Indian Journal of Science and Technology

Supplementary Article

Introducing an Efficient Programming Paradigm for Object-Oriented Distributed Systems

R. Udayakumar[1*], A. Kumaravel[2] and K. Rangarajan[2]

[1]Associate Professor, Department of Information Technology, Bharath University, Chennai-600073; udayakumar@bharathuniv.ac.in

[2]School of Computing, Bharath University, Chennai-73, India; drkumaravel@gmail.com, kr221047@gmail.com

Abstract

Implementing a distributed process using object oriented programming is challenging especially for clusters of nodes ensuring the availability. Java happens to be well suited for writing object oriented programs for applications which needs modularity and high degree of cohesion [2]. In this paper our objective is to propose appropriate programming paradigms for loosely coupled distributed and object-oriented systems with a specific focus on parallel programming and client-server object computing. We select a prototype tool made based on the research outcome as found in [1] for this purpose. This research prototype tool 'Vishwa' provides a very good starting point for graduate students, engineers and scientists in academia, industry and government to present their tasks using the aspects of parallel and distributed computing. The learning curve is shorter in the domain of grid due to the simplicity in deploying and processing with this tool. We also demonstrate with a case study for bringing out the easiness in splitting the given job into subtasks and executed in this type of grid environment.

Keywords: Grid Computing, Distributed Computing, Object Oriented Programming, Speedup Ratio, Middleware.

1. Introduction

Over the last decade, grid computing technology has found its place in numerous scientific projects, has been adopted by numerous governments, and has been widely used for commercial projects. Furthermore, grid computing is particularly suitable for resource-demanding projects and enables scientists to work in a completely new way [3]. Grid computing is a type of data management and computer infrastructure, designed as a support primarily for scientific research. Moreover it is used in various commercial concepts, business research, entertainment and finally by governments of different countries. On its simplest level, the grid computing concept integrates four components: information, computation, networking and communication [2]. It has been made as a vision as a global networking infrastructure connecting multiple high performance computational resources. Typical applications [5] were implemented as Supercomputing across globe like ' Distributed Tera scale Facility' (DTF), Collaborative Computing ' as in GriPhyN (Grid Physics Network) and ' Global data repository and data intensive computing' realized as ' iVDgL' (international Virtual-Data grid Lab).

Device oriented or data intensive or computational intensive components are designed and developed at many different places. When these components are connected into the grid, the result is a virtual platform which allows an advanced data and computation management. If this concept is implemented into the areas mentioned above (science, etc.), it provides a platform where resources can be dynamically linked together, while these resources are then used to support the execution of applications that require significant amounts of computer resources. When we embarked upon the development of grid computing lab to make students learn these ideas, we need

*Corresponding author:
R. Udayakumar (udayakumar@bharathuniv.ac.in)

to consider the economy of scale. Every requirement for bigger grid must be reflected somehow in shorter scale covering important aspects for yielding the actual experience. We introduce and recommend the object oriented paradigm based on these needs. The very first challenge was to design the sequence of exercises by identifying appropriate problems which are highly parallelizable and to implement and verify distributed programming in a configuration of clusters based on LAN or WAN inside a laboratory.

2. Existing Grid Environments

More than one computer connected together to solve a problem referred as computer clustering. Then there are several ways of implementing the cluster. Grid computing is something similar to cluster computing, it makes use of several computers connected is some way, to solve a large problem.[8, 9] There is often some confusion about the difference between grid vs. cluster computing. The big difference is that a cluster is homogenous while grids are heterogeneous. The computers that are part of a grid can run different operating systems and have different hardware whereas the cluster computers all have the same hardware and OS. A grid can make use of spare computing power on a desktop computer while the machines in a cluster are dedicated to work as a single unit and nothing else. Grids are inherently distributed by its nature over a LAN, metropolitan or WAN. On the other hand, the computers in the cluster are normally contained in a single location or complex. Another difference lies in the way resources are handled. In case of Cluster, the whole system (all nodes) behaves like a single system view and resources are managed by centralized resource manager. In case of Grid, every node is autonomous i.e. it has its own resource manager and behaves like an independent entity [10]. In most of the cases core technologies are used in the available standard Grid environments. Moreover controlling and monitoring tasks are at the level of centralized Administration/centralized Management. The pattern of information flow is carried out in Client/Server Architecture. It does not scale across the internet and suffers from single point of failure. One of the main characteristics of a grid, namely the self organizing or autonomous behavior is missing. Another intension for solving scientific applications in the trusted environment is not feasible for graduate level lab experiments.

3. Motivation and Approach

We witness the emergence of a grid technology whose sophistication level has risen while the average user understands of that technology has not. There is a vast gap between theoretical or academic and the practical, as applied, knowledge based. Needless to say, it can be rather frustrating to fill this gap due to the following reasons. While Grid computing packages used in industry level usually take enormous human hours to install and establishing the security constraints, whereas in the academic environment cannot depend on such type of laborious exercises. Hence we consider a research prototype which satisfies the requirements of distributed computing and test its properties valid within the stipulated form of lab exercises. One can hope of that the design and develop with the research prototype we consider will spark under graduate students' interest in the young yet fast-evolving field of Grid Computing. In this paper we have attempted to present the steps for submitting a problem and splitting into subtasks for all the participating grid nodes to solve. We recommend the pre-requisite knowledge in order to set hands on grid computing as follows. Firstly the students should have some knowledge of the concepts and terminology associated to LAN, WAN connectivity and basic computer networking. Secondly the students should have some Java programming experience, in particular ability to read java classes and simple data structures and have basic knowledge in parallel programming logic in order to understand parallel tasks to be executed in more than one processor.

The preliminary grid model based on VISHWA, a research prototype for grid computing lab [13] was used to explore distributed problem solving across wide area clusters. This provides task splitting capabilities, and schedule tasks according to the Grid Computing Capacity Factor (GCCF) which can be evaluated by system parameters. It is the function of CPU speed, memory, network bandwidth.

4. Experimental Setup for Grid over a LAN

Most of the Universities, Research Labs and Industries have their LAN facilities for conducting the experiments. Hence we consider the same setup for our study purpose instead of really considering WAN, MAN, etc. Moreover the installation of tightly coupled distributed platforms are not taken

in to consideration as this may yield to more complex and time consuming for carrying out the experiments.

4.1 Middleware Components for Grid Laboratory

Selection of tools depends on space and other requirements in a constrained context like laboratory for graduate students. Peer-to-Peer Systems are suitable as they have the properties as follows: decentralized administration, decentralized management, scalable on the Internet and self organized behavior in the case of failures. A scalable reconfigurable P2P middleware for Grid Computing lab is recommended here. As we mentioned earlier off-the-shelf solutions cannot be used directly. Both Grid and P2P need to be merged taking the advantages of both to be captured. Moreover an unstructured P2P like Gnutella [11] where there is a random graph overlay and uses flooding or random walks to search for data is not suitable in our context. But structured P2P like Pastry [12] involves overlays formed based on node identifiers from the same space. The tool we consider takes both the capacities and serves our purpose treating structuredness and resource management in the exercises.

The following diagram in Figure 4.1 shows a typical configuration for an experimental set up at the grid laboratory. In the LAN connected environment a zonal server is created by running the java class 'zonal server. class' and grid nodes are added in the cluster one by one by running the java class 'vishwa. class'. The connectivity of all the nodes can be checked in zonal server as a special user interface through a routing table. The ip-address for each node is also shown in this small cluster of eight work stations inside a LAN. The ip-addresses need not be sequential and the nodes need not be same type. For the sake of simple implementation we run the experiments in a homogeneous collection of work stations.

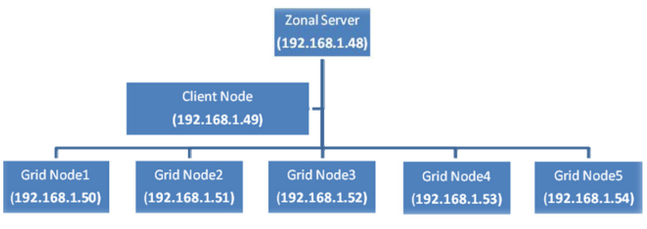

Figure 4.1 Typical configuration for an experimental set up.

The main components include Grid Initialization, the module for bootstrapping, the module for an user interface which acts as an interface between the Grid and the user, Problem Manager which splits and migrates the tasks, Subtask executor for executing the subtasks, Scheduler for resource discovery and resource optimization, Dispatcher for dispatching the task to the Subtask executor, Module for monitoring for resource monitoring and load monitoring, Route Manager for routing the requests to the available resources, Communication for making abstractions for connected problems, and the module for fault-tolerance to cope with the failures.

4.2 Main Procedure for the Experiment

Checking for the requirements: Java version 1.6 or higher is required. The main components of Vishwa are "Zonal server", "Grid node". "Zonal server" is a bootstrapping component. In order to join the grid, every node contacts the zonal server and gets the set of neighbor nodes and a node id. "Grid node" is the node which participates in the computation. Setting up the Grid by the tool Vishwa is a peer-to-peer Grid that is easy to set-up and use to run programs. Here we present a simple example to showcase the ease with which we can set-up the Vishwa Grid and run a sample program on the Grid in just 7 steps.

Step1: Download the JVishwa.jar, ZonalServer.jar and sample example program Prime.zip.

Step2: Prepare an application with business logic as say business.java and business Client.java.

Step3: Set up a Zonal Server by running ZonalServer.jar.

Step4: Start a Grid node by running JVishwa.jar. with ip address of zonal server started in step3.

Step5: Start 2 more Grid nodes.

Repeat the process give in Step 4 to add two more Grid nodes to the Vishwa Grid (use the same Zonal Server). Finally Check the Adjacency Table of the Zonal Server to confirm that the nodes are added.

Step6: Configure the sample program business.class using a Client node

Step7: Compile and Run the Program (both Step6 and Step7 to be implemented at Client node)

Our first "business" program will be to find the squares of the numbers from 1 to 10000. For execution in paral-

Table 4.1 Details of significant Components for Vishwa Grid

S.No	Name of the Component	Purpose of the Component
1.	Grid Initialization	When a node joins the grid, it initializes the selecting of closer neighbors from the Same Zone by Proximity. It contacts the closest available Zonal Server which generates node identifier and also returns a list of neighbors based on proximity along with the node identifier. Grid node chooses nodes from neighbor list based on network delay. Zonal Server also returns a list of proximity closer zonal servers to the grid node Grid node gets some neighbors from other zones. This is to ensure that whenever the resources within zone are exhausted we can expand our computation to other zones. In this stage node also initialize it's routing state required for the reconfiguration layer, node routing state consists of leaf set and routing Table which helps in routing within the zone also to maintain the replicas. The Routing table entries consist of nodes within the zone and from other zones and helps in routing within the zone, routing across the zones.
2	User Interface	Responsible for interacting with the end user and the Grid
3	Problem Manager	Requests the scheduler for the required no. of donors and splits the problem into chunks and packs the data and the tasks and sends it the dispatcher for dispatching to appropriate donors. Stores information about tasks such as task id, submitting node, donors, and subtask information in the reconfiguration layer
4	Subtask Executor	Responsible for executing the given subtask and stores information about the Subtask in the reconfiguration layer.
5	Scheduler	Each Node has the scheduling component and autonomically makes scheduling decisions, with only partial knowledge of the grid Each node maintain neighbor list within K-hops to which tasks can be migrated and dynamically updated with the HPF propagation algorithm Scheduler looks at the neighbor list to select candidate nodes base on HPF.
6	Monitoring	Keeps track of resource dynamics of the grid .Each node runs Monitoring component for making fully decentralized. Monitors the Load Condition of its own. Keeps track of the related grid nodes using the heart beat algorithm.
7	Route Manager	Routes the requests to the available resources.
8	Communication Component	Provides inter task communication among the subtasks by using the distributed pipe abstraction.
9	Fault-tolerant Component	Gets information of failure of neighbor list nodes or leaf set nodes or application level neighbors from the monitoring component

lel in the grid, we need to split the task into sub-problems and execute them separately in different grid nodes. In this example, we split the task into 10 sub-tasks. Each sub-task involves computing the squares of 1000 consecutive numbers. Thus, one subtask will compute the square of numbers from 1 to 1000, another subtask that from 1001 to 2000, etc.

We need to specify the operations which need to be performed by each subtask. In addition, we need to supply the parameters required for each subtask and also collect the result of each subtask. In Vishwa, the parameter supplying, result collection etc. are specified in a client program. The operation to be performed by each subtask is specified in a subtask program.

In our example, the client program specifies the range of numbers for which square has to be calculated by providing the minimum and maximum value for each subtask (e.g. min = 1, max = 10). The subtask program gets the specified range and computes the squares of numbers within the given range in a grid node and returns the computed values to the client program. Then, the client program combines the results from all the subtasks and stores it in a separate file.

5. Client Application Example

In our example, we will use business.java as the subtask program (source file) and businessClient.java as the

client program (source file). The final result will be stored in businessResult.txt. First, we create a directory "business" for holding the client program for task execution and open the client source file "businessClient.java" in it.In businessClient.java, first we import the required packages as in Figure 5.1:

```
import jvishwa.client.*;
import jvishwa.Result;
import jvishwa.Context;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import jvishwa.file.*;
import java.io.InputStreamReader;
```

Figure 5.1 Required packages.

We create a class businessClient and the main execution method within it.

Within the main method, get the instance of the Client Manager as in Figure 5.2 which acts as a mediator between user program and Vishwa grid middleware.

```
public class businessClient {

public static void main(String args[]) {
ClientManager manager = manager.setMinDonors(1);  …
```

Figure 5.2 Instantiation of client manager.

We set the minimum and maximum number of grid nodes required. Also, we set the IP address of the grid node to which the task is to be submitted initially, and the number of surplus nodesto be used to replace failed nodes.

```
Initialize the client manager.
try { manager.initilize();} catch (Exception e) {
e.printStackTrace();
System.exit(0);}
```

Figure 5.3 Initialization of client manager.

Create the subtask handles and context for each subtask. Sub Task Handle is the Id given to the remote subtask by client manager. Context is used for passing parameters to each subtask as a key-value pair. manager.execute() is called to execute each subtask whose operations are specified in the class business.

```
SubTaskHandle handleSet[] = new SubTaskHandle[10];
Context context = null;
//Creating subtasks
for (int i = 0; i &lt;= 9 ; i++) {
context = new Context();
context.put("fromval", i + "1");
context.put("toval", i + 1 + "0");
//pass parameter and get id of remote subtask
handleSet[i] = manager.execute(new business(), context);
}
```

Figure 5.4 Execution of subtasks.

Now, we wait for all subtasks to finish and and close the manager. The lines of code as in Figures 5.3, 5.4 and 5.5.

```
manager.barrier();
```

```
manager.close();
```

Figure 5.5 Blocking & Finishing the subtasks.

We create the file to which final results are to be written as in Figure 5.6. We collect the result of each subtask using the Result object. Each result is read from a file associated with the subtask Resultobject. The obtained results are written into the output file.

```
try {
FileOutputStream fout = new FileOutputStream("businessResult.txt");
PrintStream ps = new PrintStream(fout);
for (int i = 1; i &lt; = 10; i++) {
//get results from each subtask
Result r = handleSet[i - 1].getResult();
VishwaFile vfile = r.getFile();
BufferedReader input = new BufferedReader(new
InputStreamReader(new FileInputStream(vfile)));
String content = null;
System.out.println("Compilling results from Sub-Task:" + (i - 1));
ps.println("Compilling results from Sub-Task:" + String.valueOf(i - 1));
//placing the results into the text file....
while ((content = input.readLine()) != null) {
ps.println(content);
}
input.close();
}
ps.close();
fout.close();
} catch (Exception e) {
e.printStackTrace();
}
```

Figure 5.6 File creation for final results.

Create the file BusinessLogic.java in the same directory. Here we first import all the necessary packages which contains input, output handling coding and splitting of input job appropriately. The imported packages are shown in Figure 5.7

```
import jvishwa.VishwaSubTask;
import jvishwa.Result;
import jvishwa.Context;
import jvishwa.file.*;
import java.io.PrintStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
```

Figure 5.7 Required packages for IO files.

Create the class Square which inherits the VishwaSubtask class. Create a VishwaFile object to hold the results in Figure 5.8. We need to override the run(), getObject() and callback() method of VishwaSubtask class.

```
public class BusinessLogic extends VishwaSubTask {
VishwaFile vfile = null;
```

Figure 5.8 Instanciation of a Vishwa File.

In BusinessLogic class, override the run() method of VishwaSubTask to specify the grid operation to be performed. run() takes the input parameters from the Context object passed. In our example Figure 5.9, the minimum and maximum of the range of numbers are obtained. The square of each number within the range is calculated. The results are written to "result" file using a FileService object.

```
public void run(Context c) {
//Accessing the parameters from the context
int startpoint = Integer.parseInt((String) c.get("fromval"));
int endpoint = Integer.parseInt((String) c.get("toval"));
//File Service for creating a file
FileService fileService = new FileService(this);
vfile = fileService.createFile("result");
PrintStream ps = null;
try {
ps = new PrintStream(new FileOutputStream(vfile));
} catch (Exception e) {
e.printStackTrace();
}
//Calculation of squares for our sample business
for (int i = startpoint; i &lt;= endpoint; i++) {
ps.println(squareNum(i) + "");
}
ps.close();
}
public int squareNum(int num) {
return num* num;
}
```

Figure 5.9 Creating typical IO files and Business Logic.

Next, we write the deserialization method for the subtask object, getObject() (overrides getObject() in VishwaSubTask class). Finally, we write the callback() method to collect the results(overrides callback()) of VishwaSubTask class as shown in Figure 5.10.

```
//Deserialization method for getting the object
public Object getObject(String fileName) {
try {
FileInputStream fin = new FileInputStream(fileName);
ObjectInputStream ois = new ObjectInputStream(fin);
Object o = ois.readObject();
ois.close();
return o;
} catch (Exception e) {
e.printStackTrace();
}
return null;
}
//Aggregating the results...
public Result callback() {
Result r = new Result();
r.putFile(vfile);
return r;
}
```

Figure 5.10 Aggregating the results from subtasks.

So far we have prepared our client and we are ready to submit through the client node.

While we compile the businessClient.java and run we get the output as in Figure 5.11:

```
Min nodes...3 max nodes... 5
Starting execution...
Uploading the file through FTP
Uploading the file through FTP
```

Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps9result
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps10result
Uploading the file through FTP
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps1result
13.Downloaded Successfully result file: /home/ kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh 6wHSpuOTQps8result
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps5result
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps4result
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps3result
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps2result
Uploading the file through FTP
Downloaded Successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh6wHSpuOTQps7result
Uploading the file through FTP
Downloaded successfully result file: /home/kumaravel/vishwa/vishwajar/business/Vishwa_temp/0000021LwgceroQVA2Lwh 6wHSpuOTQps6result
Compiling results from Sub-Task: 0
Compiling results from Sub-Task: 1
Compiling results from Sub-Task: 2
Compiling results from Sub-Task: 3
Compiling results from Sub-Task: 4
Compiling results from Sub-Task: 5
Compiling results from Sub-Task: 6
Compiling results from Sub-Task: 7
Compiling results from Sub-Task: 8
Compiling results from Sub-Task: 9

Figure 5.11 Generation of results as in Zonal Server Display.

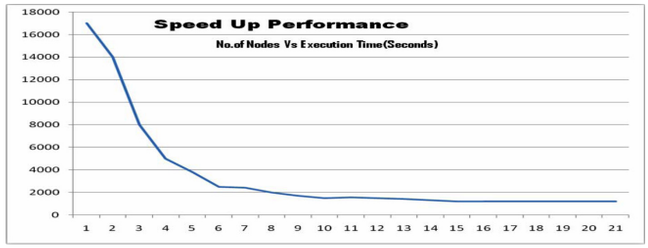

Figure 5.12 Performance of a small grid.

Hence, we have shown an example of small grid job above with the components detailed in Table 4.1. The study on this environment was carried out and the results are shown in Figure 5.12.

6. Remarks and Conclusion

The first disadvantage concerns the relative immaturity of the concept. This is especially noticeable when talking about the missing features, non-defined standards and software. More specifically, the results of all processes are sent first on all grid nodes within the cluster, and then collaboratively assessed. Before the final assessment is made, it is not possible to define or to declare a final outcome. [6, 7] This is particularly a problem when talking about time sensitive projects. Hence the students having the experience of research tools of above nature will easily overcome this problem as they get experience on a LAN with local zonal servers and grid nodes running Vishwa with flat file storage of proprietary operating systems and java virtual machines. P2P Security mechanisms need to be explored. Currently there is no support for data management. We currently work with our perspective from the local computational grid with trusted environment and Internet has not yet made as trusted yet.

7. Acknowledgement

The authors would like to thank the management of Bharath University for their support and encouragement for doing this research studies and also Dr. D. Janki Ram, Senior Professor, Department of Computer Science and Engineering, IIT Madras, for providing the research prototype and leading the distributed object computing experience in this environment.

8. References

1. Janakiram D (2010). Grid Computing, Research Monograph Tata Mcgraw-Hill Publishers.
2. Srinivas V, and Janakiram D (2009). Node capability aware replica management for peer-to-peer grids, IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, vol 39, No. 4, 807–818.
3. Berman F, Fox G et al. (2003). Grid computing: making the global infrastructure a reality, 1st Edn., Wiley Series.
4. Joseph, Joshy, and Fellenstein Craig (2003). Grid Computing, Prentice Hall PTR.
5. Joseph J, Ernest M et al. (2004). Evolution of grid computing architecture and grid adoption models, IBM Systems Journal, vol 43(4), 624–645.
6. Baker M, Apon A et al. (2005). Emerging Grid Standards, IEEE Computer, vol 38(4), 43–50.
7. Eddy Carona et al. (2007). Definition, modeling and simulation of a grid computing scheduling system for high throughput computing, Future Generation Computer Systems, vol 23(8), 968–976.
8. Parashar M, and Browne J C (2005). Conceptual and Implementation Models for the Grid, Proceedings of the IEEE, vol 93(3), 653–668.
9. Atiknson M, Chervenal A et al. (2004). Data Access, Integration and Management, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kauffmann.
10. Chawathe Y, Ratnasamy S et al. (2003). Making Gnutella-like P2P Systems Scalable, SIGCOMM'03, Available From: http://dcg.ethz.ch/lectures/ws0506/seminar/papers/scalable_p2p.pdf
11. Rowstron A, and Druschel P (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 329–350.