GANGA: Grid Application iNformation Gathering and Accessing framework

Haresh S. Bhatt[1], Dharmesh Bhansali[1], Sonal Shah[1], P R Patel[1], VH Patel[1], Arup Dasgupta[2]

[1]SATCOM and IT Applications Area
Space Applications Centre
Indian Space Research Organisation
Satellite Road
Ahmedabad – 380 015, INDIA

[2]Dharamsi Desai Institute of Technology
Dharamsi Desai University
College Road
Nadiad

{haresh, dharmesh, sonal, praful, vhpatel}@sac.isro.gov.in
arup@ieee.org

Abstract

Future generation Grid technology aims to provide transparent and easy access of remote resources to end-users. Parallel and distributed computing environments like Condor, PBS, WebDedip, load leveller, etc are now becoming Grid compliant. These environments address interoperability issues to certain extent by extending Resource Specification Language (RSL) as per their requirements. In fact, such an extension in fact restricts interoperability. In this paper, we propose an alternative mechanism that addresses the interoperability issues.

I. Introduction

The term "the Grid" was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering [43]. Since then, it is expanding its horizons in multi dimensions. Various protocols, standards, architectures, infrastructures and toolkits are being worked out. Ian Foster et. al. has defined its anatomy [1], physiology and architecture [2]. Number of computing environments like Condor-G [40], IBM Load Leveller [9], PBS [8,44,23], Nimrod-G [4], WebDedip [18], GANESH [27], Mauvi & Silver [41], Ninf [22], Javelin [35] are becoming Grid enabled. However, the applications configured on any of the above environments can only be executed on that particular environment [38]. Ninf and Netsolve have tried to make their environment interoperable. The inter-operability between the execution environments is essential to port the applications from one environment to another. It will also benefit

the application developer, who wants to use the additional features of the other execution environments. This paper presents a framework to provide the inter-operability among execution environments.

The applications archived in a standardized manner will provide the feature of universal access that will enable generalization and interoperability among execution environments. This will provide the flexibility to configure the applications only one time and execute it on various environments any time and many times. It will also help the operation institutes in carrying out systematic execution of various applications (as a part of periodic operations) through operators as requested by application designers or end users.

Globus toolkit [3] and CogKits [11] provides various tools for grid computing. GRAM [7] addresses five major challenges of site autonomy, heterogeneous substrate, policy extendibility, co-allocation and online control. Grid Resource Information Protocol [32], Grid Resource Registration Protocol [21] and Grid Resource and Information Service [36] provide mechanism for resource information management. They have not addressed application information management.

GASS [6] is very useful in staging executable on remote machines for execution while RSL [13] is the language used by the clients to submit a job. All job submission requests are described in RSL, including the executable file and condition on which it must be executed. RSL defines a bare minimum parameter required for application invocation in a right manner. RSL is an extendible language to support additional parameters.

Though Globus toolkits have tried to address certain issues related to interoperability, there are several issues still unresolved**.** RSL defines a set of primitive parameters but they are not sufficient to address all the requirements of individual environments. Condor-G, PBS, GANESH, Nirmod-G, etc have used extendibility of RSL to incorporate their requirement. But in doing so, their interoperability is very much restricted to few parameters supported by RSL.

Each environment have plenty of parameters to address various issues related to scheduling, match making, resource management, fault tolerance, monitoring, etc. Condor has about 36 machine resource attributes and 46 job attributes. Condor depends on DAGMan [12] for handling job interdependency. Load leveller has about 38 parameters. Moavi has more then 200 parameters. WebDEDIP and GANESH have about 20 parameters, which include job interdependency information too. Each environment addresses various issues differently and many a times gives different parameter names for similar work or vice versa. For example, Condor accepts single Job while DEDIP accepts Application consisting of several processes (Jobs), and Load Leveller accepts Job that contains different steps. Similarly, Condor uses term JobPrio while Load Leveller uses User_Priority whereas DEDIP uses only Priority. Thus extending RSL for making an environment Grid Complaint do not serve the purpose and none of them can be interoperable.

Normalizing all these parameters to common platform is a challenging job. Decision for parameter conversion among the different environment is a very difficult task.

Several working groups like jsdl-wg [45], wfm-wg [46], etc. are putting collaborative efforts in standardizing such interoperable interface, but are in draft stage, and will take several iterations to reach a mature stage. To make a framework that complies with rapidly changing standards is also a difficult job.

JSDL [29] has emerged as a result of the collaborative efforts in working out the common most usable parameter along with its XML schemas. It is currently in premature stage. Our framework is XML based and can be easily made JSDL compliant in future. Furthermore, JSDL doesn't address issues like how to include the Direct Acyclic Graph (DAG) information? Our framework addresses all such requirements of various environments like DAGMan, WebDedip, GridAnt [28], [47] and GANESH. It supports repetitive jobs to certain extent. DAGMan and WebDEDIP store their Direct Acyclic Graph information in text format. DAGs are widely spread due to their simple structure. Unicore [24] also uses DAGs. DAG is acyclic, so it is not feasible to explicitly define loops without additional language elements that are not related to the graph representation. GridAnt uses Ant [34] for handling its repetitive workflow requirement and uses its own proprietary XML representation. Ref [47] is based on Petry Nets [20] and also uses its proprietary XML representation [37]. GANESH extends WebDedip to address a set of workflow requirements like GridAnt. GANESH uses GANGA registry.

Our model also provides a repository that not only supports services for archival and retrieval of application information of individual environments, but also provides services for conversion from one environment to another.

GANGA is a name of a sacred river in India. It is scientifically proven that its water is pure, remains pure for a pretty long time and is capable of removing impurities of water added from any other source. We have used the name aiming at such a framework which not only provide pure interoperability, but also has the capability to remove impurities which may come while normalizing other environments from time to time.

II. Goal and Objectives

GANGA's soul vision is to have a standard format for storing application specific information, so that there can be interoperability between different grid environments including meta-schedulers supporting DAGs. Its framework presents the way of specifying application specific information with the set of abstract classes. The primitive services are defined for the same. The abstract classes and services are implemented to make GANESH to interoperate with DAGMan and Condor. Its registry will also be very useful for virtualization that Grid computing offers wherein end-users view large number of applications that are seamlessly executed on multiple resources under different environments, as operational under single umbrella.

II. GANGA Model

This paper presents GANGA framework along with an architecture that uses the framework to address interoperability issues.

A. GANGA framework

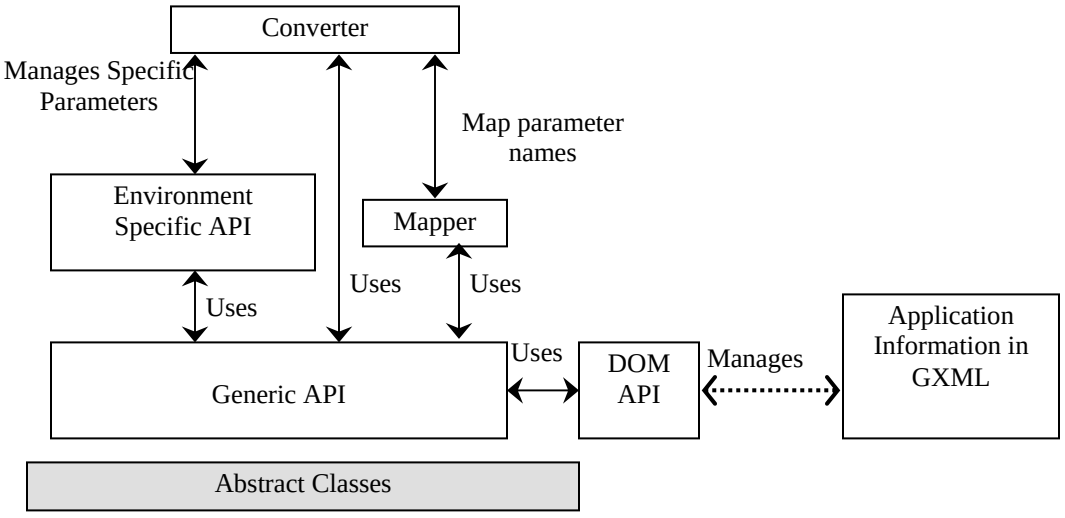

Fig 1: GANGA Framework

GANGA framework is shown in Fig 1. A set of abstract classes are developed as part of its framework after study of Condor [10], Condor-G, PBS, GANESH, Load Leveller, Nirmod-G, DEDIP [17], DAGMan, WebDedip, WebDedip load balancer [19], Mouvi & Silver [41] and JSDL. To implement each and every requirements for all the environments is a difficult task and not possible by a single person. We have generated General API base extending abstract classes for DAGMan, Condor, WebDedip, WebDedip load balancer and GANESH.

Although we emphasize to address all the issues in generic API, there are set of requirements that are specific to a particular environment and not needed in other

environment. For example WebDedip needs to store matrix information to show its DAG in its web based GUI. Such requirements should be implemented in Environment specific API of GANGA frame work so that it can be monitored and can be generalized whenever any other environment come across similar requirements. Mapper maps environment specific parameters to the GXML through generic API while Converter converts current configuration files into GXML database.

B. GXML

XML is used to work out GXML for defining application information. The idea of using JSDL instead of GXML was dropped as JSDL was in premature stage without any schema at that time. However, our GXML implementation can easily be JSDL compliant on its maturity. Document Object Model (DOM) **[49]** is used to have ease of use in working with XML. GXML also incorporates the job interdependency information typically shown in Fig 2. It's GXML schema is shown in Fig 3. The repetitive jobs are also handled. A simple case is shown in Fig 4. It's GXML is shown in Fig 5. The application designer has the freedom to run number of iterations. Two successful codes are defined for Job-C. If it is normal successful, the next child job(s) will be scheduled. If it returns "LoopBack", Job-B will be rescheduled. Meta scheduler of Ganesh takes care of the scheduling as well as data transfer accordingly. GANESH has another simpler way to handle repetition for set of applications where user enters fixed number (N) of iterations required in its GUI. GXML presents the similar case as shown in Fig 6 showing N iterations.

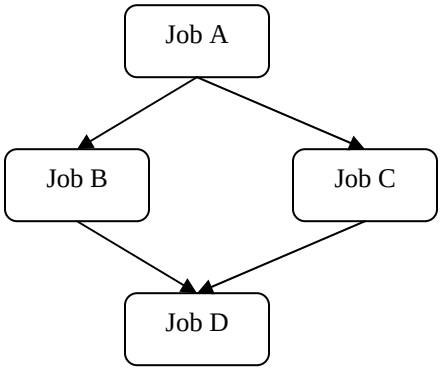

Fig 2: A typical DAG representation

```
<flow:Workflow flow:start="A">
  <jsdl:Job jsdl:id="A">
      ...
  </jsdl:Job>
  <jsdl:Job jsdl:id="B">
    <flow:Depend flow:success="A"/>
      ….…
  </jsdl:Job>
  <jsdl:Job jsdl:id="C">
    <flow:Depend flow:success="A"/>
      ...
  </jsdl:Job>
  <jsdl:Job jsdl:id="D">
   <flow:Depend flow:success="B & C"/>
      ...
  </jsdl:Job>
</flow:Workflow>
```

Fig 3: A typical Schema for application shown in Fig A

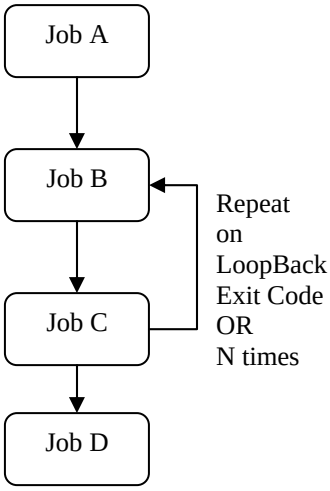

Fig 4: A typical Application with repetitive Jobs

```
<flow:Workflow flow:start="A">
  <jsdl:Job jsdl:id="A">
      ...
  </jsdl:Job>
  <jsdl:Job jsdl:id="B">
    <flow:Depend flow:success="A"/>
      ...
  </jsdl:Job>
  <jsdl:Job jsdl:id="C">
    <flow:Depend flow:success="B">
   <flow:LoopBack flow:To=”B”>
       ...
  </jsdl:Job>
  <jsdl:Job jsdl:id="D">
   <flow:Depend flow:success="C">
      ...
  </jsdl:Job>
</flow:Workflow>
```

Fig 5: A typical Schema for Application shown in Fig C

```
<flow:Workflow flow:start="A">
 <jsdl:Job jsdl:id="A">
     ...
 </jsdl:Job>
 <jsdl:Job jsdl:id="B">
   <flow:Depend flow:success="A"/>
     ...
 </jsdl:Job>
 <jsdl:Job jsdl:id="C">
   <flow:Depend flow:success="B">
  <flow:LoopCount=N flow:To=”B”>
      ...
 </jsdl:Job>
 <jsdl:Job jsdl:id="D">
  <flow:Depend flow:success="C">
     ...
 </jsdl:Job>
</flow:Workflow>
```

Fig 6: A typical Schema for Application

C. GANGA Services

Our aim is to follow OGSA and OGSI [16] in future. Hence, we have a few well defined services that can be used by any portal, specific environment, meta scheduler, workflow manager, etc. to (1) add a new Application Information (application discovery), (2) modify existing Application Information, (3) delete Application Information, (4) get list of Applications, (5) get Application Information, (6) get Job Information for an application, and (7) convert existing Application Information.

These services are used in our architecture as bridge between GANGA registry and others.

D. Architecture

A combination of layer [24] and BEC [23] patterns is followed in our architecture. Also UML [25] and RUP [26] are followed to work out reusable components of the architecture, so that it easily becomes change-resilient. GANGA architecture is shown in Fig 7.

At the top most layers, users who are application developers, resource managers, operators, etc are placed. They use various portals or specific environment (Condor-G , DAGMan, PBS, Load Leveller, etc.) or, workflow managers (WebDedip, GridAnt, WebFlow[5], [47], etc.) for application development,

configuration, installation, execution and monitoring. Currently they create environment specific configuration files to store information about its DAG, data dependency, job repetitions, job information, and resource requirements as supported by respective UI or toolkits.

In GANGA architecture, all such portals, environments, workflow managers, etc will invoke its well-defined services to store and retrieve complete information in GANGA registry. Existing application information residing in respective configuration files can be converted into the GXML, through services, to reside in GANGA registry.

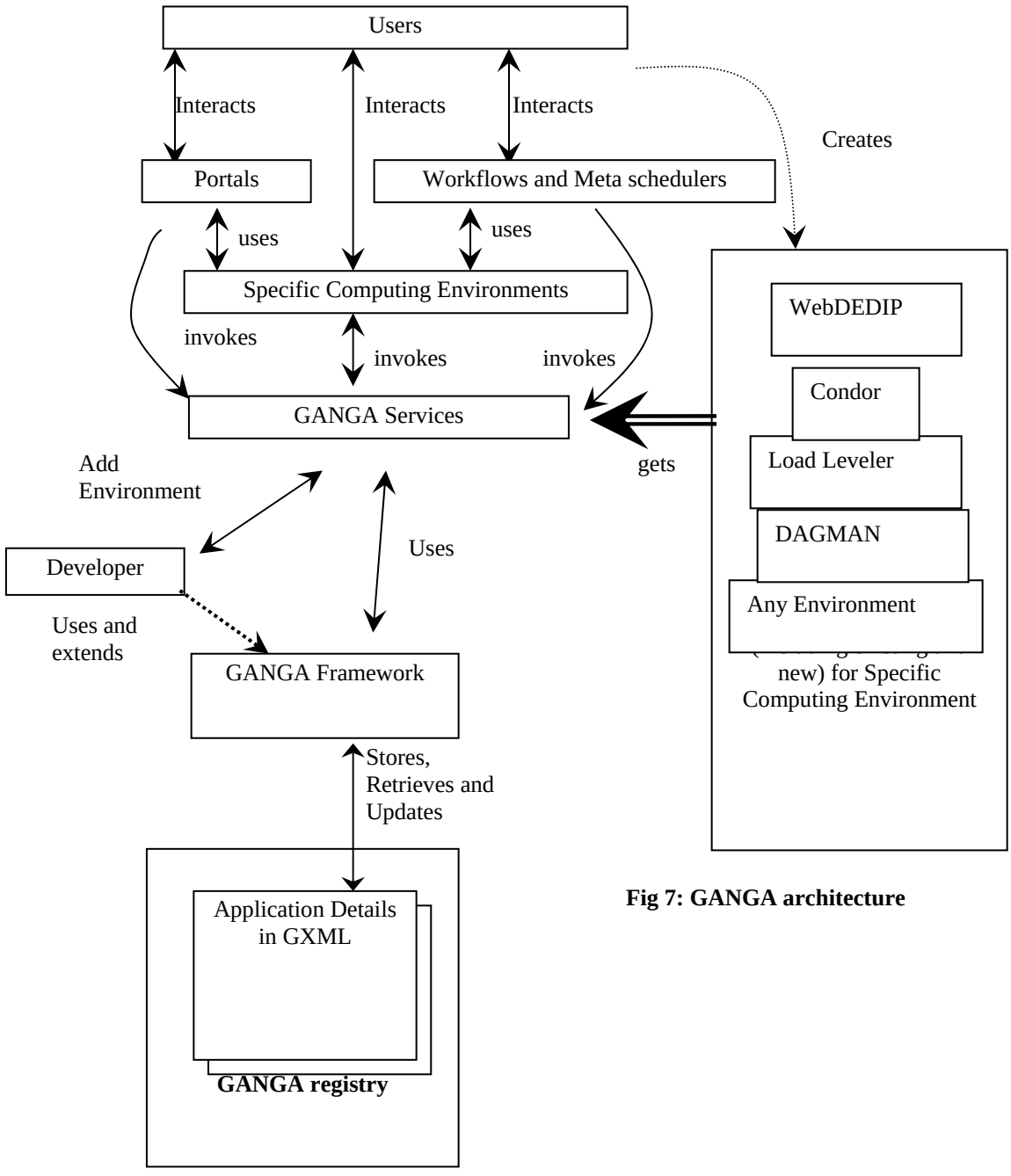

Fig 7: GANGA architecture

At the bottom most layer, there is GANGA registry containing information about all the registered jobs. It contains complete information including DAG information, job details, data dependency information, resource information of each job for each application. MDS-2 of the Globus toolkit provides mechanism for discovering and disseminating information about the structure and state of the grid resources [31]. However it could not be extended for application information management as it is read only and deprecated. Hence, LDAP [42] based repository was developed.

GANGA framework is a bridge between services and registry. It is the core part addressing the issues of parameter normalization among the environments. The current prototype supports all the requirements of WebDedip and its load balancer as well as most useful requirements of DAGMan and Condor.

A developer of a specific environment can extend the generic API following the frame work to address his parameter normalization. Finalization of JSDL and other standards will make this task easier. One can use and extend generic API and modify existing or add new service to make his environment interoperable.

III. Performance Evaluation

GANGA framework is developed using Java technologies (Java API for XML Processing [JAXP], Java Naming and Directory Interface [JNDI], core Java, etc.) to make it portable on any platform supporting JRE or JVM. Currently, abstract classes are implemented based on information analysed from DAGMan -Condor, PBS, Ganesh, Mauvi and Load Leveller. Generic API, mapped and converter are also implemented to meet the interoperability requirements for Ganesh, DAGMan-Condor to prove our concept.

We have successfully converted all the Ganesh applications into GXML and vice-verse and stored into GANGA registry. Their job execution interdependency structures in terms of Directed Cyclic Graph (DAG), is also stored. Similarly, we have also converted sample DAGMan-Condor application into GXML and vice-verse. Further Ganesh to GXML to DAGMan-Condor and vice-verse is also tested. Job repetitions are also addressed. Following scenarios were emerged while testing the interoperability between Ganesh and DAGMan-condor.

A. Scenario 1: GUI-based interactive jobs

Certain applications like distributed image processing may require GUI-interface as well as high-performance computing during its execution. We carried out performance evaluation test for such a scenario using simulated application in both the environments. Two types of sample of GUI-based interactive applications were used, one was in JAVA provided by Ganesh and non-JAVA provided by Condor.

For Non-JAVA applications, we could configure and execute sample application in Ganesh environment successfully for Windows as well as Linux. But when we tried the same in DAGMan-Condor environment, we encountered two problems. First, in order to run the application in Windows operating system , we had to explicitly add the additional parameter USE_VISIBLE_DESKTOP = true on each machine in the pool to display the GUI. Second, the former mechanism did not work for Linux. On Linux, application developers need to handle it explicitly in their applications.

For JAVA applications, we could configure and execute sample application in Ganesh environment successfully for Windows as well as Linux. But generates error in DAGMan-Condor.

For interoperability, we suggest that Condor should have the similar mechanism for Linux, that it has for Windows to handle GUI-based interactive jobs. It should take the corrective measures for interactive JAVA-based jobs. If it does not provide support than at least it should either return error or reject the request for such a job.

JSDL should provide an additional parameter for GUI-based interactive jobs, that can be useful to broker in deciding the suitable environment for such jobs.

B. Scenario 2: Input, intermediate, output and execution files (SAMD Architecture)

Consider a case in which multiple instances of a job are running in parallel on the same machine as depicted in fig 7.

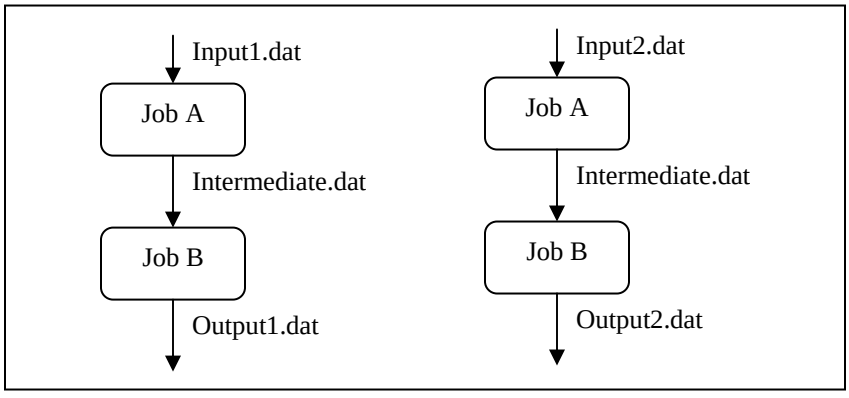

Fig 7: Multiple instances of a job running in parallel on the same machine

In the above case, the job that execute later may overwrite the Intermediate.dat file generated by the earlier job. Both the environment handles this problem differently. Condor handles the above problem transparently by executing each instance in separate directory. It presumes that both executables and data files should exist in the same directory. Ganesh provides API to handle this by appending the instance counter with each file like Intermediate_001.dat,

Intermediate_002.dat etc to prevent file overriding. Therefore, application programmers are bound to use its proprietary file handling API in their code.

Such different methodology creates the interoperability issue. Ganesh should handle the files in a similar way as Condor because most of the environments follow the same mechanism and it does not overload the application developers.

C. Scenario 3: File transfers

Applications like Distributed Disaster Management System, need to transfer input, intermediate and output files from machine to machine. Let's consider a case as shown in Fig 8. All the jobs except the Job C are scheduled on the Ganesh environment. Job C is on scheduled on DAGMan-condor.

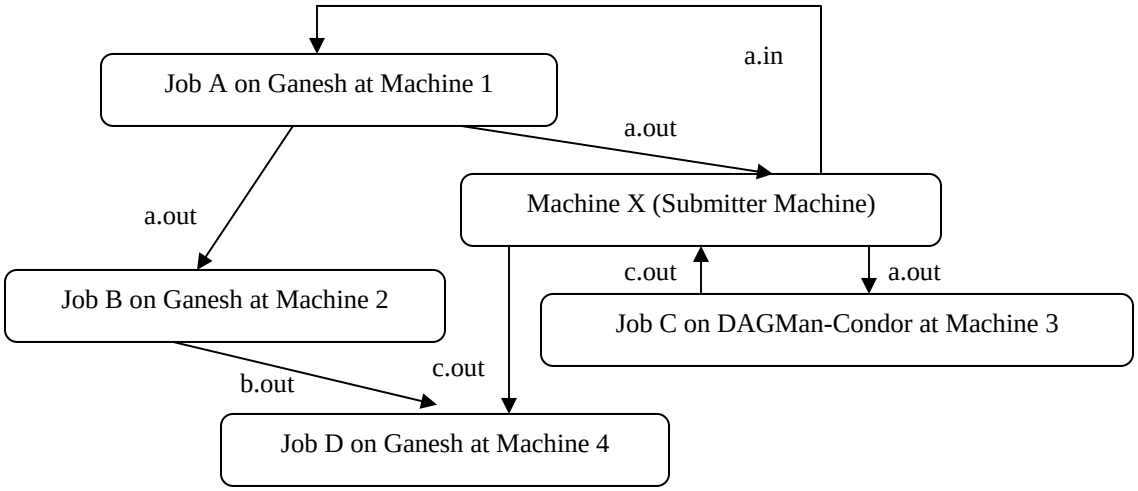

Fig 8: Jobs on different execution environment on different machines

Both of the environments handle the file transfer mechanism differently. Ganesh can transfer output files directly from Machine 1 to Machine 2 and Machine 2 to Machine 4 as it supports remote to remote file transfer, But for executing the Job B on DAGMan-Condor, the output of Job A i.e. a.out is first to be transferred on the Machine X then Condor will transfer it from Machine X to Machine 3. The output file c. out it will send back to the Machine X. This output file will be provided to Job D by again transferring the file from Machine X to Machine 4. This is because, Condor does not support remote to remote file transfer. It achieve this by first transfer the file from remote to submitter machine then from submitter to another remote machine. This creates the interoperability problem when jobs are scheduled under different execution environments. It also creates the network overhead of two hops and is inefficient particularly when file sizes are huge.

GANGA takes care of this while converting Ganesh Job into Condor Job by breaking a single remote to remote file transfer into two i.e. remote to submitter

machine then submitter machine to remote and vice-verse However it does not make use of the advance feature of Ganesh scheduler. We suggest, Condor should also support remote-to-remote file transfer to efficiently support such applications.

D. Scenario 4: Standard output and error files location

Standard output and error files are required to monitor the status of the jobs that are submitted. In a Grid scenario, the machines through which the jobs are submitted and the machines through which the jobs are monitored, may not be the same. Both the environments keep the standard output and error files at different locations that create the interoperability problem. In Ganesh, these files are transferred on a predefined machine having web server which can be accessed through any standard web browser. In Condor, these files are transferred back to the submitting machine, which restricts the roaming facility or compromise security. It creates the interoperability issue as user need to know different mechanism to access the standard output and standard error. Ideally, there should be a provision in Condor to transfer these files on any machine in the pool and not only on the machine through which the job is submitted. Till that, scheduler handles this as it knows the machine from where the jobs are being submitted However for interoperability, we address this interoperability issue by storing these files in GANGA repository. Any of the environment can retrieve these files and display on its own. However, we suggest that other execution environment should also support the sharing of standard output and error files similar to Ganesh.

We are additionally studying the GridAnt and Karajan [48] to address their interoperability with Ganesh. Karajan uses complete syntax (e.g., “For Loop” for repetitive task management) which is similar to that followed by Ganesh (it uses “Loop Count”). Ganesh has additional option for handling the repetition through “Exit Code”. It enables application to have run time dynamic repetition control. Some of GANGA’s current services are web based and are tested on Windows with IIS as well as Linux servers with Apache. OpenSSL is used for security management [30]. We are in process of using GSI [15] along with Globus certificates on Linux [14] and java certificates on Windows [39] for authentication and authorization.

IV. Conclusion and Future Scope

Interoperability among various environments is very useful as each environment has specific advantages. Furthermore, users will continue to develop applications on their most suited environment. . However, they will also be in the position to take advantage of other environments. The meta schedulers and workflow schedulers will be able to use all the resources available with all the environments. The registry of the resource information is useful in many ways. Similarly, registry of application will also be useful

in multi dimensions. It will become mandatory in future especially for virtualization (single operational umbrella) for a community grid or across community grid.

Currently GXML is used for interoperable information management which will become JSDL compliant once JSDL is frozen. Our model supports DAG information also, which is not part of JSDL. Repetition is also supported to certain extent. WFM group is in process of working out specification for work flow. We shall extend our mechanism to incorporate recommendation on finalization of WFM specifications. GANGA provides the mechanism for interoperability among the different execution environment. As Ganesh is our product, we have extended its scheduler and incorporated the recommendation given by the GANGA for interoperability. Till that other execution environments starts the support for interoperability, we anticipate there is a requirement for interoperability controllers among the different execution environments.

Acknowledgement

We acknowledge Mr. Hardik Dave, Mr. Hiren Gajjar, Mr. Shamit Mankad, Ms Anvi Shah of LD college of Engineering and Mrs. Mayuri Sharma, for their implementation help.

Reference:

[1] I. Foster, C. Kesselman, S. Tuecke “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, International Journal of Supercomputer Applications and High Performance Computing, 2001.

[2] I. Foster, C. Kesselman, J. Nick, S. Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. http://www. globus.org/research/papers/ogsa.pdf.

[3] I. Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

[4] Buyya R, Abramson D, and Giddy J “Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid”, Proc. 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia'2000), Beijing, China. IEEE Computer Society Press, USA, 2000.

[5] T. Haupt, E. Akarsu, G. Fox and W Furmanski, “Web Based Metacomputing”, Special Issue on Metacomputing, Future Generation Computer Systems, North Holland 1999.

[6] Bester, J., Foster, I., Kesselman, C., Tedesco, J., and Tuecke, S., “GASS: A Data Movement and Access Service for Wide Area Computing Systems”, *Sixth Workshop on I/O in Parallel and Distributed Systems*, May 5, 1999.

[7] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S., “A Resource Management Architecture for Metacomputing Systems”, *Proc. IPPS/SPDP ’98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[8] Henderson, R. and Tweten, D., “Portable Batch System: External Reference Specification”, 1996.
[9] IBM, “Using and Administering IBM LoadLeveler, Release 3.0”, IBM CorporationSC23-3989, 1996.
[10] Litzkow, M., Livny, M., and Mutka, M., “Condor – A Hunter of Idle Workstations”, *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, pp. 104-111.
[11] Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane, "A Java Commodity Grid Kit," Concurrency and Computation: Practice and Experience, vol. 13, no. 8-9, pp. 643-662, 2001, http:/www.cogkits.org/.
[12] [DAGMan] http://www.cs.wisc.edu/condor/dagman
[13] The Globus Resource Specification Language RSL v1.0 http://www-fp.globus.org/gram/rsl_spec1.html
[14] Globus Simple CA directory - ftp://ftp.globus.org/pub/gsi/simple_ca
[15] Globus Security Policy and Implementation. 1997, http://www.globus.org/security/.
[16] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling, “Open Grid Services Infrastructure (OGSI) Version 1.0,” Global Grid Forum Draft
[17] Haresh S. Bhatt, B. K. Singh, A. K. Aggarwal, A Generalized Environment for Distributed Image Processing, HPCS-2002, Canada
[18] Haresh S. Bhatt, Aggarwal A.K., web enabled client-server model for development environment of distributed image processing, Proceedings of international conference on meta computing, GRID-2000, pp 135- 145
[19] Haresh S. Bhatt, Singh B.K., Aggarwal A.K., Application centric load balancing based on hybrid model, International conference for advance computing and communication ADCOM-2001, organised by IEEE and ACS India, Dec. 2001, Page(s): 231-238
[20] Jensen K, “AN introduction to the Theroritical Aspects of Coloured Petri Nets. Lecture notes in Computer Science, Vol. 803, Springer-Verlag, Berlin Heidelberg New York (1994) 230-272.
[21] GRRP Czajkowski, K., Fitzzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. in *Tenth IEEE International Symposium on High Performance Distributed Computing(HPDC-10)*, (2001).
[22] NAKADA, H., SATO, M., AND SEKIGUCHI, S. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems 15*, 5-6 (1999), 649–658.
[23] OpenPBS http://www.openpbs.org/
[24] UNICORE. http://www.unicore.de/
[25] OMG Specifications for UML, http://www.omg.org/technology/documents/formal/uml.htm
[26] I. Jacobson, Grady Booch, J. Rumbaugh, “The Unified Software development Process”,Addison-Wesley,2000.
[27] Haresh S. Bhatt, RM Patel, Hitesh Kotecha, VH Patel, Arup Dasgupta, “GANESH: Grid Application maNagement and Enhanced ScHeduling”, Unpublished.
[28] K. Amin, M. Hategan, G. von Laszewski, N. J. Zaluzec, S. Hampton, and A. Rossi, “GridAnt: A Client-Controllable Grid Workflow System,” in 37th Hawai’I

International Conference on System Science, Island of Hawaii, Big Island, 5-8 Jan. 2004. http://www.mcs.anl.gov/_gregor/papers/vonLaszewski--gridant-hics.pdf

[29] Andreas Savva, Ali Anjomshoaa, Fred Brisard, R Lee Cook, Donal K. Fellows, An Ly, Stephen McGough, Darren Pulsipher, "Job Submission Description Language (JSDL), Specification", 17 May 2004, http://www.ggf.org/Meetings/GGF11/Documents/draft-ggf-jsdl-spec.pdf

[30] OpenSSL Project http://www.openssl.org

[31] Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C., "Grid Information Services for Distributed Resource Sharing", 2001.

[32] GRIP Czajkowski, K., Fitzzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. in *Tenth IEEE International Symposium on High Performance Distributed Computing(HPDC-10)*, (2001).

[33] ***http://www.isro.org/***

[34] "Ant – a Java-based Build Tool," , http://ant.apache.org/

[35] M. Neary, B. Christiansen, P. Cappello, K. Schauser, Javelin: Parallel computing on the internet, Future Generation Computer Systems, Vol. 15, (1999), 659-674.

[36] GRIS Czajkowski, K., Fitzzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. in *Tenth IEEE International Symposium on High Performance Distributed Computing(HPDC-10)*, (2001).

[37] FraunHofer Resource Grid: XML schema of the Grid Job Definition Language, http://www.fhrg.fhg.de/de/fhrg/schemas/gadl/gidl.xsd

[38] G. von Laszewski and P. Wagstrom, Tools and Environments for Parallel and Distributed Computing, ser. Series on Parallel and Distributed Computing. Wiley, 2004, ch. Gestalt of the Grid, pp. 149–187. http://www.mcs.anl.gov/_gregor/papers/vonLaszewski--gestalt.pdf

[39] Vladimir Silva, Manage X.509 certificates in your grid with Java Certificate Services, http://www-106.ibm.com/developerworks/grid/library/gr-jsc/?ca=dgr-lnxw06ManageX.509

[40] Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S., Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *10th International Symposium on High Performance Distributed Computing*, (2001), IEEE Press, 55-66

[41] D. Jackson, "Silver Metascheduler Overview,",http://supercluster.org/projects/silver

[42] OpenLDAP http://www.openldap.org/

[43] Ian Foster and Carl Kesselman (eds), "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, July 1998. ISBN 1-55860-475-8.

[44] Grid-enabled PBS: the PBS-Globus Interface, http://www.globus.org/retreat00/presentations/g_PBS-abstract.pdf

[45] https://forge.gridforum.org/projects/jsdl-wg/

[46] https://forge.gridforum.org/projects/wfm-wg/

[47] Andreas Hoheisel, "User Tools and Languages for Graph-based Grid Workflows", GGF-10. http://www.ggf.org/

[48] http://www-unix.globus.org/cog/manual-cog2.pdf

[49] http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

Dr Haresh S Bhatt is a senior scientist working in Indian Space Research Organization since 1984. He was actively involved in all the IRS (Indian Remote Sensing Satellite) programs till 1997. His remarkable contribution in automatic cloud covers estimation for IRS and in solving IRS-1C on-board sensor calibration was internationally acclaimed and was first of its kind. He has number of publications in International conferences and journals. Currently, he is working in the field of handshaking of Advanced Communication Satellites and state of art Computing Technology. He has been member of program committee and referee in

Mr. Dharmesh Bhansali is B.E. (Computers) and MBA. He is working with Space Applications Centre, Indian Space Research Organization since 1997. He has been involved in various project development work related to SACNET, EDUSAT and GSAT 4. His main area of interest is networking, software engineering, web-based application development, e-learning and Grid Computing. He has several publications in national and International conferences and journals.

Ms. Sonal Shah is Graduate in Electronics and Communications. Working as Scientist/Engineer in Space Applications Centre in the field of Networking and Satellite Communications.

Mr. Praful Patel is M.Sc. (Physics) and working with SAC, ISRO since 1984. His areas of interest are Networking and Internet & Security.

Mr. V H Patel is presently Head of Department of Networks Division at Space Applications Centre, Indian Space Research Organisation, Ahmedabad, India. He is B.Engg. in Elect. Communications Engg. from Indian Institute of Science, India. His experience includes Development in Video Subsystems and Image Processing Systems. He has vast experience in Computer Engg., Computer Architecture and Networking, Space Technology and Remote Sensing Technology.

Mr. Dasgupta is an M.E. in Electrical Communications Engineering and has been working in the Space Applications Centre since November 1970. He worked in the Satellite Instructional Television Experiment (SITE). From 1976 till 2000 he was involved in the management of applications programmes for several remote sensing satellites including Bhaskara and IRS as well as development of Image and Information Processing systems. He is currently the Deputy Director SATCOM and IT Applications Area. He is a recipient of the Astronautical Society of India Award for Space Science and Applications for the year 2000. He is a Senior Member of the Institute of Electrical and Electronic Engineers, Inc, USA and the Chairman of the IEEE, Gujarat Section, 2003-04 and Fellow of the Institution of Electronics and Telecommunications Engineers, India.