variety of programming problems. Reference [10] describes a prime number sieve and a matrix multiplication network using this pattern. Such problems would also require the communication of intermediate results.

The current work explores the transparent programmability of communicating parallel tasks on loaded heterogeneous workstations

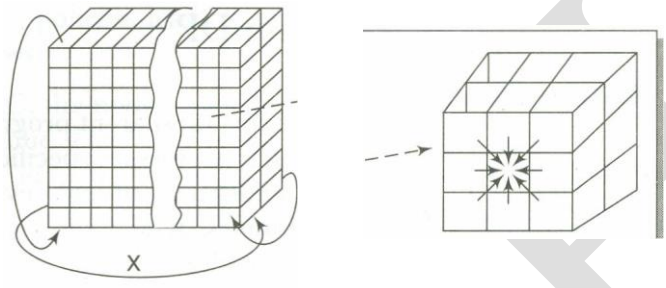
LOCATION INDEPENDENT INTER TASK COMMUNICATION WITH DP

Distributed Pipes (DP) is a model for transparent programming of communicating parallel tasks. It addresses issues specific to parallel programming on NOWs. DP provides a set of high-level location- transparent communication primitives, which support data flow between processes that are independent of their location. This enables the model to accommodate anonymous migration of communicating parallel tasks.

In the model, the communication channels between the nodes of a network are considered as global entities. The information pertaining to them is maintained globally on a designated node. A communication channel is created or deleted at runtime. Communicating parallel tasks created at runtime can be connected by using DP. The high-level abstractions of DP provide an elegant set of programming interfaces that are free from low-level network details. This contributes to the readability and maintainability of the code. Programs in the model are not tied up to specific nodes. Hence, it accommodates a changing pool of workstations, and relieves the programmer from the task of programming for specific machines, thereby rendering the resultant code portable to a different network. The DP provides a uniform set of interfaces for communication across heterogeneous nodes. This addresses heterogeneity among the nodes in both architecture and the operating system. DP uses the external data representation to handle heterogeneity. The programming level abstractions of DP wrap TCP abstractions. Message sizes exceeding the size limit imposed by TCP are handled transparently by splitting and coalescing the message appropriately.

DP MODEL OF ITERATIVE GRID COMPUTATIONS

Figure 5.1 shows the nature of a typical iterative grid computation. The iterative marching in space and time dimensions are shown in the illustration on the left in Fig. 5.1. The expanded grid on the



Grid Computation Problem

right of Fig. 5.1 shows the boundary value exchanges. The typical program structure of the problem for sequential execution is as follows:

Pseudo Code 1: Program Structure of typical iterative grid computations

```
FOR Time = StartTime TO EndTime
FOR XAxis = StartX TO EndX
FOR YAxis = StartY TO EndY
UserDefinedFunction()
END FOR
```

ENDFOR
END FOR

The outer loop of the pseudo code marches in time and the inner loops march in each dimension of space.

5.3.1 The Model

The model employs a master-worker model of computation. The program for the model consists of a master process and several worker processes. The master process is the process which initiates the computation. Worker processes are spawned on the nodes which participate in parallel computation. Worker processes are called Iterative Grid Modules (IGMs).

The model accomplishes parallel execution of the problem by Domain decomposition. Each IGM is allotted a sub-domain of Computation. The boundary value exchange of values between IGMs is effected through DPs. IGMs return the results of their Computation to the master process. The model permits communication between anonymously migrated IGMs.

In the model, the system handles domain decomposition, selection of least loaded nodes, load balanced division of tasks, anonymous migration of IGMs, transparent laying of communication primitives between IGMs, result collection, and aspects related to fault tolerance.

The model offers various advantages. The programs in the model are not tied up to specific machines. Such programs can accommodate a changing pool of workstations. It

also makes the programs portable to a different network. The number of IGMs is not decided a priori. Hence, the system can utilize the optimum number of nodes according to the runtime conditions. The programs written for the model can tolerate a heterogeneous Collection of unevenly loaded workstations. The programs in the Model are devoid of any underlying network code. This results in greater readability of the program and, hence, in greater Maintainability.

Figure 5.2 illustrates the parallel solution of a grid computation Problem using the model. In Fig. 5.2, thick circles represent IGMs, the thin circle represents the master process, ellipses represent runtime daemons, thick lines represent TCP connections, thin lines represent Unix Domain socket connections, and dashed lines depict DPs between IGMs. Grid Computation Tasks, (GCTs) represent IGMs and GCP (Grid Computation Problem) represents the Master Process.

Initialization

The master process and IGMs need to register with the system in Order to avail of system services. The master process registers with the system by using the call Initialize Work(). Upon completion, a complementary call Close Work() is used. IGMs register with the system by using the call InitializeIGM() Upon completion, a complementary call closeIGM() is used.

Domain Decomposition

In the model, the master process sends the grid information to the system. The system decides the optimum number of IGMs to be employed, the granularity of computation to be allotted to individual IGMs, and the nodes to be assigned for each IGM. The master process gathers

this information from the system and packs initial data for individual IGMs to migrate the IGMs and to create channels that collect results from each IGM.

The master process provides the grid information to the system by using the call SendGridlnfo(). Similarly, it obtains the number of IGMs employed by using the call Obtain NumberOfSplits(). The granularity of each IGM is obtained by using the call ObtainSplitlnfo()

A brief description of the calls is given below.

- Int SendGridlnfo (int Work/d, int SpaceInX, int SpacelnY, int SpacelnZ, int History, int SplitDirection).

SendGridlnfo () sends the grid information of a grid computation work to the system. Workld is the index by which the system identifies a grid computation work.

SpaceInX, SpaceInY, and SpacelnZ are the number of grids in X, Y, and Z dimensions of space.

History specifies the number of previous time slices to be stored.

SplitDirection specifies the direction of the split.

- int ObtainNumberOfSplits(int WorkId).

ObtainNumberOfSplits() collects the number of IGMs for the grid computation work denoted by the WorkId .

- int ObtainSplitInfo(int Work/d, int SplitId, int" Start, int" Total).

ObtainSplitlnfo () gathers information related to a split of the work. The starting grid of the sub-domain and the number of grids in the sub-domain are stored at the addresses pointed to by Start and Total; respectively.

Load Balancing

The system gathers availability and load information of the nodes in the network in order to decide the optimum number of IGMs to be employed and their individual granularities. Machines with load indices higher than a designated value are ignored. The domain of computation is sub-divided among the other machines. The granularity of individual sub-domains depends upon the load ratio of the machine.

Our approach to load balancing offers several advantages. The actual means of gathering and interpreting load information on the participating machines are hidden from the programmer. In a heterogeneous collection of workstations, the processing power of individual nodes is also used for load balancing. In our approach, the programmer is relieved of the task of specifying the ratio of the processing power in a collection of heterogeneous nodes. The load balancing scheme may have to be altered to accommodate different types of nodes or to prune the load interpretation mechanism. In such cases, the user programs need not be modified in order to change the load balancing scheme.

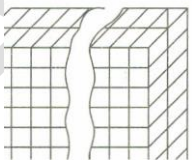
Anonymous Migration of Sub~tasks

The anonymous migration of an IGM is initiated when the master process invokes the Migrate() call. However, the master process does not furnish any machine specific arguments to the Migrate() call. The information required for an IGM to initialize its data structures as well as the initial data for the IGM are the parameters to the call. These are retained with the local lc of an IGM until the IGM claims them. The syntax and semantics of the call are given below.

- int Migrate(int Work/d, int Splitld, char;' MigrateFile, int DataType, ooid" Data, int SpacelnX, int SpacelnY, int SpaceJnZ, int History. char'!' ResultPipe}.

Migrate() migrates the code for an IGM and provides it with initial data. The data consists of the number of grid points in X,

Y, and Z dimensions of space. ResultPipe is the name of the DP to which the IGM writes its results.



Information Gathering by IGMs

The lc which collects anonymously migrated IGMs, compiles the IGM code and spawns the IGM process. An IGM process has to initialize its data structures to hold the initial data, and collect the initial data with which to begin computation, the position of the IGM, and the name of the Result Pipe to write its result. The size of the initial data is required to initialize the data structures. This is facilitated by the call ObtainTask-Gridlnfo(). After initializing the data structures, the IGM collects the initial data by invoking the call ObtainTaskData (), The call ObtainTaskMachinelnfo() provides the position of the IGM process and the name of the Result Pipe to be opened.

In our approach, the information pertaining to an IGM is not tied up with the code of the IGM. At runtime, the system provides information relevant to individual IGMs. The syntax and semantics of the calls are given below.

- int Obtain'Tasktlridlnfotint" SpacelnX, int" SpaceInY, int" SpaceInZ, int" History).

ObtainTaskGridlnfoO provides the number of grid points in X, Y, and Z dimensions of space.

- int ObtainTaskData(void* Data, int SpaceInX, int SpacelnY, int SpaceInZ, int History).

ObtainTaskDataO stores the initial data matrix at the address pointed to by Data.

- int Obtain TaskMachinelnfo (int" WhichMachine, char" ResultPipeName).

ObtainTaskMachineInfoO stores the location of the IGM in the grid computation work at the address pointed to by WhichMachine.

Transparent Communication

Each IGM has to communicate with its neighbouring IGMs to exchange boundary values. Support for communication between IGMs brings with it two issues. Since the IGMs are migrated to

anonymous nodes, an IGM will not know the location of its neighbouring IGMs. The second issue pertains to the position of an IGM. The number of neighbours of an IGM depends upon the position of the IGM. Hence, the number of DPs to be opened by the IGM cannot be known until runtime.

In the model, an IGM collects information about its neighbours and the number of DPs to be opened at runtime.

This is facilitated by the calls

ObtainTaskOpenPipeNames ..
and
ObtainT askNoOfPipes ToBeOpened

Following is a description of the call:

- int ObtainTask OpenPipeNames (char """ PipeNames, int" AeeessMode, int NoOfOpenPipes).

ObtainTaskOpenPipeNamesO provides the number, names and access modes of the Distributed Pipes to be opened.

- int ObtainTaskNoOfPipesToBeOpened(int* NoOfOpen-Pipes).

ObtainTaskNoOfPipesToBeOpenedO stores the number of Distributed Pipes to be opened at the address pointed to by N oOfOpenPipes.

DESIGN AND IMPLEMENTATION OF DISTRIBUTED PIPES

Runtime Support

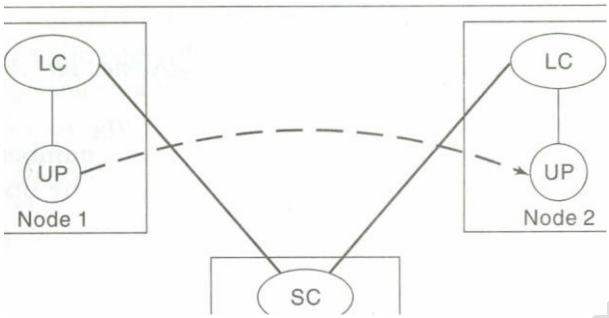
The runtime support consists of an Ie daemon running on each node participating in parallel computation and a se daemon on a designated node.

Figure 5.4 illustrates the overall structure of the system. Circles represent the user processes, ellipses represent runtime daemons, thick lines represent TCP sockets, thin lines represent Unix Domain sockets, and dashed lines represent Distributed Pipes.

Local Coordinator (lc)

The lc runs on each node that participates in parallel computation. The lc services requests generated by user processes on its node. Also, it maintains information required to coordinate the user processes.

The lc maintains two tables to support bare DP services, namely, the User Process Information Table (UPT) and the User Processes Blocked for Write Table (UPBWT). The UPT maintains information pertaining to user processes which have registered with the lc on its node. UPBWT keeps track of processes which



Node 3

have opened a DP in write mode and have not been opened by any other process for reading. The writing process is blocked until the DP is opened by some other process in read mode. The lc maintains the table in order to inform the blocked processes when another process opens the DP in read mode.

In order to support grid computations, the lc maintains a Grid Computation Task Submitted Table (GCTST). The ic uses the GCTST to service the requests of a task. The table is indexed by the process id of the task. The GCTST is updated either when a new task is submitted or when an already submitted task terminates. If the service needs additional parameters, the lc forwards the information to the sc.

The FSM of lc is given in Fig. 5.5. In the INIT state, the tc initializes its data structures and cleans the auxiliary system files. The lc establishes a TCP connection with the sc and registers with the sc. In the LISTEN state, the lc waits for messages from the sc or any user process. When it receives a message from the sc, it changes its state to SC Msg RECVD and services the message

When it receives a message from a user process, it changes its state to UP Msg RECVD and services the request.

The initial communication between a process and the lc is through a known common channel. This is required for a user process to register with the lc. User processes which register with the lc are given exclusive communication channels for subsequent communication.

System Coordinator (sc)

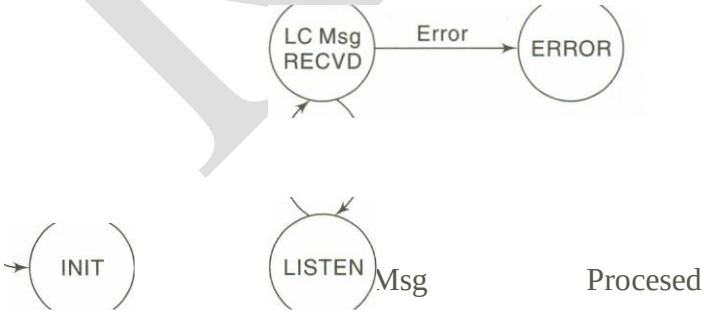
The sc coordinates the lcs in the pool. Also, the sc keeps track of individual lcs and facilitates communication between them. The sc is connected to lcs through TCP sockets. The sc maintains TCP socket descriptors which connect it to individual les.

The sc maintains two tables, namely, the Distributed Pipes Table (DPT) and the Local Coordinators Table (LCT). The DPT keeps track of the DP channels. The table is updated when a DP is created, opened, closed, or deleted. When a process opens a DP to write to, before the pipe is opened for reading, the corresponding lc information is also stored in the DPT. Thus, the process can be intimated when some other process opens the DP in read mode.

The LCT keeps track of the les in the system. The table is updated when a new lc joins the pool or when an existing lc leaves the pool.

The sc maintains two additional tables in order to support grid computations, namely, the Grid Computation Work Table (GCWT) and the Grid Computation Task Table (GCTT). The GCWT maintains information pertaining to grid computation work that is submitted to the se. It is updated either when a work is submitted to the se or when a work is completed. The GCTT maintains information pertaining to individual tasks that constitute the grid computation work. The table is updated when the work is sub-divided into tasks, when a task begins execution, or when a task terminates. The GCTT is a part of the GCWT. I

The FSM of se is given in Fig. 5.6. In the INIT state, the sc initializes its data structures and cleans the auxiliary system files. In the LISTEN state, the sc polls for connection requests from the les. When a connection request from an lc is received, it registers the lc with the system and establishes a TCP socket connection between them. It then listens for messages from the registered les on exclusive channels, and continues to listen for new connection requests. When a message from an lc is received, it changes its state to LC Msg RECVD and processes the message. Once the message is processed, it returns to the LISTEN state.



Initialized

Functional Library Support

The functional library support consists of services to support location-transparent communication with DPs and services to support the DP model of iterative grid computations. Variants of the calls are provided to support communication across heterogeneous architectures, by utilizing the external data representation. The library is built over TCP and Unix domain stream protocol.

Basic Distributed Pipe (DP) Services

The following are the basic DP services:

- int CreateDistPipe(char* PipeName).

CreateDistPipe() initiates a message to the sc through the lc running on its machine. The sc creates the DP if another channel with the same name does not exist and makes an entry in the DPT.

- int OpenDistPipe(char>:' PipeName, int AccessMode).

OpenDistPipe() initiates a message to the sc through the lc running on its machine with the name of a DP as a parameter. The sc completes the message sequence by informing the user process if the DP is created or not.

Corresponding to an open request in Write Mode, a TCP socket is created and another message sequence is initiated

by sending a message to sc through lc: The message contains the TCP socket descriptor, access mode, and process id of the requesting process. The sc updates DPT with this information. If the DP is already opened by another process in Read Mode, the information of the read process is returned to the caller. The open call uses this information to connect to the read process. If the DP is not opened for reading, it causes the update of DPT at se and tJPBWT at lc. Subsequently, the call blocks until it receives a message from the lc intimating the information of read process.

Corresponding to an open request in Read Mode, a TCP socket is created and bound to a local port. A message is generated to the sc to update the DPT with the TCP socket descriptor, port number, access mode, and the process id. Further, the sc intimates the user processes which have requested to open the channel in Write Mode with the details of the read process. The call listens on the TCP socket for connection requests.

- int ReadDistPipe(int PipeDescriptor, char;' Buffer, int BufferSize).

A ReadDistPipeO call translates to the read system call. It reads from the socket descriptor for the DP descriptor. The PipeDescriptor returned is the actual socket descriptor in order to make it a direct translation. Hence, the read call

does not cause any overheads. The call handles message sizes exceeding the limits of TCP messages.

- int WriteDistPipe(int PipeDescriptor, char':' Buffer, int BufferSize).

A WriteDistPipeO call translates to the write system call. It writes to the socket descriptor for the DP descriptor. The PipeDescriptor returned is the actual socket descriptor in order to make it a direct translation. Hence, the write call does not cause any overheads. The call also handles message sizes exceeding the limits of TCP messages.

- int CloseDistPipe(int PipeDescriptor)

CloseDistPipe() call translates to the close system call. It closes the socket descriptor for the DP descriptor. Further, the call initiates a message to the sc through the lc with the name of the DP and process id as parameters. This causes the DPT table at the sc to be updated.

- int DeletellistlPipetchar* PipeName).

DeleteDistPipe() initiates a message to the sc through lc with the name of the DP as an argument. In response to the message, the sc deletes the corresponding entry in DPT and returns the deletion status to the call.

Overhead of Interfaces

The overhead of each call is caused by the message sequences initiated by the call. The calls CreateDistPipe, CloseDistPipe, and DeleteDistPipe result in a message to the le on the node, a message from the lc to the sc over the network, a reply from the sc to the lc over the network, and a message from the le back to the user process. The call OpenDistPipe constitutes two such message sequences and, hence, twice the overhead. The typical size of data-packets exchanged is around 100 bytes. The round trip time of communication over network could range from 0.4 milliseconds (ms) to a few milliseconds. Typically, the average round trip time is less than two milliseconds. However, these calls are used only once during the lifetime of a DP. Hence, these over heads become insignificant. The calls ReadDistPipe and WriteDistPipe are directly translated to the underlying system call. Hence, they do not incur any overheads. These calls are used many times during the lifetime of a DP.

PARALLEL PRORAMMING MODEL ON CORBA

Extended Services for IOC

The extended sevices for IGC are as follows:

- int InitializeGridComputation Work()·

InitializeGridComputation Work initiates a message to the sc through lc. The sc creates an entry for the work in GCWT and returns the WorkJd .

- int SendGridlnfo(int WorkJd, int SpacelnX, int SPacelnY, int History, int SplitDirection).
-

SendGridlnfo initiates a message to the sc through lc.
The message carries the arguments to the call. The se updates
the information in GCWT and returns the updated status.

- int ObtainNumberOfSplits(int WorkId).

ObtainNumberOfSplits initiates a message to the sc through lc. The sc, in response to the message, collects the load information of all machines from the corresponding lcs. This information is used by the sc to split the work. Further, the sc creates a new entry in GCTT to store the information about the split and returns the number of splits.

- int ObtainSplitlnfo(int WorkId, int SplitId, int *Start, int *Total).

ObtainSplitInf() initiates a message to the sc through the lc. The sc gathers the information from GCTT and returns the starting and total number of grids for the split.

- int Migrate(int WorkId, int SplitId, char *MigrateFile, int DataType, void *Data, int SpacelnX, int SpacelnY, int History, char *ResultlPipe).

Migrate initiates a message to the sc through lc with its WorkId and SplitId. The sc gathers the information of lcs from GCWT and GCTT and sends messages to them. In response to the message, each lc creates a TCP socket, binds the TCP socket to a local port, and listens on it. Also, the port numbers are returned to the sc. The sc passes the collected information to

the lc which initiated the migration. The lc which initiated the migration makes a TCP connection and transfers the code, data, and result DP name to the other lc. The GCTST of the migrated lc is updated by using this information. Once the migration is over, the TCP connection is closed and the sc is informed.

- int CloseGridComputation Work(int WorkId).

CloseGridComputation Work initiates a message to the sc through lc with WorkId. In response to the message, the sc purges the corresponding entry from GCWT.

- int InitializeGridComputationTask.

InitializeGridComputationTask initiates a message to the sc through the lc. In response to the message, the sc updates GCTT with the new task entry and returns TaskId.

- int ObtainTaskGridInfo(int *SpaceInX, int *SpaceInY, int *History).

ObtainTaskGridInfo initiates a message to the sc through the lc. The message contains the process id of the task. The lc collects the relevant information from the GCTST.

- int ObtainTaskMachineInfo(int *WhichMachine, char *ResultPipe Name).

ObtainTaskMachineInfo initiates a message to the sc through the lC. The sc returns the position of the subdomain for

which the task is responsible to the lC. The lc returns this information and the ResultPipeName to the task.

- int ObtainTaskData(void *Data, int SpaceInX, int SpaceInY, int History).

ObtainTaskData initiates a message to the lc. The lc gathers the information from GCTST and returns it to the task.

- int ObtainTaskNoOfPipesToBeOpened(int *NoOf OpenPipes).

ObtainTaskNoOfPipesToBeOpened initiates a message to the sc through the lc. The process id of the task is passed along the message. The sc gathers information from the GCWT and returns the number of DPs to be opened by the task.

- int ObtainTaskOpenPipeNames(char **PipeNames, int *Accessidode, int NumberOfOpenPipes).

ObtainTaskOpenPipeNames initiates a message to the sc through the lc. The process id is passed along with the message. In response to the message, the sc returns the names of DPs to
be opened and their Access Modes.

- int CloseGridComputationTask(int TaskId).

CloseGridComputationTask initiates a message to the lc. The lc deletes the corresponding entry from the GCTST and

forwards the message to the sc. In response to the message, the sc updates GCWT.

NOTION OF CONCURRENCY

This section explains the notion of concurrency that is used in P-CORBA. It also gives the application developer's view of the model. The basis of this notion of concurrency is given in the OBS model [19].

The notion of concurrency is based on the subcontract mech-anism which models concurrency at the method level. A method can be invoked concurrently on multiple objects through the subcontract. The key idea in achieving the sub-contract is the metaobject. The meta-object is an entity which can aggregate objects of the same class. Meta-objects can be created by instantiating a meta-class called Parclass. The application developer can insert objects into the meta-object and then invoke the sub-contract on the meta-object. This ensures that the method is invoked on all these objects simultaneously.

User Interface

The sample code in Fig. 6.1 is for solving a simple parallel matrix multiplication problem to illustrate the user's view of the model. It shows the syntax of the user program in the model. The matrix class has methods to initialize the matrix, to compute the partial product, and to print the results of the partial computation. Each matrix class computes a portion of the product. In this program, results are not returned to the meta-object for the sake of simplicity. In the main body of the program, the meta-class and objects of the class matrix are instantiated. As shown in the code, the objects can be inserted into the

meta-object through the insert (symbol < <) operator. After this step, the sub-contract call on the meta

```
class matrix {
public :

voild Initialize ( .. ); // Initializes
the matrix
void Inverse ( .. ); // Matrix Inversion
void PrintMatrix( .. ); // Prints the
matrix
} ;
main () {
Parclass ParMatrix holds matrix;

// metaclass Parmatrix can hold objects
of class matrix
instantiating
metaclass

Parmatrix
matrix ml,
P; II
m2;
ml. Initialize( .. );
m2. Initialize ( .. ) ;
P « ml;
```

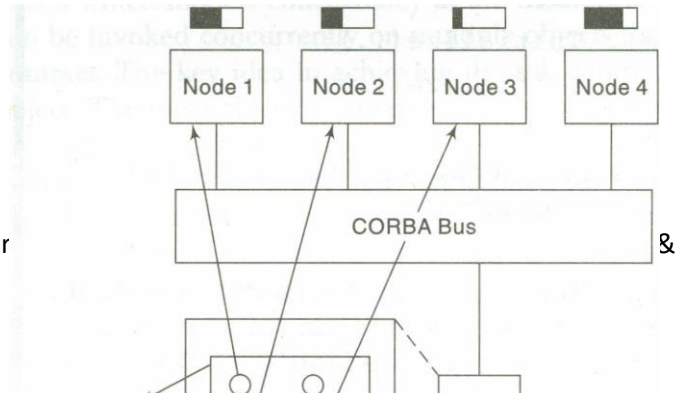
```
// Insert operator, insert matrix ml
into metaobject
P « m2;

P .. Inverse (U) ; // subcontract operator,
call for parallel execution of Inverse on
all objects of the metaobject P (U lS
unordered)
P .. PrintMatrix(O);

// print matrices in order, no concurrency
here (0 is ordered)
```

Object is made through the subcontract (symbol..) operator. This ensures that all the objects do the partial computation in parallel.

A runtime system ensures that the objects that are part of a sub-contract can be migrated to the best available machines ' based on the load conditions (Fig. 6.2). The notable point is that the application developer partitions the data by creating the different objects of the same class and inserting them into the meta-object. The runtime system decides on which machines these objects are to execute at runtime. This concept illustrates how cleanly the functions of the system and the programmer are separated in the model. The meta-object also abstracts the concept of fault tolerance in the sense that if any of the method call fails, it is re-executed on a different node



Load Factor

% of fill indicates the load

In the OBS model, support for inter-object communication is not provided. So computations involving communicating tasks cannot be modelled. The authors have introduced a new operator called 'message sender' (symbol 0 ~ 0) on the meta-object through which a message can be sent from one object to another object in the meta-object's collection. The sample code in Fig. 6.3 illustrates how grid computations like the ones discussed in [10] which involve communicating tasks, can be modelled in the program. Each grid object can be initialized with a part of a grid on which it performs the computation. During the computation, it also exchanges boundary values with other objects, say after every iteration. This is done by invoking the message sender operator on the metaobject.

The point to be noted in this context is that in the proposed model, CORBA is transparent to the application developer. From the viewpoint of the application developer, the meta-object and sub-contracting are the key points. The system programmer is the only one who is aware of CORBA. It is the responsibility of the system programmer to ensure that the code written by the application developer is translated into calls on CORBA.4 This is a fundamental difference between P-CORBA and the other two models, PARDIS and Cobra. In both these models, the application developer is aware of CORBA and the programs on top of an extended CORBA model.

SYSTEM SUPPORT

This section presents the system programmer's view of the model. The key components of the system include the translator and the runtime system or what the authors refer to as the 'kernel'. The translator converts the application developer's program into calls on the kernel and calls on CORBA's ORB. The kernel is responsible for executing the sub-contract, load balancing and fault tolerance

```
class grid {
public:
void Initialize( .. ); //Initializes the
grid

void Computation( .. );//The actual
computation to be performed

void PassBoundaryValue( .. );//To pass
boundary value to another object

void ReceiveBoundaryValue( .. ) ;//Receives
values from another object
} ;
main( )
grid gl, g2, g3;
Parclass ParGrid holds grid;
ParGrid Pg;// grid meta object

gl. Initialize( .. );//similarly for the other
two grids
```

```
Pg « gl; / / similarly for the other two
grids
Pg .. Computation(U);
Pg 0 -> 0 gl to g2;

//message sender operator on the meta object,
message sent from gl to g2
Pg 0 -> 0 g2 to g3;
Pg .. Computation( .. );
//next iteration
}
```

Sample Code to Explain Inter-object Communication

Translator

The translator parses the program written by the application developer. It converts the meta-object into a CORBA object that aggregates the object references of the objects in its collection. Each object in the collection is converted into a server object in CORBA while the meta-object becomes the client object in CORBA. If the application developer makes the objects in the collection communicate with each other, then these calls are converted into CO RBA method invocations by the translator. Hence, the translator is also responsible for ensuring inter-object communication.

This concept illustrates another difference between P-CORBA and the two other models, viz. PARDIS and Cobra. The communication between the objects in the collection is through a mechanism outside CORBA in both the models. But it is not explained clearly how a process-based communication mechanism such as MPI integrates into the object-based communication paradigm of CORBA in these models, whereas in P-CORBA this communication is also through the ORB. However, this may result in a higher overhead for the inter-object communication in the proposed model. But this simplifies the handling of heterogeneity whereas in the other models, the heterogeneity handled by the system is restricted by the mechanism used for inter-object communication

The translator converts the code of Fig. 6.1 into the form shown in Fig. 6.4. This figure shows the IDL file Matrix.idl. It shows some additional methods such as Update, SaveState, etc. These methods are required for the filtering mechanism used during object migration, as will be detailed in Section 6.4.3. The Fig. 6.4 also shows the implementation file Matrix_impl.cc. The Matrix_server.cc file is the code for actually deploying the server objects. The client code is the meta-object, which makes method invocations on all server objects concurrently. The code shown is specific to mico, the CORBA ORB used in the implementation of P-CORBA

The Kernel

The kernel is designed as a distributed kernel that is resident on all the nodes of the system. Each of the kernel entities monitors

the load conditions on the respective nodes of the system. When the application developer starts the program on a particular node of the system, the kernel entity on that node is called. This entity interacts with the other kernel entities and gets to know the least loaded machines that are available. It migrates the objects in the program to these nodes and the sub-contract is then executed.

The kernel also takes care of the sub-contract directives and the locking specifications. The kernel uses the concurrency service specification [29] of CORBA for handling locking problems. The concurrency specification provides a mechanism for ensuring the consistency of the state of an object that is accessed by concurrently executing computations. It provides an interface called LockSet interface that has methods for acquiring and releasing locks. In the context of the model, the base class is made to inherit from the LockSet interface. The kernel acquires the necessary locks before making the method invocation. The concurrency specification provides many locking modes that can be used by the kernel to ensure consistency in the state of the object. But in reality, the authors had to implement the required parts of the concurrency service as most CORBA vendors do not provide an implementation of this service.

Load Balancing Strategy

The most important issue in any load balancing strategy is the load index. Several load indices for measuring the load have been proposed and used. The list includes CPU queue length, time-averaged CPU queue length, available memory and the CPU utilization among others. In the proposed model, the CPU queue length is used as the load index as it has

been found to be simple and effective [23]. A threshold policy is used to classify the nodes of the system into three categories. If the load on a node is greater

than a threshold, T 1, then the node is called a sender. A node is termed as a receiver if its load is less than a threshold, T2, with T2 <Tl. Other nodes are grouped into the third category. Nodes in this category do not take part in task transfer. If a user program is initiated at a particular node, then that node automatically becomes a sender.

In sender initiated load balancing algorithms, the task migration is initiated by the heavily loaded node. These algorithms perform well when the load on the system is low, i.e. when it is easier to find a lightly loaded node. When we say that the load on the system is low, it means that the load on a majority of the nodes is low. In contrast, in receiver initiated load balancing algorithms, the task migration is initiated by the lightly loaded node. These algorithms perform well when the load on the system is high, in which case it is easier to find a heavily loaded node. The adaptive algorithm that is used in the proposed model combines the advantages of both the sender-initiated and the receiver-initiated algorithms [33].

There are two major components in the algorithm. One is a sender-initiated component which is triggered on a node when it becomes a sender. The other is a receiver-initiated component which is triggered on a node when it becomes a receiver. It can be observed that a node can become a sender or a receiver at different points of time depending on how the load changes in that node. Each node maintains its view of the load on the system in the form of three lists containing the nodes which fall into each of the categories mentioned above. The

information about other nodes is collected when nodes poll each other to initiate the transfer of tasks. The task transfer is equivalent to object migration in the proposed model.

If a node is a sender at a particular time, then it tries to migrate the objects residing in that node to the nodes in its receiver list. If a node is a receiver, then it tries to migrate objects to itself from the nodes in its senders list. Hence, this algorithm performs well when the load on the system is low (or the sender-initiated part dominates) and even when the load on the system is high (or the receiver-initiated component dominates).

Load Balancing Service

The interface of the kernel object is shown in Fig. 6.5. The load balancing service consists of the collection of kernel objects. The important guidelines that were observed when building the load balancing service are

- Built on CORBA concepts: The object model of CORBA is strictly adhered to. Thus, concepts like the separation of interface and implementation, and clients depending only on interfaces and not on the implementation are used.

- Allows for local and remote implementations: This could be important, if for instance, the performance requirement of an application is such that the service must be executed in the same process as the client.

- Flexible: Since services are designed as objects, they could be combined in interesting ways. For instance, as shown in this chapter, the load balancing service and object migration service are combined to balance the load on a NOW.

- Finding a service is orthogonal to using it: This means that since services are designed as a collection of CORBA objects, there need not be any special ways of finding them. It is left to the ORB vendor to make the service available to clients. But, this chapter goes a step further and gives general guidelines to the ORE vendor for deployment of the services (refer to Section 6.5.1).

The load balancing service depends on the object migration service for balancing the load on a NOW. This is similar to the dependencies between services as given in the CORBA services specification. For instance, the lifecycle service depends on the naming service and the transaction service depends on the concurrency service.

IMPLEMENTATION

The mechanism for locating a migrated object by using message filters has been implemented over a 10 Base T ethernet network, with all the nodes running the LINUX operating system. The CORBA implementation that was used is a public domain ORB called mico. It is a fully CORBA-compliant ORB with a C++ language mapping. This section describes the implementation of the location mechanism using message filters. Another approach

for locating migrated objects by using a servant manager is also explained.

The method call made by the client is intercepted by the server side filter. This filter makes a bind call to the current location of the object. This returns the new object reference to the filter. The filter subsequently makes the method invocation on this reference. As per the semantics of message filters, the filter bounces the request with the return values of this method invocation. The code for the filter implementation is illustrated in Fig. 6.10.

This exception is caught by the

```
class FilterAsClient :Public..
private:
….
public:
ReturnType ml ( .. ){

//  filtering method ml () in the
object.
Address = LookUp();

//lookup for current object address,
may be from a persistent store (this
may be required if object is
persistent) .
```

```
ref = orb→bind(Address,ObjectName);
// bind to new location and get
reference
        Objref =InterfaceName ::
_narrow (ref) ;

//  narrow to appropriate type
ReturnValue = Ob jRef→ml ( .. );
//  actual method call.
return ReturnValue;
//  return to client.
}
} ;
```

client side filter. The client side filter makes the bind call to the new location. It gets the new object reference and makes the method invocation. Finally, it returns the results to the original

Performance Studies

Two experiments have been conducted on the basis of the three implementations described above. The goal of the first experiment is to show that the home-based model (the filter approach) performs better than the servant manager-based chain model. If an object moves from its home location to another machine, it is said to have migrated by one hop. The experiment was conducted with the object moving an increasing number of hops.

The performance of the servant manager approach degrades as the number of hops increases. The filter approach (for this experiment, the model used was the filter acting as client), however, performs well irrespective of the number of times the object moved. This is because the filter always acts as a home agent and connects directly to the new location of the object. The result of this experiment is shown in Table 6.2.

Table 6.2 Comparison of Filter and Servant Manager Approach

Number of hops	Filter as client	Servant manager
Time in milliseconds		
1	18.45	13.78
2	21.75	21.70
3	21.75	32.30
4	21.75	36.89
5	21.75	44.25
6	21.75	51.05

The other experiment that was conducted in entailed comparisons of the two variants of the filter-based approach. The performance of the two approaches was measured with increasing message sizes. As the message size increased, the filter as client approach started performing poorly, as compared to the filter as forwarder approach. The reason for this is that in the filter as client approach, the filter has to marshal the parameters again. If the message size is large, the time for marshalling the parameters increases.

POSSIBLE QUESTIONS(2 Marks)

1. What is DP Model?
2. Define IGM.
3. What is the use of P-CORBA?
4. What are the basic DP services?
5. List the Extended Services for IGC?
6. Define User interface in PCORBA.
7. What is translator?
8. Write a note on Load Balancing Services.
9. What is Message Filter?
10. What is the need of Kernel?

POSSIBLE QUESTIONS(6 Marks)

1. Describe the DP model and Initialization in Iterative Grid Computation.
2. Write short notes on Existing works of CORBA.
3. Brief about Load Balancing and Domain Decomposition in DP model of Iterative Grid Computation.
4. Explain the notion of Concurrency in CORBA.
5. Differentiate between Local Coordinator and System Coordinator in Distributed Pipes.
6. Describe the implementation of CORBA with an example.
7. Discuss the Basic Distributed Pipe Services and Extended Services for IGC.
8. Give a detail description about the performance studies of CORBA with MPI.
9. Describe the design and implementation of Distributed Pipes.
10. Brief about the kernel of CORBA.

Karpagam Academy of Higher Education

Department of CS, CA & IT

Subject: Grid Computing (17CSP205B)

Batch : 2017-2019 **Class: I M.Sc CS**

Objective Type Questions

UNIT III

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	KEY
1	The library is built over_____domain stream protocol	TCP and unix	TCP and lunix	unix and lunix	TCP and HTTP	TCP and unix
2	DP stands for_____	distance pipe	distributed pipe	distributed program	distance program	distributed pipe
3	_____initials a message to the sc through the lc running on its machine	open dist pipe()	a read dist pipe()	create dist pipe()	delete dist pipe()	create dist pipe()
4	_____initials a message to the sc through lc with the name of the dp as an argument	delete dist pipe()	close dist pipe()	write dist pipe()	read dist pipe()	delete dist pipe()
5	The mainentities of the implementation are the	kernal and translator	shell and translator	unix and lunix	kernal and nuix	kernal and translator
6	ORB stands for_____	object request broker	object required broker	object request based	obtain task-Grid info()	object request broker
7	The translator finally adds a method called_____to the client object	resolve_initial references	sub contract	TCP	shell and translator	sub contract
8	The _____method also needs to be modified if it is implemented by the	list_initial_services	resolve_initial_reference()	both (a) and (b)	cpu hog	list_initial_services
9	There are _____ways of deploying the service	two	three	four	five	two
10	GCA stands for	genetic computer algorithm	genetic clustering algorithm	genetic computing algorithm	genetic annealing	genetic clustering algorithm

11	TSP stands for_____	travelling salesman	travelling salesman	travelling solve program	none of these	travelling salesman
12	SA stands for_____	similar algorithm	simulated annealing	standard algorithm	general algorithm	simulated annealing
13	The load was measured by using a linux specific system call_____	sysinfo	cpu hog	load	process id	sysinfo
14	GA stands for_____	genetic algorithm	general algorithm	genetic annealing	genetic computing algorithm	genetic algorithm
15	IGC stands for_____	international grid computing	interval grid computing	iterative grid computing	none of these	iterative grid computing
16	MPI stands for_____	message passing interface	message passing interval	monitor passing interface	local coordination	message passing interface
17	_____socket is created and bound to a local port	TCP	HTTP	both (a) and (b)	TCP socket dexriptor	TCP
18	The message contains_____	TCP socket dexriptor	access node	process id	monitor passing interface	monitor passing interface
19	GCTST stands for_____	grid computation task	grid computer task submitted	general computer task submitted task	none of these	grid computation task
20	The calls_____are directly translated to the underlying systen call	read dist pipe	write dist pipe	both (a) and (b)	distance pipe	both (a) and (b)
21	_____is a model for transparent programming of communicating parallel tasks	ARC	TCP	Distributed pipes(DP)Grid computing	COP	Distributed pipes(DP)Grid computing
22	The DP provides a ---------- of interfaces for	Random set	Uniform set	Same set	Different set	Uniform set
23	Worker processes are called as----------	Iterative grid modules	Interactive grid modules	Iterative grid models	Interactive grid models	Iterative grid modules
24	IGMs return the results of their computation to the------	Parallel process	Multiple process	Worker process	Master process	Master process
25	GCT is reffered as----------	Grid computation tasks	Grid computational tasks	Grid communication tasks	Grid component tasks	Grid computation tasks

26	The master process provides the grid information to the systemby using the call	Initialize work()	Close work ()	Close GM()	Send grid info()	Send grid info()
27	The anonymous migration of an IGM is initiated when the master process invokes the---------- call	Migrate()	migration	mig()	migrated()	Migrate()
28	After initializing the data structure,the IGM collects	Migrate()	obtain task-Grid info()	obtain task-Data()	obtain task machine Info()	obtain task-Data()
29	The lc is reffered as----------	Local coordinator	local communication	loop coordinator	local coordination	Local coordinator
30	The lc maintains---------- tables to support bare DP	three	one	two	four	two
31	UPT is reffered as	user program information table	user process information table	user program interface table	user process interface table	user process information
32	GCTST is reffered as	Grid computation task submitt table	grid computational task supported table	grid computation task submitted table	none of these	grid computation task submitted table
33	The initial communication between a process and the lc	single	multiple	common	none	common
34	What is mean by SC--------	system controller	system coordinator	servive coordinator	system communication	system coordinator
35	The SC is connected to LCs through the----------sockets	IP	SMTP	PPP	TCP	TCP
36	What is mean by DPT----------	distributed pipe task	distributed pipes table	distributed pipeline task	distributed pipeline table	distributed pipes table
37	LCT is reffered as----------	Local coordinators table	local coordinator tasks	loop coordinator table	loop coordinator tasks	Local coordinators table
38	GCWT means---------	grid computing work table	grid computationwor	grid computing work task	grid computation work task	grid computation
39	GCTT means---------	Grid computational task table	grid computing task table	grid computation task table	none of these	grid computation task table
40	Thus the process can be intimated when some other	write	read	delete	update	read
41	The notion of concurrency that is used in	P-CORBA	D-CORBA	A-CORBA	M-CORBA	P-CORBA

42	The meta-object is an entity which can aggregate objects of the_____class	same	different	two	one	same
43	Meta -object can be created by instantiating a meta-class called	sur class	par class	var class	con class	par class
44	The application developer can _____objects into the meta-object	delete	update	insert	exit	insert
45	The sub-contract call on the meta-object is made through	parallel	addition	greater than	subcontract	subcontract
46	OBS stands for	object-based sub-contracting	object-based sub-system	object-based sub-coordinate	object-based sub-cordation	object-based sub-contracting
47	A message can be sent from one object to another object using	message receiver	message sender	e-mail	mobile	message sender
48	CORBA stands for	common object request broker architecture	computer oriented request broker	common object receiver broker architecture	computer object receiver broker architecture	common object request
49	CORBA is_____for the application developer	transparent	sequence	parallel	metadata	parallel
50	Each object in the collection is converted into a_____object	client	services	customer	server	server
51	The matrix-server.cc file is the code for actually	deploying	destroying	developing	deleting	deploying
52	The kernel is designed as a_____kernel	distributed	joined	linked	unlinked	joined

UNIT-IV

SYLLABUS

Sneha-Samuham Grid Computing Model: A Parallel Computing Model over Grids – Design and Implementation – Performance studies. Introducing Mobility into Anonymous Remote Computing and Communication Model – Issues in Mobile clusters and Parallel Computing on Mobile Clusters – Moset Overview – Computation Model – Implementation and Performance

Introduction

The Sneha-Samuham model provides adaptive parallel execution of tasks over a computational grid. Its strength lies in transparent splitting of a parallel task into sub-tasks of appropriate granularity, depending on the computation capability of participating nodes. Moreover, its user interface for sharing the computing resources across the Internet, makes the model user-friendly. Users can harness the computing power of 'friendly' computers over the Internet for parallel processing.

A good parallel computing mechanism can boost the performance of a computational grid to execute applications. None of the existing grid computing models supports automatic task splitting, even though it is possible for many classes of applications. These systems expect the user to submit a batch of jobs to the system

and the scheduler schedules these jobs on the participating nodes. In the proposed model, automatic task splitting support has been provided for certain classes of applications. This relieves the user from the burden of task splitting related issues. Also, the user does not need to know the computation capabilities of individual nodes participating in the grid. Currently, Sneha-Samuham supports only purely data parallel scientific applications that are coarse-grained.

SNEHA SAMUHAM:A PARALLEL COMPUTING MODEL OVER GRIDS

In the Sneha-Samuham model, a computer, over the Internet, can donate its computing power to other computers as well as use the computing power of other computers. The computer which donates its computing power is called a 'donor' and the computer which makes use of another computer's computing power is called an 'acceptor'. A node2 can act either as a 'donor' or as an 'acceptor' or both as an 'acceptor' and a 'donor'. Ideally, the number of donors should be much higher as compared to the number of acceptors. An acceptor can use the computing power of its donors for executing parallel applications. The topology, architecture, various components and the computation model of Sneha-Samuham are explained in the following sub-sections.

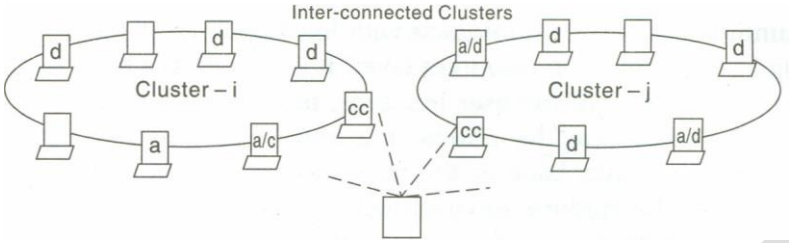
System Topology

The collection of nodes. within an individual LAN is referred to as a 'cluster'. A designated node in

each cluster acts as a mediator between the nodes inside its cluster and the outside world.

The mediator is called a Cluster Coordinator (CC). If a cluster contains a single node, then that node itself acts as the CC for that cluster. Any node inside a cluster (including the CC) can act either as a 'donor' or as an 'acceptor' or both as an 'acceptor' and as a 'donor' or may not participate in the grid computing at all.

There are several advantages with this kind of topology. Since there is no global coordinator, the model is scalable and there won't be any single point failures. The next advantage is that every node of a cluster participating in the grid computation need not be accessible from outside, over the Internet. Since every cluster that participates in the computation, contains a CC, it is sufficient if that CC is accessible over the Internet. The machines inside a remote cluster can be accessed through the CC of that cluster. Private addressing and source Network Address Translation (NAT) of a LAN need not be disturbed while the machines of the LAN are used for grid computing. Its other advantage is that providing authentication, security and fault tolerance for grid computing becomes easy. For example, there is no need to secure a machine from other machines inside its cluster. It is enough to have a good security system in the CC to secure the machines of a cluster from outside machines .



a: Acceptor *aid:* Acceptor as well as donor

d: Donor cc: Cluster Coordinator

A PARALLEL PROGRAMMING MODEL OVER GRIDS

Layered Architecture of Sneha-Samuham

The Sneha-Samuham model is viewed as a five-tiered architecture, with the middle three layers constituting the core. The functionality of each layer is explained in this section.

• **Computing Resources**

This layer contains various computing resources, with varying processor speeds, memory, architecture and operating system running on them. In this model, a computing resource implies, the combination of both the hardware resource and the local management software (operating system) running on it.

• **Runtime Environment**

Runtime environment interacts with local operating systems existing in the computing resources layer. It is responsible for collecting the sub-tasks from the user interface, migration and execution of these sub-tasks on the remote machines over the Internet, and

getting the results back to the node, where the computation was initiated. The runtime environment of the system contains three components that run as daemons on the respective nodes: acceptor, donor and ccdaemon. A node which acts as both a donor as well as an acceptor runs both the daemons. Every CC runs a daemon called ccdaemon to coordinate between the acceptors and the donors. The node to which a parallel task is submitted should have the acceptor daemon running. Also, the nodes where the donor daemon is running can participate in the parallel computation by sharing work from acceptors. The interactions among these three daemons is governed by a well-defmed protocol. The core grid computing services provided by the runtime environment can be accessed by using the APIs provided in the layer above it.

Whenever a cluster wants to participate in the parallel compu-tation, it initially starts the ccdaemon on its Cc. After a donor or an acceptor starts at a node, it registers with its local CC through the ccdaemon running on that CC.

• **Application Programming Interfaces (APIs)**

This layer contains various APIs to access the services provided by the runtime environment for grid computing. This layer is introduced to make the runtime environment application-independent. These APIs provide a standard way for the above layer (user interface), to request the runtime environment for various kinds of grid computing services, such as collecting friend

CCs information, migration of a sub-task to a remote node, getting back the results of a sub-task, etc. This layer, including the runtime environment layer, provides core grid computing services and the interface to access them. Any new service provided in the runtime environment can be accessed by including the corresponding APIs in this layer. This makes the user interface transparent from the updates made at runtime environment.

• User Interface

This layer provides user level tools for collecting the computing resources and submitting tasks to the grid. It contains two sub-components. One is the Friend Machines Interface (FMI) to collect the resources and the other is a set of user level commands to submit applications to the grid. The FMI is an instant messenger kind of Graphical User Interface (GUI) tool, which interacts with the local cluster coordinator using the appropriate APIs in the lower layer. It provides various services to the user to aggregate the resources over the Internet for grid computing as explained below:

- The owner of a cluster can request owners of other clusters over the Internet to become a friend cluster to his/her cluster. If he/she accepts, then both the clusters become friends to each other. Each CC includes the name of the other CC in its Friend Clusters Table (FCT).

- One can accept or reject the request made by others over the Internet to make his cluster a friend to them. Protocol

similar to that of 'yahoo messenger' for discovering chat friends is used.

- It displays all friend CCs of a CC and highlights the live friend CCs. Along with the friend CCs, other information such as number of donors available in each friend cluster, their GCCFs and communication latencies can also be displayed on the basis of the user's preference.

- The user or the application can select any number of donors from friend clusters for parallel computing.

If two clusters are friends to each other, any node can use the computing power of any other node in the two clusters. A cluster can have any number of friend clusters. A CC maintains a list of all its friend CCs. Whenever a parallel application is started on a node, it can make use of the machines (which are running donor daemon) that belong to all of its friend clusters for parallel processing.

In Sneha-Samuham, it is assumed that each cluster is owned by a user or a group of users. It is a realistic assumption as each LAN is owned by a research group or an organization. They decide whether their cluster will participate in the grid computing or not. They can designate any node as a CC, an acceptor or a donor by running the corresponding daemon on it. The owner(s) can make his cluster a friend to any other cluster by using the FMI described above. A cluster including all of its friend clusters

forms a 'computational grid'. A number of such grids can exist over the Internet.

Apart from FMI, the user interface also contains a set of application level commands to submit parallel applications to the grid. The set contains one command for each class of parallel applications. The functionality of these commands includes the splitting logic for that particular class of applications and makes use of the lower level APIs for accessing generic grid computing services. As the splitting logic varies for different classes of applications, each class should contain one command in this layer. A new class of applications can be supported by adding the corresponding command to the existing set of commands. The user can see the supported commands and, their description and can use the appropriate command to submit his application to the grid. However, implementing these kinds of commands is not possible for all applications as some class of applications are inherently not parallelizable.

• **Grid Computing Applications**

This layer contains the applications, which can benefit from grid computing. A grid computing application is CPU-intensive and there should be algorithms to partition the application into independently running parts. The parts must be executable remotely without much overhead, and the remote machine must meet any special hardware, software and/or other resource requirements imposed by the application. The application is more

scalable, if these sub-tasks (or jobs) do not need to communicate with each other.

Computation Model

The computation model follows the Master-Worker [13] process model. Upon receiving the task, the user level command executes as a master process and contacts the acceptor running on. that node. Then the acceptor contacts its local ccdaemon and gets the addresses and GCCFs of requested number (or available number, if requested number of machines are not available) of friend machines. The master process splits the task into sub-tasks of appropriate granularity depending upon the number of available friend machines and their current computing capabilities. Once the splitting gets over, it migrates each sub-task to the corresponding node. The donor daemon that receives the sub-task to be computed, spawns a worker process for that sub-task and returns the results after the execution is completed. The division of tasks among the available nodes is done as described in the following sub-section.

Task Splitting

The computational capability of a node over the grid can be measured from the factor called GCCF,

$$F = sm / lt$$

where s, m and I are processor speed, memory and average load on a node, respectively and t is the communication latency from

a node, from where it received the job to be executed. The division of task among the available lightly loaded nodes is done as follows:

If there are n nodes with GCCFs, $f_1, f_2, f_3 \ldots fn$ ...

Let, $F = f_1 + f_2 + f_3 + \ldots fn$

And if the total task size is G, then

Grain size assigned to the first node, $g_1 = (f_1 / F)G$

Grain size assigned to the second node, $g_2 = (f_2 / F)G$

Grain size assigned to the nth node, $g_n = (f_n / F)G$

The above task distribution is applicable only to data parallel applications. Normally in data parallel applications, a functionality will be executed on a large chunk of data. This data can be divided into several parts and can execute the same functionality on individual chunks. Each chunk could be of different size. This technique may have to be extended to apply for task parallel applications

DESIGN AND IMPLEMENTATION OF THE MODEL

The middle three layers, viz. the runtime environment, APIs and user interface, form the core of the Sneha-Samuham model. The following sub-sections explain the designs of these three layers and implementation details of the model. In the following discussion, local machine with respect to some machine, means

the one which resides in the same cluster. For example local CC means the CC of that cluster.

Runtime Environment

The collection of acceptors, ccdaemons, donors and their interactions form the runtime environment of the system. The interactions among these three daemons as well as their communication with the upper layer are governed by a well-defined protocol. The following sub-sections dtscribe the design of each of these daemons and their interactions with other components of the system.

Cluster Coordinator (CC)

The CC in the Sneha-Samuham model acts as a resource collector. A cluster can be a friend to other clusters over the Internet. The CC of each cluster maintains a table called FCT. Each entry in the FCT contains the address of a friend CC. The FCT of every cluster is updated whenever a new CC is added to the list or an existing CC relinquishes the friendship with that cluster! The interaction between a ccdaemon and FMI will do this.' It also maintains a pending requests list, whose entries are the requests that came from other ees for friendship with this cluster and which are not responded by the user (or owner) of this cluster.

Apart from the friend clusters' information, a CC also maintains lists of its local donors and of its local acceptors. Whenever an acceptor from a friend cluster asks for donors, it returns the

addresses of the nodes stored in the donors' list along with their current computational capabilities. The GCCF defined in the previous section represents a node's current computational capability. '

A CC interacts with its FMI, local acceptors, local donors, and with other CCs., In the INIT' state, ccdaemon initializes its data structures. In the LISTEN state, it waits for messages from the other daemons with which it is allowed to interact. If it receives any message from the FMI, friend CC, local acceptor or local donor, then it changes its state to FMI Msg RECVD, Other CC Msg RECVD, Acceptor Msg RECVD or Donor Msg RECVD, respectively and services the message. Any error message in these states causes the state to be changed to ERROR and the appropriate action will be taken/ Various kinds of messages could come from each of the above-mentioned daemons. For example, from FMI, the message could be to request a specific CC to become a friend to it or a response to the request from another CC, forwarded by it. For each message, the ccdaemon executes a different service routine to service the request.

Acceptor

The acceptor receives tasks submitted by the user interface and executes them on its friend machines. It gets the global computing resources from its local ccdaemon. Since users can submit more than one parallel task to an acceptor, it maintains a table of all submitted tasks. This table is called a Task Table (TT). Each entry

in the TT contains a task-ID and the corresponding sub-task IDs after the task is split into sub-tasks. Each entry is updated with the target CC and donor addresses after the sub-tasks are migrated

The acceptor interacts with its local ee, local donors (in case it submitted jobs to them) and with the user interface. In the INIT state, the acceptor daemon initializes its data structures and registers with its local ee. In the LISTEN state, it waits for messages from the other processes. If it receives a message from any of the user processes, its local ee, or its local donor, then it changes its state to UP Msg RECVD, CC Msg RECVD or Donor Msg RECVD, respectively and services the message. If any error occurs in this process, the state is changed to ERROR and the appropriate action is taken.

Donor

A donor daemon registers with its local ccdaemon and gives replies to the requests made by its ccdaemon such as current GCCF of the machine, etc. If the current GCCF is above some threshold, it receives sub-tasks from the acceptors of its friend machines, executes them locally, and sends results back to the acceptor if it is local or through its ee if it is from an outside cluster. In order to store information of all the sub-tasks that are running on this node, it maintains a table called Sub-Task Table (STT). Each entry in the STT contains a sub-taskID, taskID generated by its parent noder' source CC and acceptor addresses from where the sub-task has been migrated. This information is required to send

the results back after the execution of a sub-task is finished. If the sub-task is from a local acceptor, then its source CC address is zero and the result of the sub-task can be directly sent to the acceptor.

A donor daemon interacts with its local CC, local acceptors and the Worker Processes (WPs) spawned by it for executing sub-tasks. In the INIT state, it initializes its data structures and registers with its local ccdaemon.

In the LISTEN state, it waits for the messages from the other processes with which it could interact. If the donor daemon receives a message from its ccdaemon, local acceptor or from its worker process, then it changes its state to CC Msg RECVD, Acceptor Msg RECVD or WP Msg RECVD, respectively and services the message. If any error occurs in this process, the state is changed to ERROR and the appropriate action is taken

PERFORMANCE STUDIES

The neutron shielding simulation [14] application has been chosen for studying the performance of the Sneha-Samuham grid computing model. It is a nuclear physics application, in which a beam of neutrons is delivered in the experiment. When this neutron beam strikes a lead sheet of certain thickness perpendi-cularly, it is important to predict the number of neutrons that can penetrate from the other side of the lead sheet. This application predicts that number by using the Monte-Carlo simulation. More

details about this application can be found at [14]. For all simulation experiments, a lead sheet of 5 units thickness is assumed. The number of neutrons will be a power of 10 in each experiment. In practical situations, the number of neutrons depends upon the type of experiment and the- amount of time for which that experiment has to be conducted. The amount of computing power required is directly proportional to the number of neutrons. In the following discussion, homogeneous and heterogeneous machines (or clusters) are categorized with respect to processor speed only, that is, in a cluster of homogeneous machines, all the machines will have the same processor speed.

Power of Sneha-Samuham:Institute-wide Grid

The first column of the table shows the number of neutrons (problem size) for each experiment. The time taken for this application by a single machine and the time taken on the grid of three clusters using Sneha-Samuham has been measured. The values in the second column are the execution times for the corresponding problem size given in the first column, using a single machine of processor speed 367.5 MHz. The values in the third column are achieved by a grid of 14 machines, distributed across three clusters with varying processor speeds ranging from 267 MHz to 2400 MHz. The fourth column shows the speed-up achieved by the grid when compared to the single machine.

ISSUES IN MOBILE CLUSTERS AND PARALLEL COMPUTING ON MOBILE CLUSTERS

There has been an increasing interest in the use of clusters of workstations connected together by high speed networks for solving large computation-intensive problems. The trend is mainly driven by the cost-effectiveness of such systems as compared to large multiprocessor systems with tightly-coupled processors and memories. However, the recent proliferation of mobile devices and advancement in wireless connectivity has made parallel computing on mobile clusters a feasible proposal. Mobile devices can be part of the cluster by playing several unique roles. They can be used as a front-end to the cluster functionality, such as submitting a job, managing processes, or viewing statistics. In case of an MH which has very poor computing power, the device must be able to utilize the cluster seamlessly to access the computational power. However, the MH can also be a contributor of computing power to the cluster, in the case of devices such as laptops which have a substantial amount of computing power equal to their static counterparts. Distributing computing power in a cluster consisting of a network of heterogeneous computing devices represents a very complex task. However, it becomes complicated when mobile devices are also a part of it

There are several key issues that distinguish parallel computing on mobile clusters from that of the traditional workstation clusters, namely. These are:

- Asymmetry in connectivity,
- Mobility of nodes,
- Disconnectivity of mobile nodes,
- Timeliness,

- Changing loads on the participating nodes,
- Changing node availability on the network,
- Differences in computing capabilities and memory availability, and
- Heterogeneity in architecture and operating systems.

Asymmetry in Connectivity

The traditional cluster computing models do not face the problem of heterogeneity in the network connection as the entire set of workstations that are participating in the clusters are connected only by the wired network. Wireless networks deliver much lower bandwidth than wired networks and have higher error rates. Mobile devices are characterized by high variation in the network bandwidth, which can shift from one to four orders of magnitude, depending on whether it is a static host or a mobile host, and on the type of connection used at its current cell. Thus the program-ming model must be able to distinguish among the types of connectivity and provide flexibility for easy variation of the grain size of the task to account for the variations in bandwidth. However, these systems are suitable only for coarse grain level parallelism due to the- communication overhead.

PARALEL COMPUTING IN MOBILE CLUSTERS

Mobility of Nodes

Due to mobility of nodes, the notion of locality becomes important as users move from one cell to another. The locality becomes

important as the change in the mobile node's location means a change in the route to that node and in the consequent Communication overhead. The ability to change locations while being connected to the network increases the volatility of some of the information. Static data could become mobile in the context of mobile computing. As a node moves, nearby information servers get farther away and should be replaced by closer ones offering the same or more relevant contextual information. Traditional computers do not move, as a result of which information that is reliant on location can be configured statically, such as the local DNS (Domain Name Service) server or gateway, the available printers, and the time zone. A challenge for mobile computing is to define this information intelligently and to supply the means to locate configuration data appropriate to the present location. Mobile computing devices need to access more location-related information than stationary computers if they are to serve as ubiquitous guides to a user's environment. As the mobile device moves and as the speed of motion changes, the quality of the network link and of other available resources might change significantly. Thus, the system should be able to adjust according to the changing conditions. For example, when an MH which has taken the task moves from one cell to another, then the system still needs to track these MHs.

Disconnectivity of Mobile Nodes

The periods of disconnectivity of nodes in static networks are usually treated as faults. However, in the context of mobile nodes,

the disconnectivity may be due to roaming and hence enter an out-of-coverage area or voluntary disconnection (doze mode) to save battery power.

Timeliness Issue

Timeliness refers to the delay that is taken for the mobile device to regain its full state when it moves from one cell to the other or after reentering a coverage area after disconnection. Timeliness issue is an important issue especially in real-time systems. Whenever a mobile host moves from one cell to the other, it is associated with a hand-off, to ensure that data structures related to the mobile host are also moved to the new connecting point, the Mobile Support Station (MSS). This involves an exchange of several registration messages. This may cause some delay and it should be fast enough to avoid loss of message delivery. In addition to this, there is a possibility that the mobile host could move out of coverage after accepting the task for execution. These issues need to be addressed with respect to the mobile cluster model.

Changing Loads on the Participating Nodes

When using workstations for executing parallel applications, the concept of ownership is frequently present. Workstation owners do not want their machines to be overloaded by the execution of parallel applications, or they may want exclusive access to their machines when they are working. Reconfiguration mechanisms are thus required to balance the load among the nodes, and to

allow parallel computations to co-exist with other applications. In order to overcome these problems, some dynamic .load balancing mechanisms are needed. There are differences in loads among the nodes due to the multi-user environment, and when an application is run on a heterogeneous cluster. In these cases, it is important to balance loads among the nodes to achieve sufficient performance. As static load balancing techniques would be insufficient, dynamic load balancing techniques based on runtime load information would be essential. It would be difficult for a programmer to perform load balancing explicitly for each environ-ment application, and automatic adaptation by the underlying runtime is indispensable. This gets aggravated when mobile devices are part of the cluster.

Changing Node Availability on the Network

In traditional distributed systems, nodes keep leaving and joining the system dynamically. The joining and leaving of nodes may be due to either node failure or link failure. However, the system must be smart enough to continue with the computation. The availability of node becomes more fuzzy in a distributed mobile computing scenario as the availability is also affected by the movement of the nodes. It is possible that the node may enter an area which is not under the coverage area of any MSS. It is also possible that node availability is transient with respect to the execution of the program. While a mobile node is computing a sub-task, it can go out of coverage and enter back into the coverage area before the completion of the execution of the program.

Difference in Computing Capability and Memory Availability

As each host may have different capabilities (such as memory) and different processing powers, it is essential to allocate tasks to the nodes on the basis of their capabilities and processing power. MHs may especially have lower computing power and memory in contrast to their static counterparts.

Heterogeneity in Architecture and Operating Systems

Although it is reasonable to assume that a new and stand-alone cluster system may be configured with a set of homogeneous nodes, there is a strong likelihood of upgraded clusters or networked clusters having nodes with heterogeneous operating systems and architectures. However, it will be non-trivial to handle architectural heterogeneity, since the executable files are not compatible among architectures.

The issues discussed in this section make parallel programming on mobile clusters difficult. With the issue of mobility and other constraints associated with mobile devices, the management of distribution at the programming level further hardens the task. The existing cluster computing models solve only a subset of these issues. None of the earlier work in mobile clusters discusses these except for the timeliness issue which was discussed in Reference [7] .

MOSET OVERVIEW

The basic principle with which the Moset model was designed was to abstract out the heterogeneity of the constituting devices from the user. The user is made transparent to the hardware, bandwidth, operating system and other heterogeneity existing below the kernel. The user is given freedom to avail of any computing power required for his application, without being concerned about whether he is working with a constrained device or not. Moset is designed with clear separation between the administration functionality and the user functionality. It is the function of the administration to install and maintain the system. Once the system is deployed, the user needs only to use the APIs to interact with the system for performing parallel computing.

The MSS which is a static node covers a geographical region, namely the cell. Mobile nodes which are within that cell will be under the control of that MSS, and all communications from or to the mobile nodes in the cell can be made only through the associated MSS. In our model, the MSS aggregates the computing resources which are within its cell and presents them to the distributed system as a set of its own resources. The nodes which are participating in the cluster computing are grouped on the basis of the memory capability of the nodes. The nodes which are participating in the Moset kernel spawn their computing entity to the coordinator of the system, on the basis of their capability. The entire data which needs to be processed is multi-cast to all the participating nodes in the particular group o~ the basis of the size of the data. The runtime system of the kkrnel decides on the

anonymous node on which the task is to be executed on the basis of its capability. The details of the architecture are described in Section 8.5.

Unlike the nodes in a traditional distributed system, the mobile nodes cannot maintain high levels of availability or reliability due to wireless connectivity. Hence, in order to achieve reliable delivery of the data, considering the constraints of the mobile devices, Moset is built over an exactly-once reliable multi-cast protocol.

MOSET COMPUTATION MODEL

The Moset computational model is designed in such a way that it can handle the heterogeneity, fault tolerance, dynamic load balancing and computing power availability ..» The dynamic load on the participating systems and the nodes and link failures make the traditional cluster computing model unsuitable for parallel programming on MCC. These issues were effectively handled in [4]. However, the model does not address the mobile device participation in computation and the issues related to it. The Moset model is aimed at integrating the mobile devices with the static nodes to form a mobile cluster, and at harnessing the idle computing power of static and mobile nodes to utilize them for parallel computing.

Cluster Sub-groups

In our model, we use a notion of cluster sub-groups (or sub-groups), based on the memory capability of the nodes. A cluster

sub-group refers to the characteristics of the task submitted to that sub-group. Each host (static and mobile) based on its capabilities, joins the respective sub-groups. For example, sub-group LOW may refer to those tasks having memory requirements <10 MB. A MODERATE sub-group may have tasks having memory requirements <50 MB. A HIGH cluster sub-group may have tasks having memory requirements <100 ME. Tasks with memory requirements >100 MB may be in sub-group VERY HIGH.

Horse Power Factor and Dynamic Load Balancing

As each host may have different capabilities (such as memory) and different processing power, it is essential to allocate tasks to the static hosts and MHs on the basis of their capabilities and processing power. In order to incorporate this, each host is allocated an integer called HPF [4:], which is a measure of the computing power of a machine, the load on the machine and the network bandwidth of the communication channel. Machines in the network are normalized by a benchmark program to obtain a relative index of the machine, which is a static factor. The dynamic HPF of a machine is obtained by using this static relative index, the load on the machine and the communication bandwidth with which the machine _ is. connected to the network. This dynamic factor is normalized as a factor that represents the. number 'of entities that it could compute. When a host has HPF h, then h computing entities are allocated to the host. For example, if hosts A and B have hI and 10, as their respective HPFs, then the time taken by A to compute hI amount of a task is approximately equal

to the time taken by B to compute 10, amount of the same task. Abstracting the heterogeneity in this way makes parallel processing viable on unevenly loaded heterogeneous machines.

The dynamic communication bandwidth is not taken into consideration in HPF, as measuring dynamic bandwidth may lead to overhead. Also, to the best of our knowledge no technique has come out with a solution which could measure the dynamic bandwidth exactly without introducing significant overheads. This is clear from the fact that schemes like PathMon (which is a relatively better scheme as compared to the other techniques like pathchirp, pathload, etc.) requires about 0.25 seconds to report the available bandwidth in a wired network, and it could be even longer in a wireless channel. This also does not guarantee the exact measurement and is likely to have a relative error of 12 per cent. During hand-off, when the MH is in the process of receiving data, the HPF variations matter. But as with current technology relating to channel allocation such as the dynamic channel-allocation techniques this has become a matter of negligence.

Dynamic load balancing is done by maintaining two thresholds, viz. Upper Threshold (UT) and Lower Threshold (LT) at the MH. These thresholds are shared by all the computing entities within an MH. This can be achieved by creating the computing entities as threads of the MH. When the load on the MH is greater than UT, a computing entity on that MH leaves the group and increments UT and LT on that MH. Both UT and

LT are incremented so that all entities do not leave the groups at the same time. Similarly, when the load on the MH becomes lesser than LT, a computing entity on that MH joins the group and decrements both UT and LT. Two thresholds, UT and LT, are used to avoid oscillations of frequent join and leave.

Further, the load balancing mechanism used is non-pre-emptive and hence migration of an already executing task is not done. This is due to the additional communication overhead involved in process migration.

Parallelism in the Model

The data set which is very large, is multi-cast to the sub-group, on the basis of the size of the file. Multi-cast provides an' efficient mechanism for transferring the data to the computing entities for processing. Each computing entity independently splits the task on the basis of its ID and N For example, in the case of distributed image rendering application, frames having frame number 'f' such that mod (f, N) = ID are rendered by the computing entity with identifier ID. When an entity completes its share, it sends the result back to the destination host.

A host is assigned as a coordinator to the cluster and it keeps track of the total number of computing entities (called N) under each cluster sub-group. Further, each computing entity has a unique membership identifier (called!D, ranging from 0 to N-l) associated with each group subscribed by it.

Whenever an MH wants to participate in distributed processing, the daemon at the MH spawns a set of computing entities, on the basis of its capabilities.

IMPLEMENTATION

The system structure of Moset is shown in Fig. 8.2. A distributed Moset kernel is spread over the nodes that participate in Moset. A Moset kernel consists of multiple local coordinators (lcs) to coordinate local activities, multiple co-coordinators (ccs) to coordinate the global activities within their cell, and one system coordinator (Sc) to coordinate the overall global activity of the entire cluster. Each node, either static or mobile, that intends to participate in the Moset kernel runs a local coordinator. MH has a client process and a daemon. The client process and daemon run over a reliable multi-cast protocol. The client processes are used to submit tasks to the MHs (via multi-cast protocol) for distributed processing. The daemons are the computing entities at the MHs that execute a part of the submitted task concurrently with other daemons.

Local Coordinator

The lc runs on mobile and static hosts that participate in the Moset system, either to improve utilization or to share its work with an anonymous remote node. Any Moset communication to or from the local processes is achieved through the lc. In case of image rendering application, the huge data set which is to be

rendered, is multi-cast by the sc to the sub-group lcs, on the basis of the size of the file. Each lc independently splits the task on the basis of its ID and N. For example, frames having frame number '1'such that mod if, N) = ID are rendered by the computing entity with identifier !D. When an entity completes its share, the lc sends the result back directly to the sc if it is executed on a static host or through the cc if it is executed on a mobile host.

System Coordinator (sc)

The Moset kernel has one se in the logically grouped mobile cluster. The functions of the se are to manage all the nodes that take part in the cluster process and the spawned computing entities, to coordinate all the functions related to task distribution and execution, and to maintain the migration history of the tasks.

Whenever a static machine wants to participate in distributed processing, thus enhancing the machine utilization, it spawns computing entities based on its capabilities through its lc: The static machine also includes the MSS which spawns a set of computing entities representing the MHs which are within its cell and which are interested in participating in the computation. As discussed in the previous section, the se groups these computing entities into cluster sub-groups on the basis of the memory capacity of the machine which has spawned the entities. The se keeps track of the total number of computing entities (called N) under each cluster sub-group. Further, the se assigns each computing entity a unique membership identifier (called ID, ranging from 0 to N-l)

associated with each group subscribed by it. The exact number of computing entities spawned by each daemon will depend upon the HPF of the node.

The data set which is very large is multi-cast to the sub-groups' les by the se directly to the static host's lc; and through the ees to the mobile host's lc, based on the size of the file. Multi-cast provides an efficient mechanism for transferring the data to the computing entities for processing. A history of recent migrations is maintained.

The single point failure of the se is handled by replicating the state of the se in another nearby static node so that in case of failure, the states are not lost and the system can still survive. When an lc learns that the se has failed, it initiates the process of identifying the next se by using an election algorithm [14].

Co-coordinator (cc)

The Moset has multiple ees running on MSS which has at least one MH participating in the Moset kernel. The ee acts as an se with respect to the MHs which are within its cell. Any MH within the coverage area of the ~SS, which wants to participate in the sharing of resources, will spawn a set of computing entities to the cc running on that MSS. The cc collects the set of computing entities spawned and registers with the sc. The cc takes care of multi-casting the data set to be rendered to the participating MHs and also maintains the history of the execution that takes place in the MH within its cell.

The cc also takes care of the mobility of the MHs. When an MH takes the frames for rendering and moves out of the cell and enters another cell, then the new MSS, through hand-off, will be able to inform the cc of the old MSS. If the new MSS already has the cc daemon running, then it continues with the process by exchanging the information among the ccs. In case the new MSS does not run the cc daemon, then it gets registered with the sc and runs the cc.

Time-outs, Mobility and Fault Tolerance

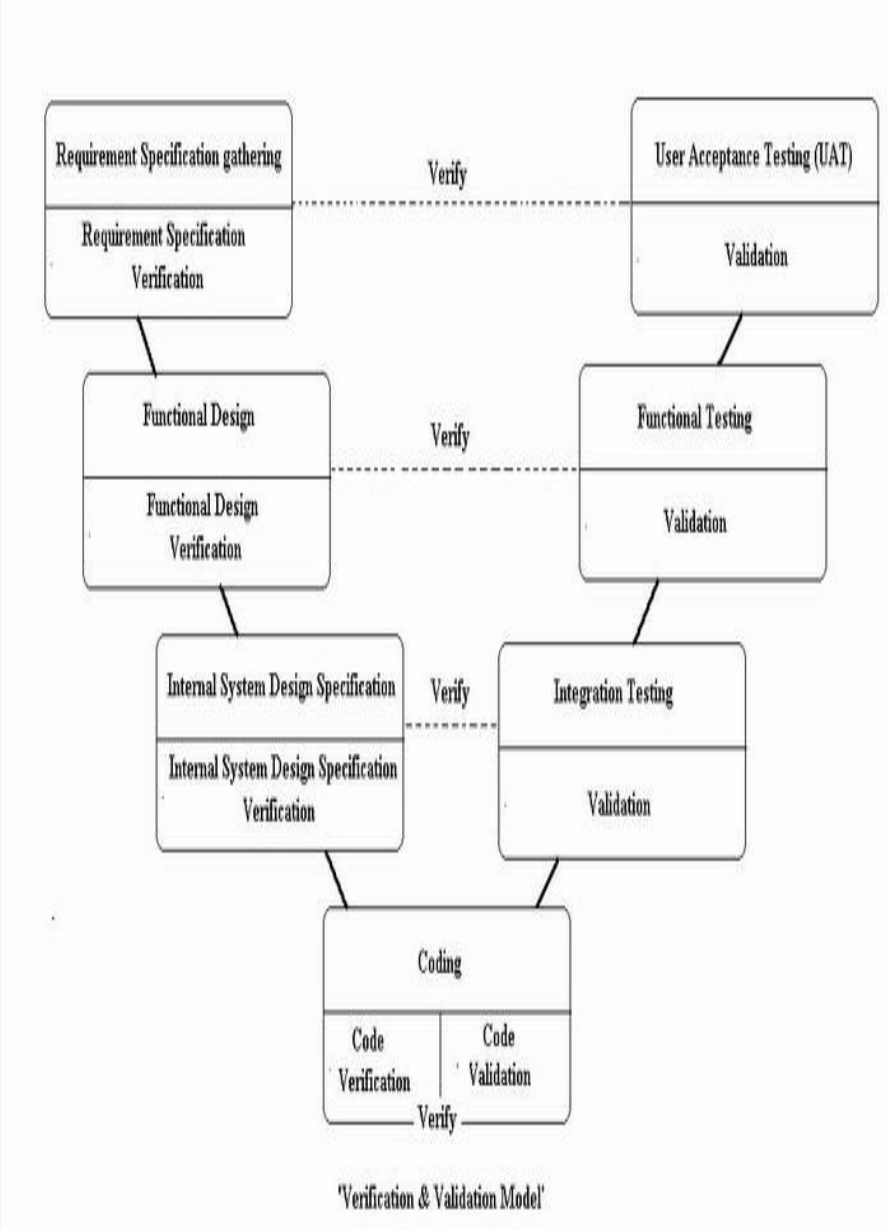
The timeliness issue is an important issue, especially in real-time systems. However, in cluster computing systems, the system is mainly meant for computation-intensive problems like environ-mental modelling wherein the factor of time can be relaxed. But still the workstation cannot take an infinite amount of time for executing the sub-task which it has accepted to execute. In order to handle this, time-out mechanisms are used. The time-outs are maintained by the sc and the ccs. A timer is a data counter that ticks at regular intervals. If the workstation does not return within the stipulated time set in the timer, then the sub-task is re-submitted to some other idle workstation for getting executed or in the worst case, it gets executed in the coordinator. In case of sub-tasks executing in MHs, the cc takes care of the timer. When a sub-task assigned to an MH does not return before the time-out then the cc tries to reassign the sub-task to some other MH within

the sub-group, within the cell or as a worst case, it executes the task itself.

The failure of a remote node is detected by the sc when the lc fails. However, this will work only with static nodes. Mobile nodes may move out of the cell after taking the task for execution and return before the timer time-outs. In this scenario, as the MH was out of coverage for a while, the cc will be able to detect this and cannot decide that the MH has failed. Thus the cc needs to wait until the timer time-outs. This ensures the fault tolerance of the system.

PERFORMANCE

The Moset approach provides parallel programming on a mobile cluster thus improving the utilization of the computing resources of the participating nodes. Moset provides for heterogeneity, fault tolerance and dynamic load balancing to parallel computing. The performance study of the model was done by implementing the distributed image rendering application over the FTEORMP [10]. The FTEORMP is an exactly-once reliable multi-cast protocol. The simulation of FTEORMP was carried out on an object-oriented discrete event simulator in C++ similar to the that used in [15].



'Verification & Validation Model'

POSSIBLE QUESTIONS(2 Marks)

1. Define Cluster Coordinator.
2. What is donor or acceptor.

3. What is FMI?
4. List Grid Computing Applications.
5. What is task splitting?
6. What are the issues in Mobile Clusters?
7. What is the use of Moset?
8. What is Dynamic Load Balancing?
9. What is Fault Tolerance?
10. What is Local Cordinator?

POSSIBLE QUESTIONS(6 Marks)

1. Describe the system topology of Sneha- Samuham Grid Computing Model.
2. Discuss about the issues in Mobile Clusters and Parallel Computing on Mobile Clusters.
3. Describe the Layered Architecture of Sneha- Samuham Grid Compuitng Model.
4. Discuss about the overview of Moset.
5. Write about the Computational Model of Sneha- Samuham Grid Computing Model.
6. Discuss about the implementation of Moset.
7. Describe the design and implementation of Sneha- Samuham Grid Computing Model.
8. Discuss about the Moset Computational Model.
9. Discuss the performance of Sneha-Samuham over MPI and Wide Area Grid.
10. Explain any four issues in Mobile Clusters and Parallel Computing on Mobile Clusters.

Karpagam Academy of Higher Education

Department of CS, CA & IT

Subject: Grid Computing (17CSP205B)

Batch : 2017-2019 **Class: I M.Sc CS**

Objective Type Questions

UNIT IV

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	KEY
1	Parallel programming on workstation cluster mustly follows the _____model	ARC	ARCC	COP	ACRC	COP
2	____are formed by exploiting the existing computing resource on the network to Work together as a single system	Networks	Computer	Node	Clusters	Clusters
3	In order to address there issues, two models, namely_____ were recently proposed	ARC[1] and DP[2]	ARC[5] and DP[4]	ARC[4] and DP[5]	ARC[1] and DP[5]	ARC[4] and DP[5]
4	MCC stands for_____	Model cluster computing	Model cluster constraints	Mobile cluster computer	Mobile cluster computing	Mobile cluster computing
5	Which one is an extension of ARC mode[4] over distributed mobile	Moset	ARMCC	MHS	MCC[7]	Moset

6	_____refers to the delay that is taken for the mobile device to regain in full state When it moves from one will us the other	TImeliness	Timeout	Error	Out of time	TImeliness
7	Timeliner issue is an important issue especially in _____ system	Computing	Real-time systems	Time delay	Real-time	Real-time systems
8	MSS stands for_____	Mobile system support	Mobile station system	Mobile support system	Mobile support station	Mobile support station
9	Heterogeneity can be handled by_____ computation model	Moset	Mobile	Cluster	ARC	Moset
10	_____coven a geographical region	Moset	ARMCC	MSS	MCC	MSS
11	HPF stands for_____	Home power factor	Horse power factor	Horse process function	Horse power function	Horse power factor
12	_____kernal is spread over the node that participating in moset	Computer	System	Moset	Computing	Moset
13	_____model based on the integrahon of mobile and static node as clinters inter onnected by wireless and wired network	Moset	ARMCC	ACC	ARC	ARMCC
14	The moset kernel has one____ in the logically grouped mobile cluster	System cluster	System coordinator	Co coordinator	System computing	System coordinator

15	The moset has multiple _____ running on MSS which has at least one MHParticipating in the moset kernal	Co coordinator	System coordinator	Co computing	System computing	Co coordinator
16	Moblity of the MHS is maintained by	System coordinator	Computer	Co coordinator	Cluster coordinator	Co coordinator
17	LC stands for_____	Local computing	Local computer	Local cluster	Local coordinator	Local coordinator
18	SC stands for_____	System cluster	System computing	System computer	System coordinator	System coordinator
19	A part from loading the processor ____ also log the memory resource	CPU log	Hard disk	CPU	UPS	CPU log
20	_____model provides adaptive parallel execution of tasks over a computational grid.	server model	client model	sneha-samuham model	client-server model	sneha-samuham model
21	The sneha-samuham model the computer donates its computing power is called a_____	accepter	donor	client	node	donor
22	The sneha-samuham model the computer which makes use of another computers computing power is called	acceptor	donor	node	client	acceptor
23	The number of donors should be much higher as compared to the number	LAN	cluster	acceptor	client	acceptor

24	_____can act either as a 'donor' or as an 'accepter' or both as an accepter and a donor	client	server	LAN	node	node
25	The collection of nodes with in an individual LAN is referred to as a_____	accepter	cluster	client	donor	cluster
26	'cc' stands for_____	cluster computing	cluster co-ordinator	client co-ordinator	cluster constraint	cluster co-ordinator
27	NAT stands for_____	network address translation	network asynchronous	network address tranmission	node address translation	network address translation
28	Sneha-samuham model is viewed_____ tired architecture	three	five	four	two	five
29	The runtime environment of the system contains _____components that run as daemons on the nodes	three	five	one	six	three
30	Every 'cc' runs a daemon called_____	cluster co-ordinator	cc daemon	cluster computer	cluster daemon	cc daemon
31	API stands for_____	accessing programming interface	application programming interface	accessing program interface	application processor interface	application programming interface
32	FMI stands for_____	friend machine index	free machine interface	friend machine	free machine index	friend machine
33	FCI stands for_____	friend computing	friend cluster table	free cluster table	free computing	friend cluster table
34	The computation table model follows the_____model	master worker process model	sneha samuham model	client server model	ARC model	master worker process

35	The cc of each cluster maintains a table called_____	friend computing table	free cluster table	FCT	free computing table	FCT
36	The computational capability of a node over the grid can be measured from the factor called	GCCF	SSGF	SSF	GCF	GCCF
37	_____recieves tasks submitted by the user interface and execute them on its friend machine'	donor	acceptor	node	client	acceptor
38	'STT' stands for_____	system tasks table	system target table	sub task table	sub target table	sub task table
39	A donor daemon interaets with its local cc,local accepters and_____spawned by it for executing sub tasks	worker processes	worker processor	window processes	window processor	worker processes
40	A threshold policy is used	eight	one	three	five	three
41	The load balancing service	THEN	ALL	AND	NOW	NOW
42	The life cycle service	MOVE( )	START( )	CALL( )	MALLOC( )	MOVE( )
43	By filtering the method on	add( )	delete( )	rename( )	update( )	update( )
44	MNO uses object wrappers	proxies	plugs	pumbs	pide	proxies
45	Message filters can be	error	save	run	debug	run
46	The filters makes	bind( )	wind( )	sind( )	mind( )	bind( )
47	The CORBA	micro	macro	minicro	mico	mico
48	The mediator is called	System Coordinator	Cluster Coordinator	Coordinator	Process Coordinator	Cluster Coordinator
49	Sneha Samuham Model is tired architecture	two	three	four	five	five
50	_____Model is Five tired architecture	Sneha Samuham	ARC	Mobile cluster computer	ARCC	Sneha Samuham

51	Each CC includes name of other CC in its _____table.	Friends Cluster	GUI	API	User Interface	Friends Cluster

UNIT-V

SYLLABUS

.

Distributed Simulated Annealing Algorithms for Job Shop Scheduling - Implementation. Parallel Simulated Annealing Algorithms - Simulated Annealing (SA) Technique – Clustering Algorithm for Simulated Annealing (SA) – Combination of Genetic Algorithm and Simulated Annealing (SA) Algorithm - Implementation. Epilogue : DOS Grid: Vision of Mobile Grids - Mobile Grid Monitoring System – Healthcare Application Scenario.

Introduction

Distributed algorithms represent the algorithmic formulation of Distributed Problem Solving (DPS) [6]-[9]. DPS can be termed as co-operative problem solving by a loosely coupled network of problem solvers. The main purpose of distributed algorithms is to exploit the processing power of a number of nodes on a network. Since the SA technique for JSS is inherently sequential and -highly compute-intensive, distributed algorithms for the SA technique for JSS can make the technique applicable for large-scale problems. Two different approaches to the development of distributed algorithms for SA technique for JSS have been contemplated. One approach is to divide the problem space into independent

sub-tasks. The algorithm based on this approach is termed as the Locking Edge algorithm. This algorithm is modified for large problem sizes. The other approach involves distributing the reduction rate of the temperature among various nodes of the network. The algorithm based on this approach is called the Temperature Modifier algorithm. These algorithms have been explained in the subsequent sections

DISTRIBUTED ALGORITHMS FOR JOB SHOP SCHEDULING

This section describes the development of distributed algorithms for JSS, using the SA technique. Initially a sequential algorithm [1], [5] is presented which will be modified subsequently to develop distributed algorithms

Sequential Algorithm

The sequential algorithm involves the following major steps:

1. Finding all initial schedules,
2. Evaluating cost of the schedule,
3. Finding the critical path,
4. Generating a neighbour.

These are discussed in detail below.

Initial Schedule

Given a disjunctive graph G = (V, A, E) for solving the problem, an initial schedule is generated. The Giffler and Thompson

algorithm [2] is employed for this purpose. The algorithm attempts to construct the schedule by considering all the operations (n) on all the machine (m), with the criteria employed being the earliest starting time and the processing time of each of the operations. At each stage an operation not yet included in the schedule and requiring a minimum time is chosen and included in the partial schedule. The partial schedule becomes a complete schedule when all the operations of the jobs are included in the schedule. The generated schedule can be represented as a digraph.

Cost Function

After obtaining the digraph representing all initial schedules, the earliest and the latest start times of each of the operations in the graph are calculated. The Critical Path Method (CPM) is used for this purpose. The makespan is the earliest start time or the latest start time of the last operation. This forms the cost of the schedule.

Critical Path

After evaluating the cost function, the critical path in the digraph is identified. The critical path can be defined as a set of edges from the first vertex to the last vertex which satisfy the following properties:

(a) The latest start time and the earliest start time of each vertex on the edge must be the same.

(b) For the same edge U -7 v, the sum of the start time and the operation time of u must be equal to the start time of v.

An edge in the critical path is reversed to generate a neighbour and this is discussed in the next section.

Generating a Neighbour

The neighbourhood of a schedule can be defined as a set of schedules that can be obtained by applying the transition function on the given schedule. Neighbourhoods are usually considered by first choosing a simple transition function. A transition in the case of a JSS problem is generated by choosing the vertices v and w (as given in [1]). The following facts need to be considered.

(a) v and ware any two successive operations performed on the same machine k;
(b) (v, w) E Ej is a critical edge, i.e. (v, w) is on the longest path of the digraph.

A neighbour is generated by reversing the order in which v and ware processed on the machine k. It has been shown that by using this transition function, it will be possible to eliminate infeasible solutions and also keep non-decreasing paths out of the search space [1].

Thus, in the digraph such a transition results in reversing the edge connecting v and wand replacing the edges (u, v) and (w, x) by (u, w) and (v, x) respectively, wherein u is the previous operation to '» on the same machine, and x is the next operation to w on the same machine.

Distributed Algorithms

They can be termed as:

1. Temperature Modifier Algorithm,
2. Locking Edges Algorithm, and
3. Modified Locking Edges Algorithm

They are explained in detail below.

Temperature Modifier Algorithm (TMA)

The choice of the temperature modifier in the sequential algorithm affects the probability of the algorithm getting struck at a local minimum. A low value for the modifier makes the algorithm fast but the probability of the algorithm getting struck at a local minimum is high. In order to reduce this probability, the sequential algorithm can be simultaneously executed on different nodes.

Locking Edges Algorithm (LEA)

It can be observed that the TMA does not result in much improvement with respect to the execution time on the computer. The LEA has been developed to improve this. This algorithm generates sub-tasks by 'locking' edges in the digraph. The term 'locking' can be defined as marking an edge of the digraph such that its orientation cannot be changed. If the number of edges locked in the digraph is m then we can generate 3 m equal sub-tasks. This is equivalent to dividing the entire search space into 3m divisions. For example, if the number of locked edges in the digraph is one, then we can generate three

subtasks. These sub-tasks can be obtained as follows:

(a) In the first sub-task, the search space consists of all possible orientations of the edge except that of the locked edge. The orientation of the locked edge remains intact.
(b) In the second sub-task, the orientation of the locked edge is reversed.
(c) In the third sub-task, the locked edge is removed.

The generation of three sub-tasks is illustrated by taking example 1 (please refer to Table 9.1). The initial schedule generated for the example problem is represented in Fig. 9.1(a). In this schedule, the edge 15-5 is locked to generate the sub-tasks. Hence the initial schedule in Fig. 9.1(a) with edge 15-5 locked forms the starting schedule for sub-task 1. The starting schedule for the

Select appropriate temperature modifiers for different nodes.

Run the following sequential algorithm with the chosen temperature modifier on each node.
(a) Generate the initial schedule, given the processing times or all operations and the machine order for each job.

(b) Repeat
Counter =0.
Compute the cost of the initial schedule [t[i]];
Repeat
Calculate the critical path;
Generate the neighbourhood;
Compute the cost of the Generated schedule [t[j]];

Accept or reject the generated schedule with the probability

min(l.exp (-{t[j] - t[i]}/T));

Until (counter=number_of_rune_at_a_temperature);

T = T + t_modifier;

until (T = 0 V Minimum schedule not changed for a long time);

3. Send the result back to the central node

second sub-task is generated by reversing the locked edge 15-5 in the initial schedule and the schedule thus obtained is given in Fig. 9.1(b). The starting schedule for the sub-task 3 is generated by removing the locked edge 15-5 from the initial schedule and this is shown in Fig. 9.1(c).

The above concept can be further extended to a case in which there can be m locked edges. Figure 9.2(a)-(c) explains the method of generation of the sub-tasks. The generated sub-tasks are assigned to the cooperating nodes in the Distributed Problem Solving (DPS) network. The detailed algorithm is presented in Table 9.3.

The edges to be selected for locking are chosen at random. The edge chosen affects the division of the search space and influences the quality of the solution in some cases. This is discussed in the next section.

1. Generate the initial schedule with the input given.
2. Choose the number of edges to be locked, say m.
3. Generate the sub-tasks depending upon the number of locked edges i.e. for m locked edges 3 power m sub-tasks are generated.

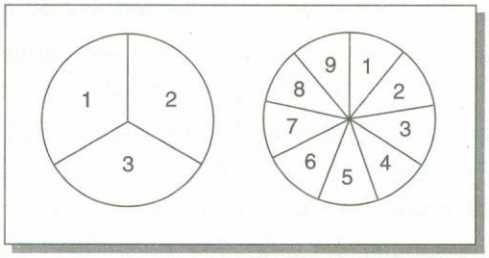
4. Assign each of the sub-tasks to the cooperating nodes, pass information about the locked edges.
5. Wait for the results from the cooperating nodes.
6. Choose the optimal cost solution.

Modified Locking Edges Algorithm (MLEA)

It is observed that as more and more edges are locked, it results in a decrease in the performance with respect to the optimal scheduling cost in some cases. This is explained theoretically in the subsequent paragraphs.

A new term called 'collision' is defined in the case of the locking edges version of the distributed algorithm. In the locking edges version, the search space is divided equally among all the co-operating nodes. For a three-node distribution, the search space can be represented as in Fig. 9.3(a) and for a nine-node version, it can be represented as in Fig. 9.3(b). One of the edges on the critical path is chosen at random for generating a neighbour. If this selected edge (say el) affects the locked edge (say e2), then such a situation is termed as 'collision of edge el with locked edge e2'. In such cases, the edge el is rejected and a new edge is selected at random to generate a neighbour. It can be interpreted as a node trying to 'penetrate' into the search space belonging to another node. It can be observed from the figures that these collisions will be more in the nine-node case compared to that in the three-node case. This results in a marginal increase in the schedule cost as compared to the previous case. In cases where the solution corresponding to the minimum exists on the boundary

or in the search spaces of two nodes, there is less likelihood of it being reached in the nine-node case because of collisions



Search Space Division:

(a) Three-node Distributlon (b) Nine-node

In order to minimize the affect of collisions, a modification to the 'locking edges' version is attempted. This can be explained as follows. Consider a case wherein a single edge is locked. Here there are two sets of nodes. The first set of nodes consists of three nodes and will be assigned the same tasks as in the 'locking edges' case. The second set of nodes, also consisting of three nodes, are assigned tasks in such a way that the boundary points of the previous set of search spaces become the active search spaces for these nodes. As pointed out previously, this is done mainly to reduce the effect of collisions. This can be extended to any number of locked edges. The detailed algorithm is given in Table 9.4.

Table 9.4 Modified Locking Edges Algorithm

1. Generate the initial schedule with the input given.
2. Choose the number of edges to be locked, say m.
3. Generate the first set of sub-tasks (3 power m).
4. Assign the sub-tasks to the cooperating nodes. Pass information about the locked edges also.
5. Generate the second set of sub-tasks (3 power m).

6. Assign the sub-tasks to the cooperating nodes. Pass information about the locked edges also.
7. Wait for the results from the cooperating nodes.
8. Choose the optimal cost solution.

The edges to be locked are selected at random and the selected edges may be the cause for collisions. However, it is not easy to predict beforehand this effect of a particular edge on collisions. Hence, in the absence of such knowledge, the modified locking edge version guarantees that the search space is divided to avoid excessive collisions at least in one set of the search space division

Implementation

These algorithms have been implemented on the Distributed Task Sharing System (DTSS) [7] running on a network of Sun Workstations having three servers of Sun 3/60 and 15 clients or Sun 3/50 connected together by a thin Ethernet. The DTSS has been developed around a message kernel. The message kernel is implemented by using datagram sockets. Messages across nodes are transferred by these datagram's. The message kernel provides support for reconfiguring the nodes on the network, and for sending and receiving the task award and the result messages. The nodes on the network are initially configured such that one of the nodes is identified as a central node. During the initial configuration, many other required client nodes are also identified. The central node has the responsibility of dividing the search space and of communicating the tasks to the client nodes through task award messages. After receiving the task award messages, the client nodes execute the required task and send back the results through the result messages. The network is highly flexible and can be reconfigured with any number of client nodes.

The messages are retransmitted in the case of loss of message packets in transmission. This is detected by the non-receipt of an acknowledgement for the packets sent within a specified time-out period. The node failures are also detected in a similar fashion. In case of node failures, the corresponding sub-tasks are assigned to other client nodes available on the network.

Before proceeding further on the implementation details, two terms are defined:

- Central Node: This node holds the responsibility of task division, task award to client nodes, receiving result messages and synthesis of the final solution from the results obtained from the client

- Client Node: This node solves the sub-task assigned to it and returns the result

Temperature Modifier Algorithm (TMA)

The implementation of the algorithms has been carried out by a central node and a set of client nodes. The central node generates the initial schedule, given the processing times of all the operations and the machine order for each job. This node then sends the initial schedule and also different temperature modifier parameters to each of the client nodes on the network. The client nodes execute the sequential algorithm (Table 9.2) with their corresponding temperature modifiers and send back the result to the central node.

After receiving the results from the client nodes, the central node chooses the solution that has the minimal cost.

The quality of the solution generated is influenced by the cooling rate. We employ a three parameter cooling schedule, as seen in [1]. The parameter delta controls the rate of cooling. A lower value of delta reflects a slower cooling rate and consequently the algorithm takes more time. The value of delta employed is in the range of 10-1-10-4. Hence, it is important to choose an appropriate value of delta for each node in the network. A marginally high value for delta than that employed in the sequential algorithm can be chosen for TMA. For example, if a delta value of 10-2 is employed for a sequential algorithm, for a TMA case, a delta of 0.5 x 10-1 may be chosen. As a higher cooling rate affects the quality of the solution, it may be appropriate to employ TMA only in small size problems, where LEA or MLEA doesn't give better results.

Locking Edges Algorithm (LEA)

As in the above implemeritation, here too a central node and a set of client nodes participate in the execution of the problem. In this case, the number of client nodes is equal to the number of generated sub-tasks. The generated sub-tasks are based on the number of locked edges.

Initially, the central node generates the initial schedule and depending upon the number of locked edges, it generates the sub-tasks and assigns each of the latter to one of the client nodes. The

client nodes execute the sub-task and return the result to the central node. The central node synthesizes all the results and chooses the minimum cost solution as the best one.

Modified Locking Edges Algorithm (MLEA)

In order to implement this algorithm, a central node and sets of client nodes are identified. The central node divides the search space into sub-tasks and assigns them to one set of client node. This process is same as the one described in Locking Edges algorithm. However, in the case of the MLEA, the search space is again divided by the central node such that the boundaries of the search space in the earlier set become the active search spaces in this case. The sub-tasks generated in this process are assigned by the central node to the next set of client nodes. Thus many sets of client nodes participate in problem solving in this case.

PARALLEL SIMULATED ANNEALING ALGORITHMS

SIMULATED ANNEALING (SA) TECHNIQUE

Often the solution space of an optimization problem has many local minima. A simple local search algorithm proceeds by choosing a random initial solution and generating a neighbour from that solution. The neighbouring solution is accepted if it is a cost-decreasing transition. Such a simple algorithm has the draw-back of often converging to a local minimum. The SA algorithm, though by itself a local search algorithm, avoids getting trapped in a local minimum by also accepting cost-increasing neighbours with some probability. In SA, first an initial solution is randomly

generated, and a neighbour is found and accepted with a probability of min (1, exp (2d/T)), where d is the cost difference and T is the control parameter corresponding to the temperature of the physical analogy and will be called temperature. On slow reduction of temperature, the algorithm converges to the global minimum, but the time taken increases drastically ..

SA is inherently sequential and hence very slow for problems with large search spaces. Several attempts have been made to speed up this process, such as development of parallel SA techniques and special purpose computer architectures

Parallel Versions of SA

Parallelism in SA can be broadly classified into two approaches- single-trial parallelism and multiple-trial parallelism [5]. But these methods are highly problem-dependent and the speed-up achieved depends wholly on the problem at hand. Another taxonomy divides parallel annealing techniques into the following three major classes:

1. serial-like algorithms,
2. altered generated algorithms, and
3. Asynchronous algorithms [1].

Each class of the algorithm makes some trade-off among cost function accuracy, state generation, parallelism, and communi-cation overhead. High-performance special purpose architectures show the promise of solving computationally expensive appli-

cations without expensive supercomputers and include specially designed computer architectures to suit the annealing algorithm [11].

CLUSTERING ALGORITHM FOR SIMULATED ANNEALING (SA)

Experiments on the SA technique have shown that a good initial solution results in faster convergence. Similar observations have been made in [13]. The proposed distributed algorithms take advantage of this observation. Initially, the n nodes of the network run the SA algorithm by using different initial solutions. After a fixed number of iterations, they exchange their partial results to get the best one. All the nodes accept the best partial solution and start applying the SA technique for that best partial result. They again exchange their partial results after some fixed number of iterations. After repeating this process for a pre-defined number of times, each node works independently on its partial result. The complete algorithm is given in Table 10.1.

Input to the algorithm:

n = Number of the nodes in the network.

p = Exchange parameter for partial results.

r = Reduction parameter for the number of iterations before exchange of partial results.

i = Input graph for scheduling.

Coordinator node algorithm:

1. Distribute the n random initial solutions to the n nodes and wait.
2. Upon receiving the first converged result from any of the nodes, stop SA on other nodes.

Worker node algorithm:

- Accept initial solutions from the coordinator.
- repeat

2.1. Execute SA for p iterations. Exchange partial results among the worker nodes. Accept the best partial result.

2.2. p = P - fA' (loop iteration number).

until (p = 0).

3. Execute SA by using the best solution found as the initial solution.
4. Send the converged value to the coordinator.

COMBINATION OF GENETIC ALGORITHM AND SIMULATED ANNEALING (SA) ALGORITHM

Experiments have shown that a good initial solution for SA improves both the quality of the solution as also the execution time. Genetic algorithms try to improve a set of 208 Ram, Sreenivas, and Subramaniam solutions rather than a single solution. Since we require n initial solutions for distributing among n nodes, we choose to combine SA with GA.

Genetic Algorithm

In GA [11], an initial population consisting of a set of solutions is chosen and then the solutions are evaluated. Relatively more effective solutions are selected to have more offsprings, which are in some way, related to the original solutions. If the genetic operator is chosen properly, the final population will have better solutions. GA improves the whole population. SA aims at producing one best solution. For the distributed SA implementation, we require

several good initial solutions to ensure the fast convergence of SA. We chose GA for obtaining the required number of good initial solutions. The operator used for generating offsprings in JSS is related to the processing order of jobs on the different machines of the two parent solutions. Let PO1l, PO12, ... , POlm be the processing orders of jobs on machines 1,2, ... , m in parentI and P021, P022, ... , P02m be the processing order on machines 1,2, ... , m in parent2. If random (1, m) = i, then processing orders in childl and child2 are POll, ... , P01~ P02i 1 1, ... , P02m and P02I, ... , P02,POli11, ... , POlm respectively. After getting the offspring, a check is made to see if there are any cycles in the offsprings and if there is one, the operation is performed once again by generating another random number. A cycle in a state indicates an invalid schedule. The pseudo-code for the GCA is given in Table 10.2.

Table 10.2 Genetic Clustering Algorithm (GCA)

1. Central node generates n initial solutions using GA. It runs GA for fixed number of iterations, t.

1.1 Choose initial population of fixed size and set i = 1.

1.2 while (i <= t)

begin

1.2.1 Apply the operator on the two parent schedules chosen randomly to produce two offspring and replace the parents by the best two out of the four schedules.

1.2.2 i = i + 1

end

2. Central node sends n best solutions chosen to the n remote worker nodes.

3. Each worker node runs the SA algorithm by using the initial state received.
4. Upon receiving a converged result from one of the worker nodes, the central node stops execution.

IMPLEMENTATION OF THE ALGORITHMS

Both the above algorithms have been implemented by using a platform called DiPS (Distributed Problem Solver) [3] running on a network of 18 Sun workstations. It is built on a communication kernel. Using the kernel, it is possible to send task award messages, task result messages, configure messages, and partial result messages, among the various nodes of the DiPS network. The full implementation details of both algorithms are given in the subsequent sections.

Implementation of the Clustering Algorithm (CA)

In the CA, the central node executes the code in Table 10.3 and the worker nodes the code in Table 10.4. The algorithm for SA is given in Table 10.5.

Table 10.3 Clustering Algorithm for the Central Node

1. Initialize ( ).
2. Generate n random initial states and assign to the n nodes of the network.
3. Wait for results.
4. Output_Results.

Table 10.4 Clustering Algorithm (CA) for the Worker Node

1. Get the sub-task from the central node and p, the exchange parameter.
2. while (p> 0)

begin

2.1 Simulated_annealing (n).

2.2 Send the best solution obtained to the central node.

2.3 p = P - (loop_iteration_ value)8 r.

end

3. Run SA.
4. Send the converged value to the central node

Simulated annealing (n)

begin

1. Set t = Initial_temperature
2. repeat

2.1 Counter = O.

2.2 **repeat**

2.2.1 Compute the cost of the schedule (f[i)).

2.2.2 Find the critical path schedule.

2.2.3 Generate a neighbour and compute the cost of the neigbour (f [j)).

2.2.4 Accept or reject the neighbour with a probability of min(1, [(f[i] - [(1)/1).

2.2.5 Increment counter.

until (Counter = Number of iterations at t).

3. t= t* temp_modifier.
4. After every n iterations exchnage results and accept the best schedule found.

until (shopping criteria)

end.

IMPLEMENTATION OF GENETIC CLUSTERING ALGORITHM (GCA)

In the case of GCA, first the genetic algorithm is run on the central node to get the required n initial solutions. These initial solutions are used by the n client nodes of the distributed systems as a starting solution for the SA algorithm. (The code that is executed on the central node! is the same as the code in Table 10.3 except that in step :;p the n schedules are the best n solutions chosen from the population after applying GA. The genetic algorithm starts with an initial population. It then performs the crossover operation and the population is updated. This is repeated a number of times.

EPILOGUE DOS GRID: VISION OF MOBILE GRIDS*

DOS Grid

Overview of the Mobile Grid

The mobile grid is visualized as a cluster of clusters. In the proposed model, the nodes are encapsulated as objects called 'Surrogate

Object' (SO). The SO encapsulates all the characteristics and properties of a device fully, such as the computing power, memory availability, bandwidth, and all other resources and services associated with the device. The representation of the characteristics of the devices in the SO is made as attributes, methods and sub-objects. The attributes of the SO include the computing capability of the node, the memory capability and the bandwidth of the medium by which the node is connected. The methods and the sub-objects of the SO represent the services and other resources that are offered by the node. In addition, each SO encapsulates the security policy and agreement for each of the services that is associated with that node, which will specify how and by whom the service may be used. By encapsulating the participating nodes, in distributed objects, the grid is transformed from a collection of nodes, offering and consuming services, into Distributed Shared Object (DSO) space.

Each cluster is coordinated with a designated node acting as a Cluster Head (CH). The CH maintains all the repositories related to the trading and naming services of the DSO, and handles the service discovery [3]. The CHs coordinate among the other neighbouring CHs in a peer-to-peer fashion. The mobile grid as DSO space is shown in Fig. 11.1. In order to have a unified design, the static nodes are also represented as objects in the DSO. As the MSS are already loaded with maintaining the information related to the MH which are within its cell, the neighbouring static node is designated as the CH of the cluster. In some cases the MSS and CH may be the same.

The proposed SO model of the grid enhances the availability and helps in providing the current location of the mobile devices. When the mobile device acts as an information service provider, depending upon the nature of the service requested, the monitoring system of the mobile grid decides whether to contact the SO of the corresponding device or the device itself. Contacting the device directly will lead to consumption of the constrained resources like battery and bandwidth. In the case of the services offered by the SO, it would suffice if the corresponding SO is contacted to get the information. In addition, the SO can be replicated to prevent congestion in the network and to improve scalability of the system. The major advantage of the paradigm is that the network connectivity need not be continuous because connections are required only to inject SO from mobile nodes into the wired network. With the SO being fully autonomous, users can access services even if the node disconnects because the SO delivers the results upon re-connection. The proposed model virtualizes all the resources and services offered by the participating nodes as services

The proposed approach significantly helps in realizing a distributed and decentralized infrastructure of SOs that work on behalf of the participating devices and are hosted by the wired network. With the SO, the MH movements do not affect service provisioning as the entire state of the device is stored and maintained. The model helps in achieving the properties of dynamicity, asynchronicity, autonomy and security.

Some memory-constrained devices such as mobile phones could also be participating in the mobile grid. In such a case, the user of

the mobile device should be able to store the data in some other databases located elsewhere in addition to the data stored locally. Thus, the proposed mobile grid can also efficiently handle storage and data access from federated databases. The meta-data contains information about file instances, the contents of file instances, and the various storage systems contained in the data grid. These meta-data usually refer to application meta-data. These meta-data are wrapped with wrappers and made as objects in the DSO. The meta-data objects are registered by using trading services

Scalability and Consistency Issues

Middleware services such as naming and trading as well as replica object management in the grid are handled through a wide-area shared object space that we have built named as Virat [2]. Virat uses an independent checkpointing and lazy reconstruction mechanism to handle failures of object repositories. The object repositories (one per cluster) are responsible for the cluster level management of replicas. Communication between the object repositories themselves is through a peer-to-peer protocol. This is useful for locating objects or services across clusters. Virat also uses a data-centric concurrency control mechanism to realize various consistency schemes such as serializability and causal consistency. Virat has been extended to a shared event space wherein events can be created, published and subscribed. Events can be delivered in causal or serializable orderings on the basis of application requirements.

Scalability is a key issue in distributed systems, especially in mobile grids as the number of devices can be quite high. One of

our key observations is that scalability, consistency and availability need to addressed together, not in isolation from each other. It is well-known that in the presence of network partitions, both consistency and availability cannot be completely attained in purely asynchronous systems [4]. Availability has been quantified in [5] and its trade-off with consistency has been studied. However, both [4] and [5] do not address the scalability issue.

One dimension of consistency is the 8 value, the number of updates that can be buffered by a replica before updating other replicas. This has been related to availability in [6]. However, another dimension of consistency, namely update ordering, has not been considered. Various consistency criteria can be realized on the basis of update ordering. These include serializability, causal consistency and Pipelined Random Access Memory (PRAM) consistency. The idea is that given these two dimensions of consistency, there is a trade-off between scalability and availability. We have come up with an upper bound on scalability (in terms of productivity) for a given availability and the two dimensions of consistency, for specific workload and faultload combination. This theoretical upper bound is difficult, if not impossible, to achieve in practice. We are currently conducting performance measurements to evaluate the practical scalability of Vir at. We are also optimizing Virat to make the scalability closer to the theoretical upper bound.

MOBILE GRID MONITORING SYSTEM

Monitoring is the act of collecting information concerning the characteristics and status of resources of interest. Grid resources may dynamically join and leave, resulting in varying membership over timJ Even in fairly static conditions, resource availability is subject to failures. Due to such a transient nature of the grid, the system must support the finding and keeping track of the required resources dynamically/This is the main purpose of Grid Information Services (GIS). This requires a process called monitoring, which systematically collects the information regarding the current and past status of the grid resources to satisfy the users' need.

Several groups are developing grid monitoring systems [7]. In most of these monitoring systems, the monitoring system is a part of the discovery system However, in our proposed approach, the publishing, discovery and handling of resources are done by the DSO structure which is considered as a platform to build the mobile w,id infrastructure. The monitoring system resides over the shared space in the peer-to-peer layer.

In order to incorporate intelligence into the mobile grid, we ~a monitoring system that manages the vast heterogeneous resources including those offered by the mobile devices across administrative domains. The mobile grid monitoring system essentially helps in scheduling and task allocation for parallel computing, in enforcing Quality of Service (QoS) and Service Level Agreements (SLA), in identifying the cause of performance problems, optimized resource usage and fault detection, in addition to building prediction models of mobile device movement

Different kinds of data are collected from the different components that make up the mobile grid.

The monitoring information traffic can be huge and with only one monitoring manager, it becomes a bottleneck for the entire system. Hence, the hierarchical monitoring structure was considered by most of the existing monitoring systems. However it is proved in [8] that the scalability of peer-to-peer structure is better than a hierarchical structure. Hence the eHs associated with each cluster, which forms the peer-to-peer overlay, share the monitoring activities among themselves.

The proposed Mobile Grid Monitoring System provides the following features:

- Mobile Host (MH) monitoring: Data regarding the location of the device, connectivity, mobility, signal strength, etc. at different times of the day can be monitored and used for predicting the future values of these parameters. This could help in characterizing the movement pattern of the devices. This information can be used in scheduling to provide guaranteed QoS and SLA. In addition, information regarding load, battery life, available memory, etc. is also collected. When these parameters cross the threshold value, the MH sends the associated information asynchronously to the monitoring system. The mobile node parameters are also sent on the basis of the request from the monitoring system. Mobile device monitoring can be done by monitoring the SO associated with the mobile device.

- Providing dynamic space allocation and file management on shared storage components on the grid: They provide storage reservation and dynamic information on storage availability for data movement, and for the planning and execution of grid jobs. This is designed to facilitate the effective sharing of files, by monitoring the activity of shared files, and making dynamic decisions on which files to replace when space is needed.

- Data store monitoring: The access patterns of the data can be observed to assist in replica management. Data regarding data movement overhead and data lifetime can also be collected. This can be used to decide whether the task must be scheduled near the data store or data moved near the task. It is also possible to maintain the meta-data of the different data stores and their elements in order to assist in storage management, handling requests, etc. The meta-data could also contain information about the locks on the different data elements
- Static Host (SH) monitoring: We collect the CPU load, available memory, bandwidth details, etc. from the host. The hosts can either be monitored continuously or on being triggered by detecting network activity on a particular port. This information can be used by the scheduler to choose the donor.

- Process monitoring: Changes in the process state can be monitored.
- Application monitoring: Checkpoints can be inserted into the application to capture the intermediate state of the application and the data required for performance analysis.

Any query to a mobile device is routed through its associated SO. The object reference of the SO is obtained from the naming service of the DSO by querying for the IP of device as name [3]. The query is forwarded to the MH, with the help of the current location information available at the SO. In case the MH is not reachable due to reasons like out-of-coverage, the query is handled at the SO level. The monitoring system stores the historical data related to the MH movement pattern in the repository for characterizing the movement pattern. This helps in scheduling tasks and storing data.

In our proposed model, the monitoring system will observe the requests and try to find out from which place and for which file the requests have come. Based on the observations, the lifetime of the file, data transfer time and the available space in the data store, the system chooses the optimal location. This is done by maintaining the history in the monitoring system about the requested queries and the lifetime of the data which they are accessing to predict the location as well as to initiate the replica dynamically. Although the probability of prediction is high, still even if it fails, it affects only the performance of the system, not its correctness.

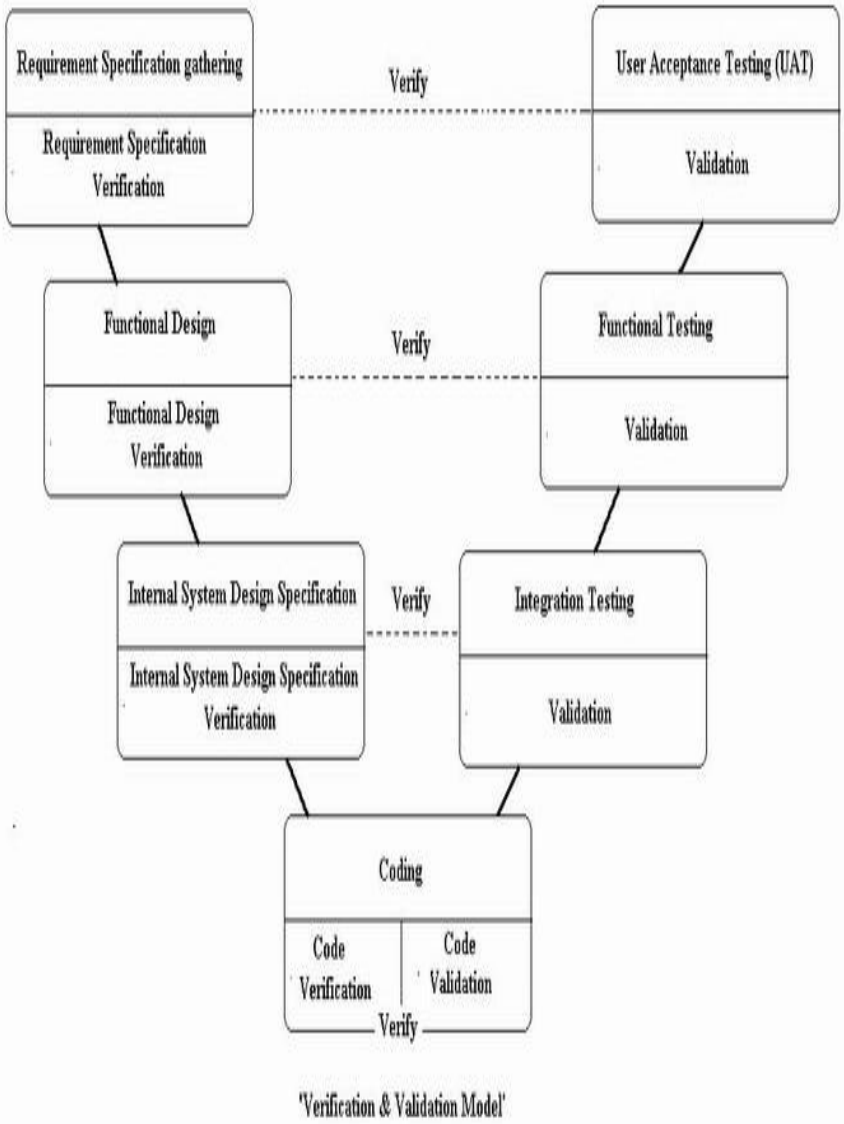
The monitoring daemon, which collects the monitoring data, runs on the static nodes of the mobile grid. Mobile nodes run the exporter daemon, which exports the monitoring data of the mobile nodes to the mobile support station. The monitoring system resides on the CR of the cluster and maintains a monitoring data repository to store the data observed within the cluster. As the

CRs interact among themselves over a peer-to-peer overlay, the monitoring system is scalable, fault-tolerant and ensures automatic reconfiguration.

HEALTH CARE APPLICATION SCENARIO

One of the possible scenarios wherein we can envision the integration of sensors, mobile nodes and data grids is the following health monitoring and treatment example. Patients could have sensors embedded inside their bodies to keep checking for specific data such as blood pressure, cholesterol level, etc. When local sensor data exceeds certain pre-defined thresholds, the sensor passes the data to a nearby mobile device, possibly the patient's hand-held device. The device is part of our mobile grid and can utilize and provide services. It uses the grid to aggregate data from multiple sensors from the same patient, uses historical information and some computation to decide if this pattern (combination of data values from various sensors on a patient) is abnormal and requires emergency handling. If this is the case, the details of the patient are collected from the grid and forwarded to a health care centre. Based on the location of the mobile device which sent the data to the grid, the location of the patient is tracked.

A computer system at the health care centre (it is also part of the grid) locates a nearby ambulance by querying the grid. The ambulance carries some mobile devices and can be directed to the patient's location as soon as possible. This computer system



'Verification & Validation Model'

CLASS: I M.SC CS **COURSE NAME: GRID COMPUTING**

COURSE CODE: 17CSP205B UNIT V(Distrubuted Algorithms) BATCH-2017-2019

POSSIBLE QUESTIONS(2 Marks)

1. What is mobile Grid
2. Define Simulated Annealing Technique.
3. What is DOS Grid?
4. Write the use of Genetic algorithm.
5. Write the need of Clustering algorithm.
6. What is Genetic Clustering Algorithm?
7. List any two application of Mobile Grid.
8. Define JSS.
9. What is scalability issue?
10. What is consistency issue?

POSSIBLE QUESTIONS(6 Marks)

1. Explain the implementation of Distributed Simulated Annealing Algorithms for Job Shop Scheduling.
2. Give an overview of Mobile Grid.
3. Explain Simulated Annealing Technique.
4. Give an overview of DOS Grid.
5. Brief about the Clustering Algorithm for Simulated Annealing.
6. Write about Mobile Grid.
7. Brief about the combination of Genetic Algorithm and Simulated Annealing Algorithm.
8. Describe a sample health care application scenario for Mobile grid.
9. Discuss about the implementation of Clustering Algorithm and Genetic Clustering Algorithm.
10. Discuss how scalability and consistency are ensured in Mobile Grid.

Karpagam Academy of Higher Education

Department of CS, CA & IT

Subject: Grid Computing (17CSP205B)

Batch : 2017-2019 **Class: I M.Sc CS**

Objective Type Questions

UNIT V

S.NO	QUESTIONS	OPTION 1	OPTION 2	OPTION 3	OPTION 4	KEY
1	JSS belongs (job shop scheduling) to which class of problem	NPP hard optimisation	NP_soft optimization	NP_hard optimization	NPP_soft optimization	NP_hard optimization
2	What is the expanded form of SA	simulated analysis	sorted algoritm	sorting algoritm	simulated annealing	simulated annealing
3	can be defined as a set of the edges from the first vertex to the last vertex	long path	short path	critical path	vertical path	critical path
4	TMA means	temperature method algorithm	time modify algorithm	time method algorithm	temperature modify algorithm	temperature modify algorithm
5	LEA can be expanded as	Lock edge algorithm	Locking Edges Algorithm	Locking End algorithm	Lock End algorithm	Locking Edges
6	In ----------monitoring changes in the process	applications	static host	static applications	process	process
7	NASA's IPG,is in the expand IPG	Information power grid	Inform power grid	Inform poor grid	Information power ground	Information power grid
8	The mobile grid is visualized as a cluster of	clusters	mobile	grid	computing	clusters
9	TSP is expanded as	Travelling Salesman Problem	Travel sale Problem	Traveling sales problem	tools in sales problem	Travelling Salesman
10	CPM stands for	critical path and methodology	critical path method	critical path modification	critical path method	critical path method

11	the nodes holds the responsibility of task divisions,task awards,to	central node	client node	server node	task node	central node
12	Which node solves the sub task assigned to it	central node	client node	server node	vertical node	client node
13	Expand MLEA	Modified lock end algorithm	mode lock edges algorithm	modified lock end applications	Modified locking edges	Modified locking
14	is the act of collectint informations concerning the characteristics and status of resources of	concerning	monitoring and concern	monitoring	both a and b	monitoring
15	MH monitoring stands for	mobile host	mobile home	moving host	move host	mobile host
16	The messages are retransmitted in case of loss of message------- in	data	delivery report	package	cluster	package
17	Expanded form of GCA	Genetic clustring algorithm	Generic cluster Algorithm	Generic clustering	General cluster algorithm	Genetic clustring
18	The messages are --------- in case of loss of message package in transmission	retransmitted	banned	stopping forever	transmitted	retransmitted
19	monitoring access patterns of data and assist in replica management	data stored	mobile host	mobile data	process	data stored
20	is key issue in distributed systems	consistency	availability	both a and b	scalability	scalability
21	JSS stands for	Job shop scheduling	jop somu schedulig	job system scheduling	jop system somu	Job shop scheduling
22	A set of jobs whose operators are to be provided by a set of	MCC	JSS	ARC	ARCC	JSS
23	_____ belongs to the type of local starts	SA	ARC	JSS	MCC	SA
24	DPS stands for	Digital protocol system	Digital processing system	Digital program system	Digital process system	Digital protocol

25	The node holds the responsibilty of task down task award to client node is	Control nodes	client node	sewer node	server node	Control nodes
26	TMA stands for	Temperature modifier algorithm	Temperature modifier assembly	Temperature mode assemble	Temperature measuring	Temperature modifier
27	LEA stands for	Locking edges algorithms	Locking edges assembly	Locking algorithms	Locking easier assembly	Locking edges
28	Cpm stands for	Critical parallel method	critical path method	critical parset method	critical program method	critical path method
29	The node wholes the responsibilty of task decision is called	Central node	Central path	central client	central data	Central node
30	MLEA stands for	Mode language edge assembler	Modified language edges algorithms	Modified locking edges	Modified lock end assembly	Modified locking
31	GIS system stands for	Grid information server	Grid information service techonogly	Grid information	Grid information	Grid information
32	_____ is a interconnected	Parallel computing	Parset	Cluster	several groups	Cluster
33	_____ is the act of collection information concering the characterstic and state of resuming of	several groups	Grid resources	cluster	Monitoring	Monitoring
34	_____ data regarding the block of device connectivity ,mobility strength etc at the	Mobile host modulation	Mobile host monitoring	Mobile handling monitoring	Mobile grids	Mobile host monitoring
35	_____ changes in the presented can be monitored	System	services	Monitoring	client	Monitoring
36	_____is possible to	JBS	SA	Travelling	Simpleprogram,	Travelling
37	The client-server	Simulate	Remote Execution	Remote	Collection of	Remote
38	SA belongs to the type of _____algorithms	global search	breadth first	depth first	local search	local search
39	_____ belongs to the	ARC	ARCC	Simulated	JSS	Simulated

40	_____ algorithms represent the algorithmic	ARC	ARCC	Simulated Annealing	Distrubuted	Distrubuted
41	Distrubuted algorithms represent the algorithmic formulatin of _____	Remote	Distrubuted	local	global	Distrubuted
42	Critical Path can be	points	edges	nodes	paths	edges
43	Critical Path can be defined as a set of edges from	first	second	zero	start	first
44	_____ has been	ARC	ARCC	Simulated	JSS	Simulated