A mechanism to improve the performance of Hybrid MPI-OpenMP applications in Grid

Shikha Mehrotra, Shamjith KV, Prachi Pandey, Asvija B, R Sridharan

C-DAC Knowledge Park,
#1, Old Madras Road,
Bangalore-560038

***Abstract*— In the current scenario of grid computing, heterogeneous resources are distributed across different administrative domains and geographical boundaries. Every node in a cluster consists of multiple core CPUs wherein the distributed memory across nodes and shared memory co-exists, thereby paving way for hybrid architectures. The hybrid programming approach combines MPI and OpenMP libraries to exploit this hierarchical multicore architecture. The clear requirements of such hybrid application and knowledge of the system architecture will help to boost the application performance. Scheduling these hybrid applications on the grid becomes a critical task for obtaining better performance. In this paper, we outline the attempt made in improving the scheduling mechanism for the hybrid applications based on the requirements of the application.**

***Index Terms*— Grid Computing, Parallel Computing, Hybrid job scheduling**

I. INTRODUCTION

Most HPC systems comprise of clusters [(1)] with shared memory [(2)] nodes. These shared memory nodes usually contain multiple sockets with multiple cores per socket. Traditional parallel programming models such as MPI [(3)] and OpenMP [(4)], quite often run into limitations regarding performance and scalability. Hybrid programming combines the distributed programming paradigms on the node and shared memory programming paradigms within each node. The hybrid programming approach, which combines MPI and OpenMP programming models, seems to be a good approach. However, the scheduler must possess considerable knowledge of system architecture and the clear requirements of the application to enforce better scheduling decisions for hybrid applications thereby enhancing its performance [(5)].

Grid [(6)] computing provides the ability to access, utilize, and control a variety of underutilized heterogeneous resources distributed across multiple administrative domains.

Grid Meta scheduler and grid middleware are incorporated to manage and negotiate with these distributed resources to identify the suitable resource for job submission. Grid Meta scheduler neither considers total cores available per cluster nor the requirement of application. We develop a model for evaluating the requirement of hybrid applications along with an effective scheduling mechanism for resource allocation.

II. OVERVIEW OF GARUDA GRID

GARUDA Grid [(7)] is India's national grid computing initiative, funded by the Department of Information Technology (DIT), Government of India, binds together heterogeneous computational resources, mass storage and scientific instruments. GARUDA aims to provide the technological advances required to enable data and compute intensive science for the 21st century, connecting 28 cities across the nation.

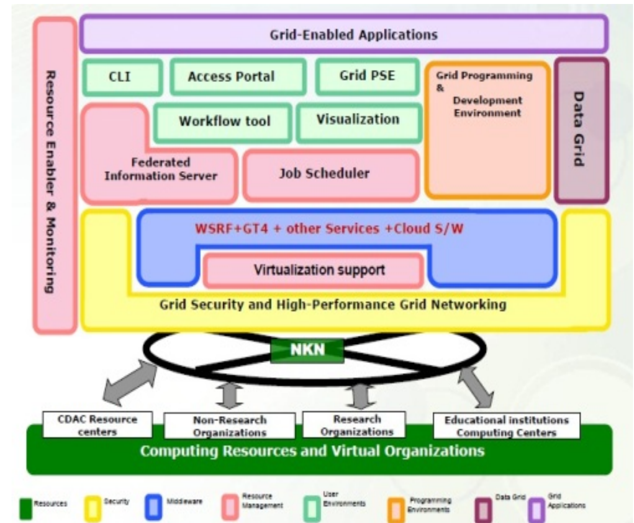


Fig. 1. GARUDA Grid Architecture

The GARUDA network is a Layer 2/3 MPLS Virtual Private Network [VPN] connecting selected institutions at 2.43 Gbps with stringent quality and Service Level Agreements. National Knowledge Network (NKN) [(8)] has undertaken the

implementation of GARUDA network across research, higher education and scientific institutions through ultra high-speed backbone/data-network communication highway, encouraging sharing of knowledge, specialized resources and collaborative research.

The major components of GARUDA are the computing resources, high-speed communication fabric, middleware & security mechanisms, job scheduler, tools to support program development, collaborative environments, data management and grid monitoring & management, as shown in Figure 1.

GARUDA has adopted a pragmatic approach for using existing Grid infrastructure [9] and Web Services technologies. The deployment of grid tools and services for GARUDA will be based on a judicious mix of in-house developed components, the Globus Toolkit (GT) [10], industry grade & open source components. The resource management and scheduling in GARUDA is based on a deployment of industry grade schedulers in a hierarchical architecture. At the cluster level, job scheduling is achieved through Torque (11). Gridway (12) is deployed at GARUDA as the meta-scheduler, which is responsible for scheduling jobs at grid level.

III. Current scenario of hybrid MPI-OpenMP applications in Grid

In a computational grid, each high performance cluster has number of nodes with cores, as depicted in the Figure 2. These nodes are configured with grid middleware, grid meta-scheduler and Local Resource Manager (LRM) to create a parallel processing environment. The meta-scheduler takes the responsibility of job submission on the grid.

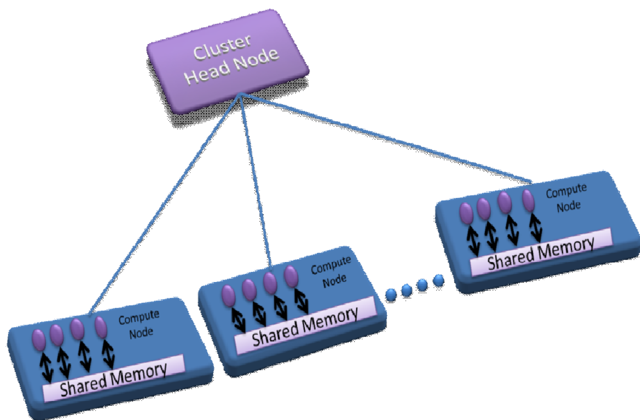


Fig. 2. Cluster Architecture

Prior to job submission onto grid, the meta-scheduler checks for application requirements and selects the suitable resource to schedule the job on the appropriate cluster. Resource broker lists the candidate resources as per the job requirements. Grid meta-scheduler takes complete control of a job from its submission, execution and its output available to the user.

Generally, grid meta-schedulers schedule jobs based on the availability of the resources at any given instance. Gridway is the meta-scheduler used in GARUDA grid. Using Gridway, the user cannot specify the total cores required for their hybrid applications, as it does not take into account the multicore feature of clusters.

IV. Need for Hybrid MPI-OpenMP application support in Grid

In addition to MPI, OpenMP is a specification for compiler directives, library routines (the OpenMP API) and environmental variables used in Fortran and C programs to utilize shared memory and distributed shared memory architectures. The primary advantage of using OpenMP directives lies in the ability of multi processors to access the same memory pool, without the costly communication overheads and network transit times found in message passing. The clusters of compute nodes with shared-memory multiprocessors provide a good platform for parallel applications. The concept of using MPI between cluster nodes and OpenMP within a node results in less overhead in the shared memory environment. This leads to the emergence of hybrid MPI-OpenMP programming [13].

The requirement of MPI application is the number of processors, whereas the OpenMP applications require the number of hardware threads in the node, as the input parameter. Therefore, the job submission of hybrid MPI-OpenMP application requires two input parameters – the number of nodes and number of hardware threads in the node.
.

The job submission on grid therefore necessitates meta-scheduler to filter resources based on application's requirement. This underscores the importance of providing a new improved scheduling mechanism based on hybrid application demand in a grid computing environment. While executing an MPI application, each node runs only one MPI process no matter how many cores it has. The MPI process then forks OpenMP threads on the node, which in turn complete execution in parallel. The effective scheduling of hybrid applications depends on the application requirements and total number of cores in the cluster node.

V. Major components of a Grid

Figure 3 represents the components of a Grid. Grid Meta-scheduler plays a major role in selecting the right resource based on the application requirement and is responsible for judiciously scheduling jobs on to the grid. It provides the interface for users/applications/portals [14] for submitting jobs onto grid. The local resource manager, the lowest layer entity, is responsible for getting the jobs run on a cluster system.

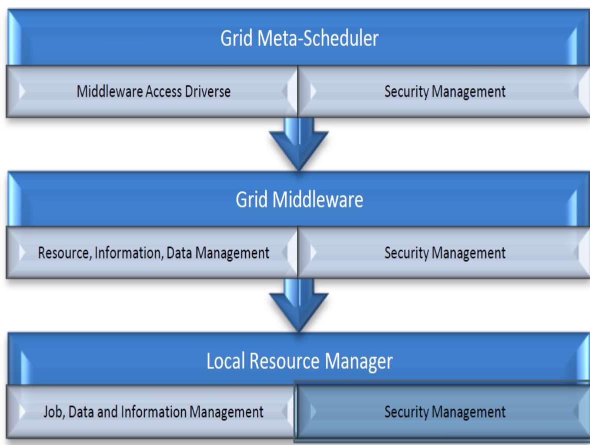


Fig. 3. Components of Grid

Another major entity, Grid middleware and the bridge system co-ordinates and accomplishes the filtering activity for resources. This component is responsible for integrating both Meta scheduler and Local resource manager for the smooth running of hybrid applications.

VI. IMPLEMENTATION

In the current scenario, Gridway is deployed as the Grid Meta scheduler on GARUDA. The figure 4 [(15)] depicts the components of Gridway, which consists of Gridway Core, Scheduler, Information manager, Execution manager and Transfer manager.

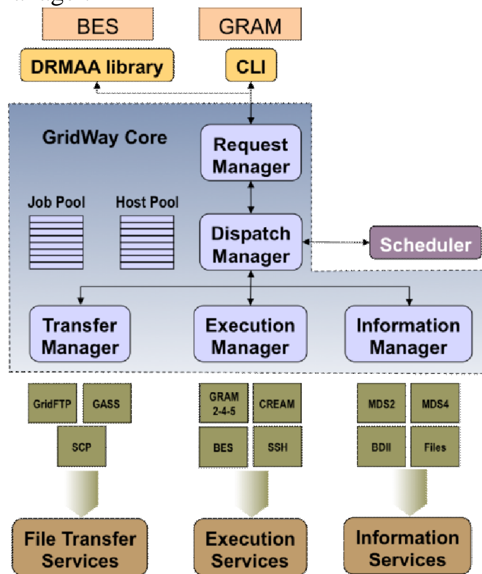


Fig. 4. Components of Gridway

Gridway accepts the number of nodes for the parallel applications as its input parameter. However, the hybrid MPI-OpenMP application need to specify the number of cores required along with node requirements for its job execution. Gridway is customized to accept additional input parameter, understand and implement this requirement at different layers of Grid, in turn helping to improve application performance.

The development has been made at three different layers of a grid as shown in the Figure 5.

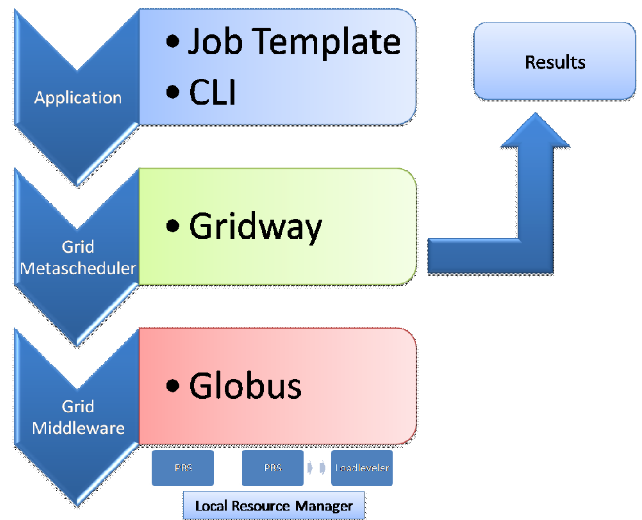


Fig. 5. Different layers in Grid

As shown in the figure 5, Grid comprises of three layers, namely Grid Middleware, Grid Meta-scheduler and Application layer. The grid middleware layer aids in enabling a high performance computing cluster by providing necessary security, information, job and data management facilities between different resources in grid. The local resource management layer is actually responsible for spawning the different processes of a parallel application across the computing nodes of the cluster. The middle layer, grid meta-scheduler layer, takes care of job allocation to the high performance clusters. The Application layer provides an interface to users for job submission.

Figure 6 depicts the modification that has been introduced in the meta-scheduler level to provide the support for hybrid application requirements from user.

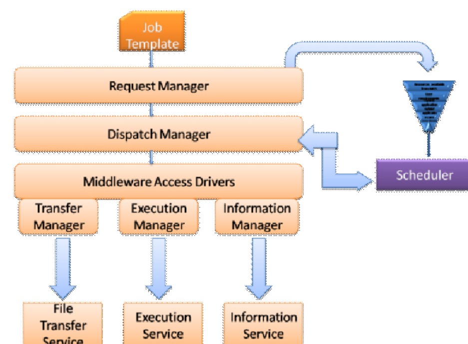


Fig. 6. Gridway modification for Hybrid MPI-OpenMP application Support

The Grid meta-scheduler, Gridway has been customized to support hybrid applications. In order to support number of cores required for the hybrid application, the Job Description Language (JDL) [16] of Gridway has been extended. A new parameter, NUM_THREADS is introduced either to describe the number of threads application creates or the number of cores in the node required by the hybrid application.

The extended Gridway uses the parameter NUM_THREADS to filter the resources as per the requirement of hybrid application. The meta-scheduler Job Description Language (JDL) processing mechanism has been modified to accept and interpret NUM_THREADS for indicating the number of threads spawned by the application. The minimum number of threads to run on each core decides the performance of their application. The extended Gridway ensures to enhance the application performance by selecting appropriate resources for job scheduling.

Gridway provides the command "gwhost" to list the availability of resources. The knowledge of number of cores available in the cluster node would help user to specify the parameter NUM_THREADS, thereby enhancing their application performance. The better performance can be achieved with considerable knowledge of system architecture and the requirements of the application. Therefore, the Gridway resource monitoring command has been modified to include the additional information, i.e. the cluster's multicore property.

The role of meta-scheduler, Gridway, for job submission includes identifying appropriate resource as per application requirement, scheduling jobs on those resources, stage-in input(s), and stage-out output(s) in the selected cluster. Then the task of job execution passes on to grid middleware, Globus. The Gridway Job Description Language (JDL) provides necessary information to grid middleware deployed in the identified cluster, for job execution. The grid middleware has to be modified to implement the additional feature of the extended Gridway, thereby supporting hybrid applications.

The Globus 4 based on Web Service Resource Framework (WSRF), the de facto standard for the Grid middleware, supports web service based jobs as well. The Globus middleware consists of four components, namely, Security Management, Information Management, Job Management and Data Management.

The Security Management in a Globus middleware is handled by Grid Security Infrastructure (GSI). The Information Management in Globus is accomplished through the Monitoring and Discovery System (MDS) [17] for providing information about the available resources on the Grid and their status. The Monitoring and Discovery System (MDS) is a suite of Web Services (WS) to monitor and discover resources and services on Grids.

The Data Management component of Globus takes care of both data transfers and replication as required by the grid application for staging-in their inputs and staging-out their outputs, along with necessary data replication based on the proximity of simulation run. These data movement activities are accomplished through the GridFTP protocol. GridFTP is the grid enabled version of the FTP protocol, for providing the secure, robust, fast and efficient transfer of huge data. The Reliable File Transfer (RFT) service aids data transfer through web service interface. The data replication is accomplished through the Replica Location Service (RLS), a tool that provides the ability to keep track of one or more replicas of files in a Grid environment. This is very helpful for users or applications that need to find where existing files are located in the Grid.

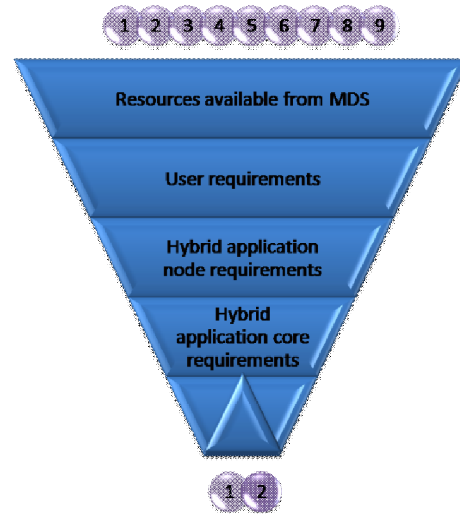


Fig. 7. Resource Filter

The Job Management component of Globus is responsible for submitting the application to the computing resources, by co-coordinating with other Globus components. The job management in Globus enabled grid is primarily under the control of Globus Resource Allocation Manager (GRAM).

The Globus Resource Allocation Manager (GRAM) integrates new module to include application's core requirement, as specified in (Gridway) JDL. The understanding of new parameter by LRMS facilitates the actual job execution in the identified cluster nodes as per application requirement.

Even though the integration of new modules with the GRAM component of Globus is carried out, it will be completed only if the core requirement, which has been selected by the user through the job descriptions specified to the grid meta-scheduler, is integrated with the Local Resource Manager System (LRMS). LRMS is primarily responsible for actual job submission on the high performance computing

cluster. The necessary integration modules are added to facilitate this activity. Hence, a user can submit their hybrid MPI-OpenMP applications by specifying the required number of nodes and cores in node, ensuring the resource allocation as per application requirement and eventually improving the application performance.

VII. CASE STUDY

To demonstrate the effectiveness of the approach followed in our paper, we have conducted extensive study and experiments by submitting hybrid MPI-OpenMP applications on to the test bed to analyze the application performance. We have identified some HPC resources which are part of GARUDA computational grid infrastructure distributed across the nation. The test bed deploys Gridway as Grid meta-scheduler, Globus as Grid middleware and Torque as LRM.

To analyze the improvement in performance, we identified Stream benchmark [(18) (19)], a hybrid application, to be executed on the selected resources with and without the customization to support hybrid applications. The application requires two processors for execution. The clusters have 4, 8, 16 and 24 cores each. We conducted the same experiment twice by varying the parameter NUM_THREADS.

Experiment1: application spawning 8 threads

Case 1.1*: Without implementing the proposed mechanism*

In the first case, our application creates 2 processes spawning 8 threads each and results are noted for 8 different runs. Table 1 records the application execution time for 8 runs, without implementing the proposed mechanism.

	A-Cluster (4 cores)	B-Cluster (8 cores)	C-Cluster (16 cores)	D-Cluster (24 cores)
Run 1		9.19		
Run 2	18.23			
Run 3	17.56			
Run 4				9.56
Run 5		9.49		
Run 6			9.84	
Run 7			10.45	
Run 8				9.45
Mean	17.895	9.34	10.145	9.505
Mean execution time for the grid				
11.72125				

TABLE 1. EXECUTION TIME FOR CASE 1.1

The graph is plotted accordingly as shown in Figure 8. The x-axis depicts the execution time in seconds and y-axis shows values for 8 different runs.

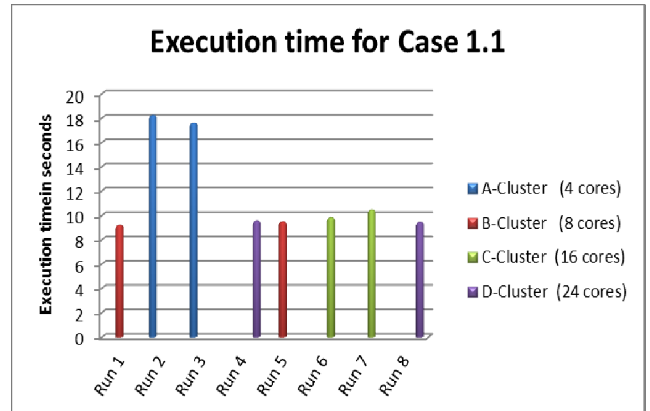


Fig. 8. Graph for case 1.1

Case 1.2: With the proposed mechanism setup

For the second case, we execute the application on the test bed implementing the proposed mechanism for improving application performance. The execution time is tabulated in Table 2, for 8 runs and the graph for the same is plotted in Figure 9. The mean execution time is evaluated for both the scenario.

	A-Cluster (4 cores)	B-Cluster (8 cores)	C-Cluster (16 cores)	D-Cluster (24 cores)
Run 1			10.24	
Run 2		9.48		
Run 3				9.67
Run 4		9.34		
Run 5			9.45	
Run 6				9.29
Run 7				9.32
Run 8		9.61		
Mean	0	9.476667	9.845	9.42666667
Mean execution time for the grid				
9.582777778				

TABLE 2. : EXECUTION TIME FOR CASE 1.2

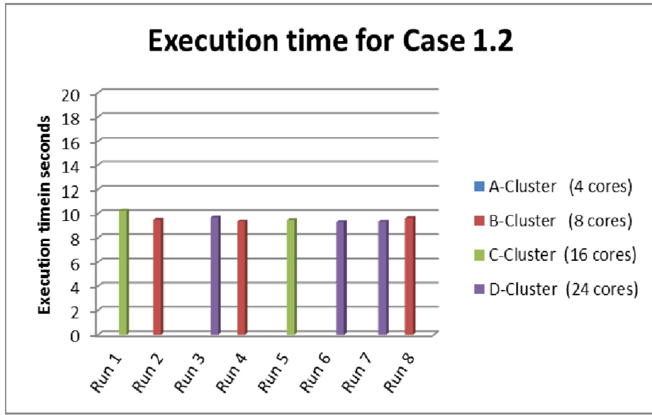


Fig. 9. Graph for Case 1.2

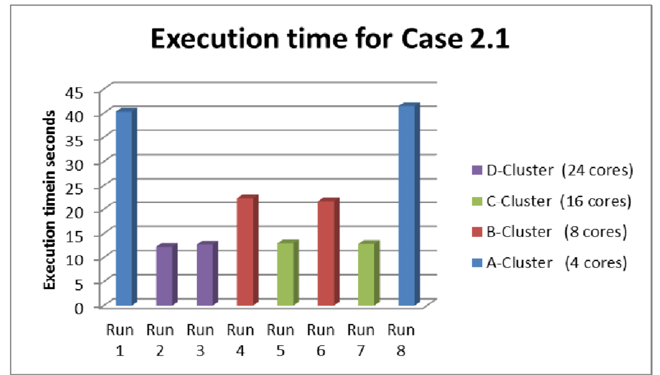


Fig. 10. Graph for Case 2.1

By analyzing both the cases, we notice that irrespective of the application requirement, the application is scheduled for execution on all the resources, in case 1.1. The execution time varies significantly for each run, with mean execution time of 11.27 seconds.

In case 1.2, the application is not scheduled on resources which do not fulfill the application requirement, thereby reducing the average execution time for hybrid application. The execution time for each run remains nearly constant. We can observe that the mean execution time is reduced considerably while implementing the proposed mechanism.

Experiment 2: Application spawning 16 threads

Case 2.1: *Without implementing the proposed mechanism*

In the first case, our application creates 2 processes spawning 16 threads each. Table 3 lists the application execution time for 8 runs, without implementing the proposed mechanism. The graph is plotted accordingly as shown in Figure 10.

	A-Cluster (4 cores)	B-Cluster (8 cores)	C-Cluster (16 cores)	D-Cluster (24 cores)
Run 1	40.32			
Run 2				12.24
Run 3				12.74
Run 4		22.34		
Run 5			13.01	
Run 6		21.69		
Run 7			12.89	
Run 8	41.51			
Mean	40.915	22.015	12.95	12.49
Mean execution time for the grid				
22.0925				

TABLE 3. EXECUTION TIME FOR CASE 2.1

Case 2.2: *With the proposed mechanism setup*

In the second case, we execute the application on the test bed implementing the proposed mechanism for improving application performance. The execution time is tabulated in Table 4, for 8 runs and the graph for the same is plotted in Figure 11. The mean execution time is evaluated for both the scenario.

	A-Cluster (4 cores)	B-Cluster (8 cores)	C-Cluster (16 cores)	D-Cluster (24 cores)
Run 1				12.62
Run 2				12.93
Run 3			12.94	
Run 4			13.38	
Run 5				12.45
Run 6			12.54	
Run 7				13.31
Run 8			12.86	
Mean	0	0	12.93	12.8275
Mean execution time for the grid				
12.87875				

TABLE 4. EXECUTION TIME FOR CASE 2.2

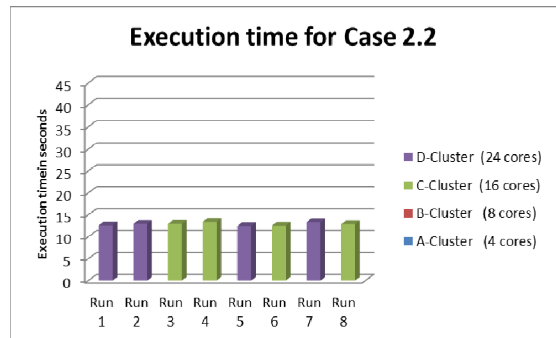


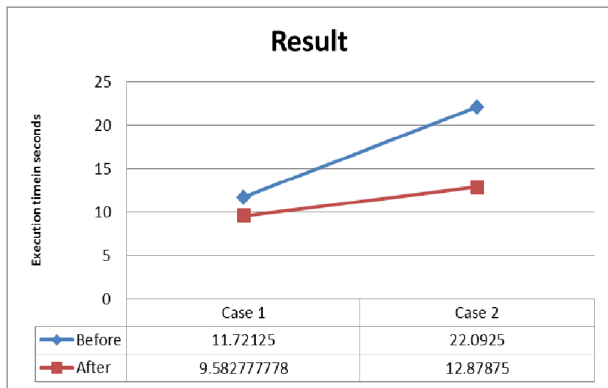
Fig. 11. Graph for Case 2.2

When both the cases are analyzed, we notice that irrespective of the application requirement, the application is scheduled for execution on all the resources. The mean execution time in both the cases varies greatly, i.e 22.09 seconds and 12.87 seconds for case 2.1 and case 2.2 respectively.

Figure 12 plots a bar graph representing the mean execution time for both the cases.

Fig. 12. Execution time comparison

The cases with and without the proposed mechanism for case 1 and 2 respectively are plotted on the x-axis. The y-axis represents execution time in seconds. We observe that if jobs are scheduled at appropriate clusters, there is a noticeable difference in the mean execution time for both the cases.



	Case 1	Case 2
Before	11.72125	22.0925
After	9.582777778	12.87875

Fig. 13. Result Analysis

The figure 13 can be plotted as a line chart to effectively highlight the impact of proposed mechanism.
With figure 13, we can understand that as the number of threads spawned by an application increases, the difference in mean execution time is greatly increased.

VIII. SIMILAR WORK

The proposed mechanism for improving the performance of hybrid MPI-OpenMP applications proves to be a very useful feature in a heavily used grid computing environment. There has been no remarkable attempt facilitating support for hybrid applications in current grid scenario. The proposed system is implemented in GARUDA grid. Most of the works carried out earlier either concentrate on effective scheduling of MPI jobs or core level scheduling of Multi-threaded applications. However an attempt to unify these strategies, to harness the optimal utilization of the resources, has not been carried out effectively.

IX. FUTURE WORK

Hardware accelerators on HPC resources are becoming the key factors influencing the application performance. We are researching on developing new scheduling approaches for hybrid applications with GP-GPU [20], and MIC [21] program modules, in a distributed computing scenario, specifically with respect to Grid. This would pave a path for discovering the optimized resources for running hybrid applications in a heterogeneous Grid environment.

X. CONCLUSION

The emergence of hybrid applications presents the need for their enablement in Grid environment. The implemented mechanism allows user to specify the requirement of hybrid applications and schedule the jobs accordingly. The selection of right cluster for submitting hybrid application becomes very critical, in terms of performance improvement.

So the implementation must address the resource identification and allocation based on the requirement of hybrid applications. Grid middleware, grid meta-scheduler and Local Resource Managers must align themselves to support hybrid applications. The existing grid meta-scheduler is tuned to understand hybrid application parameters to identify the appropriate cluster and resource allocation for the job submission. The grid middleware and Local Resource Managers are modified and configured to understand and implement the core requirement to facilitate the execution of hybrid applications. The case study on GARUDA grid testbeds indicates the usefulness of the proposed mechanism. The experiments conducted revealed the enhanced performance for hybrid application execution.

XI. REFERENCES

1. Computer cluster. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Computer_cluster.

2. Shared Memory. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Shared_memory.

3. **Barney, Blaise.** Message Passing Interface (MPI). *https://computing.llnl.gov/tutorials/mpi/.* s.l. : Lawrence Livermore National Laboratory.

4. *OpenMP: an industry standard API for shared-memory programming.* **Dagum, L.** 1, Jan-Mar 1998, Computational Science & Engineering, IEEE, Vol. 5, pp. 46- 55 .

5. **Hysom, E. Chow and D.** *Assessing Performance of Hybrid MPI/OpenMP Programs on SMP Clusters.* Livermore CA : Lawrence Livermore National Laboratory, 2001. UCRL-JC-143957.

6. *What is the Grid? A Three Point Checklist.* **Foster, Ian.** s.l. : GRIDToday, 2002.

7. *e-Infrastructures in IT: A case study on Indian national grid computing initiative – GARUDA.* **B. B. Prahlada Rao, S. Ramakrishnan, M. R. Raja Gopalan, C. Subrata,N. Mangala, and R. Sridharan.** 3-4, s.l. : Springer-Verlag, June 2009, Computer Science- Research and Development, Vol. 23, pp. 283 - 290.

8. *National Knowledge Network.* [Online] http://www.nkn.in/.

9. **Ian Foster, Carl Kesselman.** *The grid: blueprint for a new computing infrastructure.* s.l. : Morgan Kaufmann Publisher, 1999.

10. *Globus Toolkit Version 4: Software for Service-Oriented Systems.* **Foster, I.** s.l. : IFIP International Conference on Network and Parallel Computing, 2006. pp. 2-13. LNCS 3779.

11. *Torque.* [Online] http://www.adaptivecomputing.com/products/torque.php.

12. *The GridWay Framework for Adaptive Scheduling and Execution on Grids.* **Eduardo Huedo, Rubén S. Montero, Ignacio Martín Llorente.** 3, s.l. : Scalable Computing:Practice and Experience, 2005, Vol. 6, pp. 1-8. ISSN 1895-1767.

13. *Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes.* **Rolf Rabenseifner, Georg Hager,Gabriele Jost.** s.l. : Parallel, Distributed and Network-based Processing, 2009.

14. *An access mechanism for Garuda Grid.* **Arackal, V.S., Arunachalam, B., Bijoy, M.B., Prahlada Rao, B.B.,Kalasagar, B., Sridharan, R., Chattopadhyay, S.** Bangalore : Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference, 2009.

15. Internal Architecture of Gridway. *www.gridway.org.* [Online] http://gridway.org/doku.php?id=documentation:release_5.14:ia.

16. *The Grid[Way] Job Template Manager, a tool for parameter sweeping.* **A. Lorca, E. Huedo, I.M. Llorente.** s.l. : Computational Physics Communications, 2011. 1047-1060.

17. *A Performance Study of Monitoring and Information Services for Distributed Systems.* **X. Zhang, J. Freschl, and J. Schopf.** s.l. : HPDC, August 2003.

18. The STREAM Benchmark: Computer Memory Bandwidth. [Online] [Cited: April 30, 2013.] http://www.streambench.org/.

19. STREAM: Sustainable Memory Bandwidth in High Performance Computers. [Online] [Cited: April 30, 2013.] http://www.cs.virginia.edu/stream/.