MULTIMEDIA ON GLOBAL GRIDS: A CASE STUDY IN DISTRIBUTED RAY TRACING

Bader Aljaber, Thomas Jacobs, Krishna Nadiminti, Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
Email: {baljaber, tpjacobs, kna, raj}@csse.unimelb.edu.au

ABSTRACT

Grid computing, a form of distributed computing, has become a hot topic recently. Currently, there are many ways in which Grids are being used in order to enhance existing IT infrastructure to better utilize distributed resources and data. Grid technology is considered to be a very powerful solution to the problem of sharing heterogeneous resources including compute resources, data stores, and different kinds of services provided by various entities. In this paper, we look at applications that make, or could make, good use of a distributed computing environment, such as a Grid. We provide some background on a selected application, the POV-Ray ray tracer, and investigate how it can benefit from running on a Grid. We use the Gridbus Resource Broker for scheduling application tasks on a Global Grid.

Keywords:* *Grid Computing, Multimedia, Gridbus Resource Broker, Ray Tracing

1.0 INTRODUCTION

Distributed computing harnesses the power of many computers, or clusters of computers, to provide a virtual supercomputer. Applications that are otherwise infeasible to run on individual computers today can be run by distributing the workload over a large network of computers within an organisation or across multiple organisations.

Grid computing, a form of distributed computing, has become a hot topic recently. Distributed computing is the process of aggregating the power of several computing entities to collaboratively run one or more tasks in a transparent and coherent way.

Grid computing is an innovative approach that enhances existing IT infrastructure to aggregate resources and manage data and compute-intensive application workloads. It is the process of utilizing various resources, such as data and computational power of distributed entities to cooperatively, concurrently and transparently execute a single and huge task, so that they appear as a single system. Buyya defines a grid [1] as:

> "a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements."

Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, personal computers, storage systems, data sources, and specialised devices that are geographically distributed and generally owned by different organizations. Grid-based systems can be used to solve large-scale computational and data intensive problems in science, engineering, and commerce [2].

A number of factors have led to the development of Grid systems. The main factors are the availability of low-cost, high-speed personal computers, and specifically, those that are being used as workstations. These workstations are often idle, resulting in available processor power being wasted.

Grid technology grew out of the needs of high-performance computing (HPC) applications from the scientific and academic communities and has since matured to be gradually adopted by enterprises and businesses.

A great many of current multimedia applications need huge compute power. For example, applications like image-rendering, using ray-tracing techniques take a long time to produce high quality results and need more than one high powered resource to complete the job. This paper looks closely at one such application, which was originally not designed to be distributed, and describes how to make it function on a distributed computing platform. The original application, called POV-Ray, uses ray-tracing to render very high-quality 3D images.

2.0 MULTIMEDIA APPLICATIONS AND HIGH-PERFORMANCE COMPUTING (HPC)

Multimedia applications, a combination of interactive text, graphics, video, animation and sound, have increasingly become an important part of people’s lives today. They are widely used in many activities such as entertainment, education, business and communication. Table 1 shows some of these multimedia applications. Multimedia applications can be shared between people situated across the street, in a neighbouring state or even in another country [3], via a distributed system. This introduces the term *distributed multimedia*.

When it comes to implementing distributed multimedia, many concerns have been brought about such as price, performance, efficiency and plumbing. Our focus in this paper is what is commonly desired - to obtain the highest performance with the lowest price possible. In distributed multimedia applications, huge amounts of stored media files must be managed and accessed efficiently, and a large number of complex, compute-intensive calculations are involved. As a consequence, normal computers such as personal computers (PCs) would be insufficient because processing on such systems takes a very long time. Also since some multimedia operations can occur concurrently, distributed multimedia applications could be executed in parallel for better efficiency and quality. This raises the need for a technique that can manage resources with a capability to run operations in parallel [4].

Table 1: List of Some Multimedia Applications.

Multimedia Application Class	**Examples**
3D Ray-tracing	POV-ray[32], MPI-POV[21], PVM-POV[22]
Movie	XviD[24], Shrek2[23]
Games	AgentSheets[6], Dazzler[19], CyberEditor[20]
Paint program	UberPaint 4p, Deluxe Paint 4 [30]
Picture viewer	MysticView[26]
Sound and Music	Roland TB 303[27]

Computer researchers and analysts have investigated various techniques to solve issues such as load balancing, increasing the speed of processing and getting high performance with low cost. One of the most effective ways found is by using High Performance Computing (HPC) resources such as Clusters. Another approach is aggregating the power of distributed HPC systems to create computational Grids. In other words, an advanced and complex multimedia application can work better with the usage of high performance computing and communication technologies for the processing, storage, and real-time delivery of data.

2.1 HPC in the Movies

The movie ‘Shrek2’ (see Table 1), released in May 2001, is a computer-animated motion picture that renders moving images by using high performance computers. This application came to life with help from technology developed by HP Labs for providing high quality rendering capability. Using HP’s Remote 3D software and DreamWorks' high-resolution imaging technology, Virtual Studio Collaboration (VSC) linked all the creative elements together, enabling real-time communication and the efficient exchange of storyboarding, content review, editing, creative consulting and other vital production information [5].

2.2 HPC in the Graphical World

Tasks like creating class room simulations, for example, are more manageable when people are working together, exchanging thoughts and assisting each other with the development. AgentSheets (see Table 1) is a tool for building interactive, graphical systems that can enable such scenarios and more. A free demo version can be downloaded from [6]. The AgentSheets application allows a group of people to collaborate on a project resulting in better and more comprehensive results than managing the project in other ways. For example, creating a web site and building a simulation of implementing a complex project allowed a group of students to distribute the workload among themselves according to their individual interests [7].

Such scenarios need the tools to support collaboration and communication between the group members contributing to different tasks in the project. Grids are an effective and reliable way to implement application scenarios which involve distributed groups of people collaborating remotely. Distributed computing utilizes multiprocessors to perform concurrent jobs in parallel. These techniques can significantly help reduce costs, increase efficiency and improve overall price/performance of complex applications using networks.

2.3 Grid Technologies for Media Applications

HPC and Grids can significantly help because a single multimedia application would be run by multiple powerful resources concurrently and transparently. Moreover, since distributed multimedia applications may require a variety of resources that are not available in a single organisation, heterogeneity and availability, security, and fault tolerance and other issues arise. Grid technology takes care of all these issues by using what is called “middleware”[9] [15]. The architecture of a typical Grid is shown in Fig. 1.

The Grid Fabric consists of distributed resources such as computers, networks, storage devices and scientific instruments. The distributed resources could be logical or physical clusters, supercomputers, servers and ordinary PCs which run a variety of operating systems (such as UNIX variants or Windows) [8].

The Core Grid middleware sits on top of this fabric and provides services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and some aspects of Quality of Service (QoS) such as resource reservation and trading. So, the complexity and heterogeneity of distributed resources can be hidden by these services by providing a uniform and consistent access to them. A popular Grid middleware which supports many of these core services is Globus [9] developed by researchers from the Argonne National Laboratory and University of Southern California, USA. Alchemi [10] is another middleware, which is predominately focuses on supporting enterprise Grids and it offers Web services-based remote job management services.

User-level Grid middleware provides abstractions to the services provided by the low-level (core) middleware. These include application development environments, programming tools and resource brokers. The most important component in this layer is the broker which is used to submit, manage, schedule and gather jobs across grid resources. A popular user-level Grid middleware providing utility-oriented grid resource management and scheduling services is Grid Resource Broker [11] developed by researchers from the University of Melbourne, Australia.

Several applications can be constructed on top of grid middleware, developed using Grid-enabled languages and utilities such as HPC++ or MPI. An example is multimedia applications which need huge computational power, and access to remote data sets.

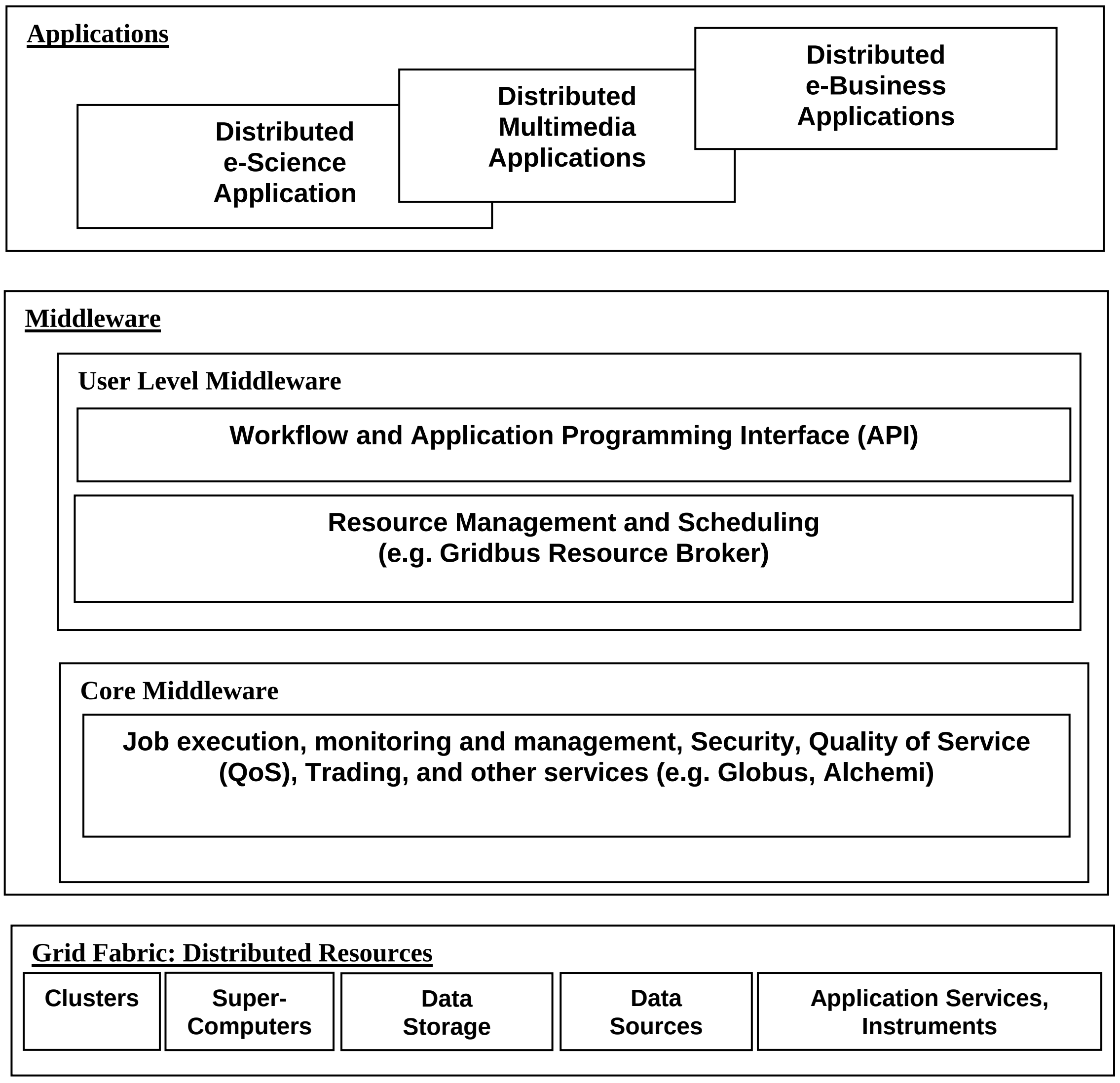

Applications
Distributed e-Science Application
Distributed Multimedia Applications
Distributed e-Business Applications
Middleware
User Level Middleware
Workflow and Application Programming Interface (API)
Resource Management and Scheduling (e.g. Gridbus Resource Broker)
Core Middleware
Job execution, monitoring and management, Security, Quality of Service (QoS), Trading, and other services (e.g. Globus, Alchemi)
Grid Fabric: Distributed Resources
Clusters
Super-Computers
Data Storage
Data Sources
Application Services, Instruments

Fig. 1: The Architecture of Grid.

3.0 PERSISTENCE OF VISION RAY-TRACING (POV-RAY) AND HPC

POV-Ray is short for Persistence of Vision Ray tracing, a tool for producing high-quality computer graphics. The POV-Ray program creates three-dimensional, photo-realistic images using a rendering technique called ray-tracing. It reads in a text file containing information describing the objects and lighting in a scene and generates an image of that scene from the view point of a camera also described in the text file [12]. Ray-tracing is a very compute-intensive method for generating 3D images, but it produces very high quality images (see Fig. 2) with realistic reflections, shading, perspective and other effects.

Ray-tracing generates images by tracing one beam of light per pixel from the “camera” into the scene, through the virtual display screen, and intersecting it with objects in the scene (see Fig. 3). At each intersection, we follow secondary rays, generated by reflection, transmission, and shadows, which in turn intersect with other objects, generating more rays, and so on, to generate a ray-tree, which usually has about 10 levels of depth. When the ray-tree is completed, we determine the intensity and colour of each pixel by adding up the components from the base of the ray-tree.

This iterative or recursive 'following reflections' approach requires a lot of intersections to be calculated per pixel. POV-Ray in its normal form uses only one computer to render the image. Because of this, some images may take many hours or even days to fully render on up-to-date PCs. Animation, as a natural extension, of course can take an intractable amount of time [13].

L'oiseau mouillé - The wet bird - Gilles Tran © 2000 www.oyonale.com

Fig. 2: A scene rendered by POV-Ray [32].

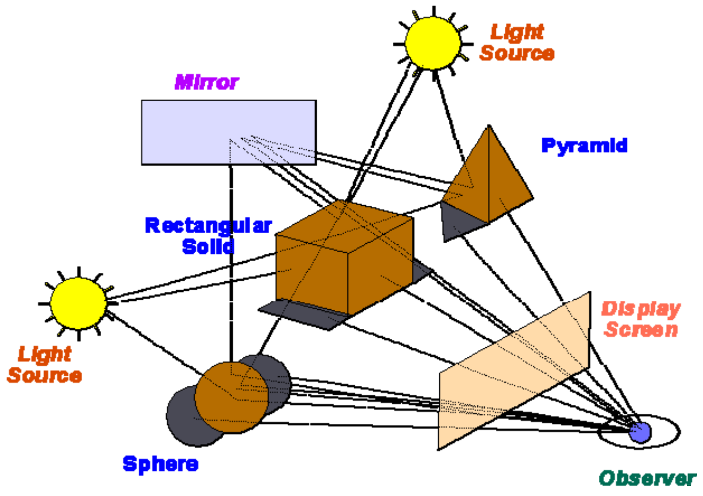


Fig. 3: How ray-tracing works [12].

3.1 Clusters and Ray-tracing

Photo-realistic image synthesis is widely used in many applications such as architectural design, industrial design, visual simulation for landscape design, and art and so on. However, producing photo-realistic images with high quality is very time-consuming.

Researchers and scientists have attempted to use parallel computers and clusters to speed up the rendering of photo-realistic images. When using cluster computing, the image is split into a number of frames or slices either by rows or columns and each computer renders a subset of the total number of frames. The frames can be sent to each computer in contiguous chunks or in an interleaved order. In either case a preview (every nth frame) of the animation can generally be viewed as the frames are being computed.

Basic clustering, though useful, might not be sufficient to render a very complex image or animation in a reasonably short time. One way to achieve a more efficient rendering process is using more than one cluster: that is, applying Grid computing technology.

3.1.1 MPI-POV / PVM-POV

There are two distributed versions of the ray-tracing application that have been developed, based on POV-Ray. They are called MPI-POV and PVM-POV (a Parallel Implementation of POV-Ray based on MPI (Message Passing Interface) and PVM (Parallel Virtual Machine) respectively). These distributed versions are both patches on the original POV-Ray source code. With these versions, we are able to enlist the power of several computers to render one image or animation, thus making the rendering more tractable, and feasible.

PVM-POV works by having one master and many slave tasks. The master and workers are in a single cluster system working concurrently. The master has the responsibility of dividing the image up into small blocks, which are assigned to the slaves. When the slaves have finished rendering the blocks, they are sent back to the master, which combines them to form the final image. The master does not render anything by itself, although there is also, usually, slave running on the same machine as the master, since the master task itself does not use much CPU power [16].

MPI-POV works much the same way as PVM-POV, and is actually a patch on PVM-POV (which itself is a patch on POV-Ray). The main difference is that it runs using MPI as opposed to PVM.

3.2 Weakness of current systems

MPI-POV and PVM-POV are aimed at being run on clusters, and so do not take into account the heterogeneity of resources; for example, computers with different computational power, different network link speed and other characteristics. They are not able to adapt to changes in network conditions, node failures, overloading or other dynamic conditions that prevail in the computing environment, instead they just assume homogeneity in compute power and other attributes in their algorithms of work distribution, which will result in overloaded or under utilised nodes when executed on a heterogeneous environment. This makes them (MPI / PVM) unsuitable for use on a Grid, which consists of heterogeneous resources, leaving them unable to scale well. The work presented in this paper aims to allow POV-Ray to balance the workload evenly over non-uniform computational resources. The work described in [17] looks at this very problem and describes various methods for load-balancing.

Another weakness is that they only run over PVM or MPI, limiting the possible deployment of the systems. POV-Ray extended in this work, distributes the rendering load using the Gridbus Broker system to enhance its execution speed by running on global Grids.

4.0 POV-RAY ON GLOBAL GRIDS

The distributed POV-Ray rendering application is developed in Java, and uses the Gridbus Grid Resource Broker [11] Application Programming Interface (API) [18]. The broker allows transparent and seamless access to heterogeneous resources, providing services such as resource discovery, secure access, scheduling, monitoring, and job submission [18]. It supports both computational and data grid applications. Fig. 4 shows a common way which the broker can be invoked from within a Java program.

As described in [18]: a resource on a grid could be any entity that provides access to a service. This could range from compute servers to databases, scientific instruments, applications and the like. In a heterogeneous environment like a Grid, resources are generally owned by different people, communities or organizations with varied administration policies, and goals. Naturally, obtaining and managing access to these resources is not a simple task. Resource brokers aim to simplify this process by providing an abstraction layer to users who just want to get their work done. In the field of Grids and distributed systems, resource brokers are software components that let users access heterogeneous resources transparently, without having to worry about availability, access methods, security issues and other policies. The Gridbus resource broker is designed with the aim of solving these issues in a simple way.

The broker takes care of job submission and scheduling. It dispatches tasks to available resources and collects the results back. It isolates the complexity of underlying system architecture of the grid and provides a uniform access method to different core middleware grid architectures. The rendering program we implemented creates tasks in the Gridbus broker by specifying the start row and end row for each slice of the image. The broker executes the tasks by sending them to its dispatcher module, which is responsible for submitting and monitoring the tasks on the resources. The broker then polls the resources periodically, and raises events when the status of tasks changes.

The program updates the display accordingly when it receives these events. Once all the slices of one image have been collected after rendering, the Java program displays all these slides as one image. It then generates the final PNG file from the slices, and writes it to the working directory of the program. When the image is being rendered, the slices are sent to the resources in order, but will be returned back from the resources possibly out-of-order, depending on the various speeds and load levels of the resources.

In terms of speedup and performance, if all slices take about the same time to be completely processed then dividing the images evenly either by rows or columns would be the optimal way to distribute the load amongst the nodes. Unfortunately this is not always the case; the rows across some parts of the image, for instance the background, might take a trivial length of time to render while other rows that intersect the object's part of the scene might take a lot longer. As a result, the machines that rendered the fast portions will stand idle for most of the time or the whole process of rendering will take much longer because the most complicated parts on the image are being rendered by a few nodes while the others are idle. This case violates the aim of rendering images through Grid system. Therefore, this can be overcome by using load balancing techniques.

One way around this, suggested by [14], is to split the parts that would take longer into many more row chunks than there are computers and create a program that submits jobs to machines as they become free. Another way is to create row chunks of different heights according to the estimated rendering time. The script used to render the final image can be modified to render a much smaller image with the same number of row chunks. An estimate of the time for the different rows can be used to create narrow row chunks in the complex regions and wide row chunks in the fast rendering regions.

Another possible way is to divide the image into chunks and keep node busy by sending chunk to any node if that node has finished its current work and has become free to receive other work. In other words, assuming that all jobs, chunks, are in a pool and nodes should ask for new job once they finished their current jobs. Consequently, there would be better node utilisation.

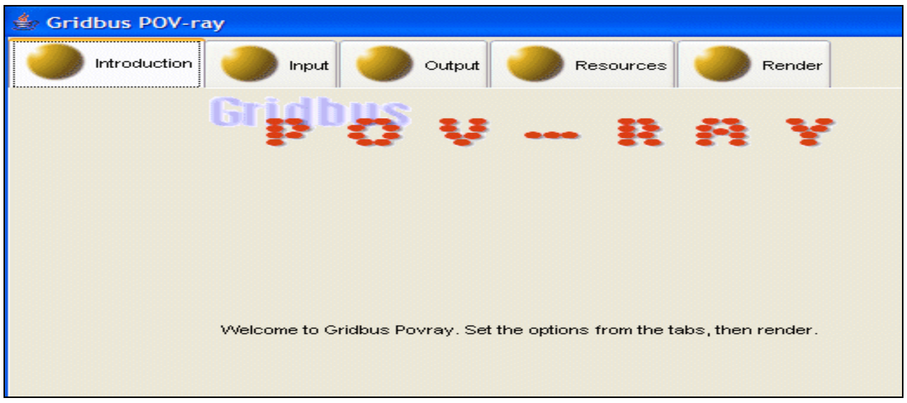
```
try{
        //Create a new "Farming Engine"
        GridbusFarmingEngine fe=new GridbusFarmingEngine();

        //Set the Application-description file
        fe.setAppDescriptionFile("calc.xml");

        //Set the Resource-description file
        fe.setResourceDescriptionFile("resources.xml");

        //Call the initialise method to setup jobs and servers
        fe.init();

        //Start scheduling
        fe.schedule();

        * The schedule method returns immediately after starting the
        * scheduling. To wait for results / monitor jobs,
        * use the following loop:

     while (!fe.isSchedulingComplete());
}catch (Exception e){
  e.printStackTrace();
}
// To use the APIs to create jobs:

  Job currentJob = new Job();
  currentJob.setJobID("j1");

  //Create commands
  //Command to Copy the program
  CopyCommand copy = new CopyCommand();
  copy.setSource(true,"workingDIR/calc",true);
  copy.setDestination(false,"calc",true);

  //Command to execute the program
  ExecuteCommand exec = new ExecuteCommand(TaskCommand.EXECUTE_CMD);
  exec.setExecutable("./calc");
  exec.addArgument("$X");
  exec.addArgument("$time_base_value");

  //Command to collect the results
  CopyCommand results = new CopyCommand();
  results.setSource(false,"output",true);
  results.setDestination(true,"output."+currentJob.getJobID(),true);
  Task task = new Task();
  task.addCommand(copy);
  task.addCommand(exec);
  task.addCommand(results);
  currentJob.setTask(task);
  currentJob.addVariable(new SingleVariable("$X", "1"));
  currentJob.addVariable(new SingleVariable("$time_base_value", "0"));

  fe.addJob(currentJob);
```

Fig. 4: Creating a new "Farming Engine", more information can be found in [18].

4.1 The Gridbus POV-Ray Distributed Rendering Application - Screenshots

The GUI for the distributed rendering application is organised into five tabs (Fig. 5). Interaction with the program is briefly described with snapshots of some of the screens.

4.1.1 Opening screen (see Fig. 5)



Welcome to Gridbus Povray. Set the options from the tabs, then render.

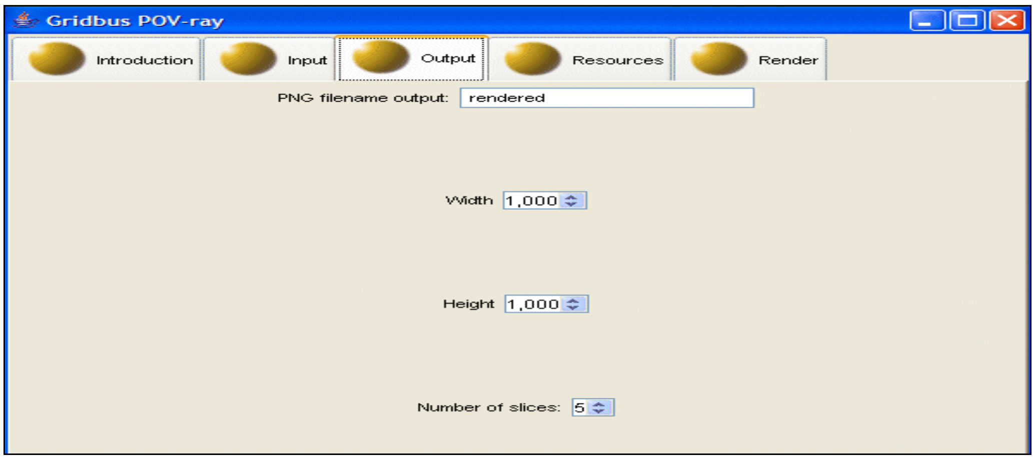
Fig. 5: The first tab of the project GUI.

4.1.2 Input options

The user can choose the image that needs to be rendered. The “fish.pov” file, for example, is a good choice for a demonstration, as it is quite a complex, detailed image and takes a while to render. All the files in the directory containing the ‘.pov’ file are copied to each grid node, including ‘.inc’ files and the ‘.pov’ file. The user must make sure that all the files required for rendering a particular image, such as texture files and ‘.inc’ files are included within the same directory. It is also a good practice to make sure that no other large files are in that directory, as these will be unnecessarily transferred to each node.

4.1.3 Output options

This allows the user to specify the size of the image to render as well as the name of final result image (see Fig. 6). The final image is placed in the working directory of the program. The user can also specify the number of slices for the program to divide the image.



PNG filename output: rendered

Width 1,000

Height 1,000

Number of slices: 5

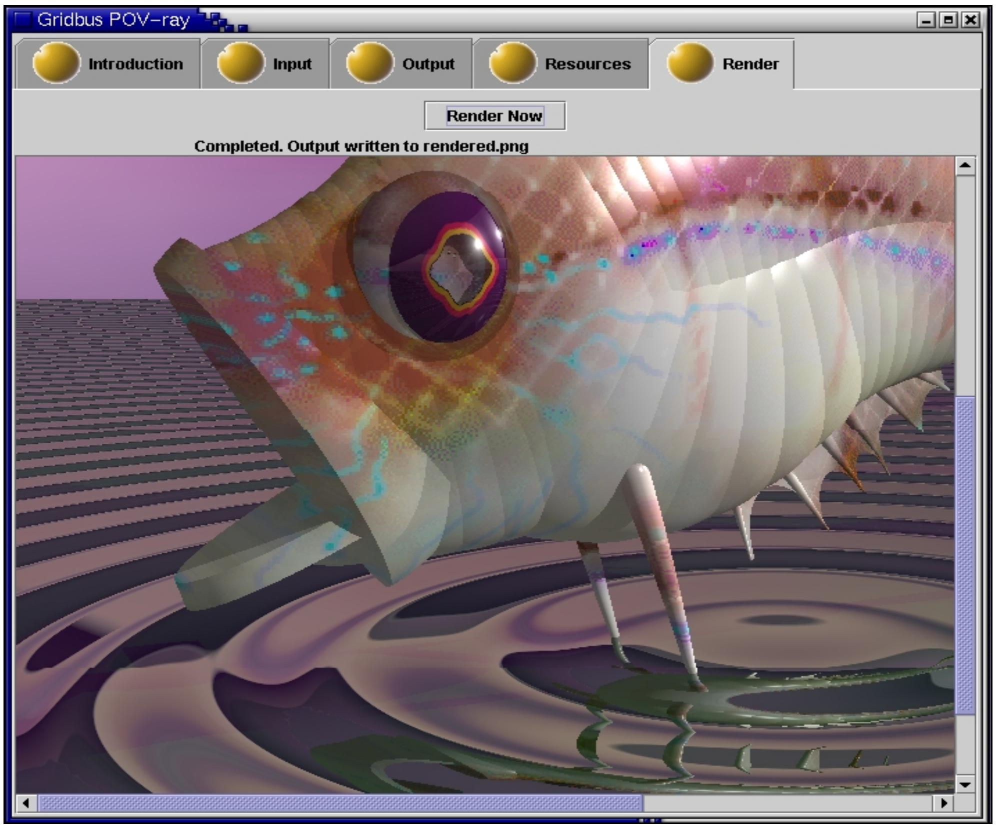
Fig. 6: The third tab of the project GUI.

4.1.4 Selection of Grid resources

This tab allows the user to choose the distributed resources from the pre-defined list, or to specify a XML resource listing file, which contains a list of Grid resources to use in the rendering.

4.1.5 Rendering the Image

When all options have been set, the user can initiate the rendering. This brings up the status of all the slices of the image. Each one starts up, then renders, and when finished rendering that slice of the image is displayed in place. The order of the slices rendered will vary, depending on the load on each resource. Once all of the slices have been rendered, they will form the final image on-screen, and the final '.png' file will be generated and written to the working directory of the program. When all nodes finish their jobs, complete output is displayed (Fig. 7).



Completed. Output written to rendered.png

Fig. 7: The final look of a sample rendered image.

4.2 Results and Performance

The program delivers a speedup factor nearly equal to the number of resources that the execution is divided over, because it is an embarrassingly parallel application. Some overhead is incurred, because the input files need to be copied to each grid node and this takes time, especially if the image is complex and involves many textures.

A complex image of a lighthouse was used for testing the performance. This was rendered over the three grid resources available to the study. The three grid resources were manjra.cs.mu.oz.au, belle.cs.mu.oz.au, and belle.anu.edu.au. manjra.cs.mu.oz.au is a 13-node x86-based Linux cluster located at the Department of Computer Science, The University of Melbourne. belle.cs.mu.oz.au is an SMP with four 2.6 GHz Intel Xeon processors with 2 GB RAM, 70 GB HD running Linux located at the same place. belle.anu.edu.au has a configuration similar to bell.cs.mu.oz.au and is located at the Australian National University, in Canberra.

It takes about 5 minutes to render 4 out of 100 slices of the image, which means that it would take about 2 hours to render the complete image. In contrast, the image takes approximately 9 days to render on a single Semptron 2600+ desktop PC.

5.0 SUMMARY AND CONCLUSION

In this paper we have looked at a number of distributed applications. We have investigated their differences, similarities, weaknesses, and how these applications work to distribute the load among the many workers that process sections of the data to be analysed.

We have presented an overview of the Grid approach to distributed multimedia applications, and how Grid environments can play a great role in not only speeding up computational processing by implementing methods for balancing the load over nodes evenly, but also enable on-demand sharing of heterogeneous distributed resources. As we have seen in this paper, the original, non-distributed application takes a long time to execute on a single desktop PC, however, it runs efficiently, scales well, and takes a much shorter time when it is implemented using Grid technology.

Finally, all these distributed applications and others have a significant impact on the development of science in different areas. Today, e-Science and e-Business are increasingly done through distributed global collaborations and research enabled by the Internet. In the future, perhaps there will be a Grid connecting not just the people and resources on the Earth but also those on other planets.

REFERENCES

[1] R. Buyya, Grid Computing Info Center, http://www.gridcomputing.com/gridfaq.html, 2002.

[2] M. Baker, R. Buyya, and D. Laforenza, "Grids and Grid Technologies for Wide-Area Distributed *Computing"*, *International Journal of Software: Practice and Experience (SPE)*, Vol. 32, Issue 15, December 2002, pp. 1437-1466, Wiley Press, USA.

[3] http://www.bridgeco.net (accessed in June, 2006)

[4] http://www.dbpd.com/vault/9705davd.htm (accessed in May, 2006)

[5] http://www.gridtoday.com/04/0503/103139.html (accessed in June, 2006)

[6] http://agentsheets.com/ (accessed in June, 2006)

[7] http://jasss.soc.surrey.ac.uk/3/3/forum/1.html (accessed in May, 2006)

[8] P. Asadzadeh, R. Buyya, C. Ling Kei, D. Nayar, and S. Venugopal, "Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies", *High Performance Computing: Paradigm and Infrastructure*, Laurence Yang and Minyi Guo (editors), Wiley Press, New Jersey, USA, June 2005.

[9] I. Foster I. and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer Applications*, Vol 11 No.2: pp. 115-128, 1997.

[10] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .NET-Based Enterprise Grid Computing System", in *Proceedings of the 6th International Conference on Internet Computing (ICOMP'05)*, June 27-30, 2005, Las Vegas, USA.

[11] S. Venugopal, R. Buyya and L. Winton, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", *Concurrency and Computation: Practice and Experience*, Vol. 18 No. 6, pp. 685-699, Wiley Press, New York, USA, May 2006.

[12] POVray project description - http://www.povray.org/documentation/view/3.6.1/3/ (accessed in June, 2006)

[13] Davis, T., "Generating Computer Animations with Frame Coherence in a Distributed Computing Environment", in *Proceedings of the 36th Annual Southeast Regional Conference, April 1998.*

[14] M. Baker, A. Apon, R. Buyya, and H. Jin, "Cluster Computing and Applications", *Encyclopaedia of Computer Science and Technology*, Allen Kent and Jim Williams (editors), Vol. 45 (Supplement 30), Jan. 2002, pp. 87-125, Marcel Dekker, Inc., New York, USA

[15] P. Asadzadeh, R. Buyya, C. Ling Kei, D. Nayar, and S. Venugopal, "Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies", *High Performance Computing: Paradigm and Infrastructure*, Laurence Yang and Minyi Guo (editors), Wiley Press, New Jersey, USA, June 2005.

[16] POVray theory of operation - http://www-mddsp.enel.ucalgary.ca/People/adilger/povray/pvmpov.html (accessed in June, 2006)

[17] B. Freisleben, D. Hartmann, T. Kielmann, “Parallel Raytracing: A Case Study on Partitioning and Scheduling on Workstation Clusters System Sciences”, in *Proceedings of the Thirtieth Hawaii International Conference on System Sciences,* Vol. 1, 7-10 Jan. 1997, pp. 596 – 605

[18] K. Nadiminti, S. Venugopal, H. Gibbins, and R. Buyya, “The Gridbus Grid Service Broker and Scheduler (2.0) User Guide”, Technical Report, GRIDS-TR-2005-4, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, April 22, 2005.

[19] http://www.dazzlersoft.com/ (accessed in May, 2006)

[20] http://www.cogniscienceinc.com/en/cyberediteur-e-description.html (accessed in May, 2006)

[21] http://www.verrall.demon.co.uk/mpipov/ (accessed in June, 2006)

[22] http://pvmpov.sourceforge.net/ (accessed in June, 2006)

[23] http://www.shrek2.com/ (accessed in June, 2006)

[24] http://www.xvid.org/ (accessed in May, 2006)

[25] MegaPov – http://nathan.kopp.com/patched.htm (accessed in June, 2006)

[26] http://www.mysticview.org/ (accessed in June, 2006)

[27] http://www.synthmuseum.com/roland/roltb30301.html (accessed in May, 2006)

[28] POVray theory of operation - http://www-mddsp.enel.ucalgary.ca/People/adilger/povray/pvmpov.html (accessed in June, 2006)

[29] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, “A High-Performance, Portable Implementation of the Message Passing Interface (MPI) Standard”, *Parallel Computing Journal*, Vol. 22 No. 6, September 1996.

[30] http://amiga.emucamp.com/dpaint4.htm (accessed in June, 2006)

[31] R. Buyya and S. Venugopal, “The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report”, in *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models* (GECON 2004, April 23, 2004, Seoul, Korea), pp. 19-36, IEEE Press, New Jersey, USA..

[32] The official Site of POVray http://www.povray.org/ (accessed in June, 2006)

BIOGRAPHY

Bader Aljaber is a final year Master of Software Systems Engineering student in the Department of Computer Science and Software Engineering at the University of Melbourne.

Thomas Jacobs is a final year Bachelor of Computer Science (hons) student in the Department of Computer Science and Software Engineering at the University of Melbourne.

Krishna Nadiminti is a research programmer at the GRIDS Lab, The University of Melbourne. He is one of the lead developers of the Gridbus Service Broker, and is involved in developing the next-generation of service-oriented Grid technologies for the Gridbus project. His interests include desktop Grid computing, resource allocation, middleware development and peer-to-peer networks.

Rajkumar Buyya is an Associate Professor and Reader of Computer Science and Software Engineering, and Director of the Grid Computing and Distributed Systems (GRIDS) Laboratory at the University of Melbourne. He is a world-renowned researcher in the area of Cluster and Grid computing and has co-founded and chaired four IEEE/ACM international conferences: CCGrid, Cluster, Grid, and E-Science. He has presented over 140 invited talks (keynotes, tutorials, and seminars) on his vision on IT futures and advanced computing technologies in several international conferences and institutions in Asia, Australia, Europe, North America, and South America. For further information on Buyya, please browse: http://www.buyya.com.