www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 4 April, 2013 Page No. 853-863

Grid computing Issues, Challenges, Need And Practice

Ayushi Pathak[1] and Nisha Kaushik[2]
Computer Science Department, Dronacharya College of Engineering
Greater Noida, Uttar Pradesh
India
ayushi112233@yahoo.in; thomson99kaushik@gmail.com

ABSTRACT- In the last few decades with the rapid improvement in the field of computer technology, data and communication the concept of "GRID COMPUTING" have become more prominent. The popularity of the Internet and availability of high-speed networks have gradually changed the way we do computing. These technologies have enabled the cooperative use of a wide variety of geographically distributed resources as a single more powerful computer. This new method of pooling resources for solving large-scale problems is called as grid computing.

The concept of grid computing have not only evolved in terms of topology but have also included wide spread data communication by the means of resource-sharing, however it is still lacking in terms of trust and reliability. Since a major ratio of people are communicating and sharing data which thereby decreases the trust of the users. It has not able to convince the user fully. This paper describes the concepts underlying grid computing and the trust model on which the reliability factor totally depends.

KEYWORDS-Grid computing, problems, need, challenges, applications.

I. INTRODUCTION

In the last few years there has been a rapid exponential increase in computer processing power, data storage and communication. But still there are many different and computation intensive problems, which cannot be solved by supercomputers. These problems can only be met with a vast variety of heterogeneous resources. The use and increased popularity of the Internet and the availability of high-speed networks have gradually changed the way we do computing. These technologies have enabled the cooperative use of a wide variety of geographically distributed resources as a single more powerful computer. This new method of pooling resource and solving large-scale problems called grid computing. Grid computing is a form of distributed computing it involves coordinating and sharing computing, application, data and storage or network resources across dynamic and geographically dispersed organization. Grid technologies promise to change the way Organizations tackle complex computational problems. The vision of grid computing was to allow access to computer based resources (from CPU cycles to data servers) in the samemanner as real world utilities. This gave rise to the idea of Virtual Organizations (VOs). Through the creation of VOs, it was possible to access all resources as though all resources were owned by a single organization.

Two key outcomes exist in grids: the Open Grid Service Architecture (OGSA) and the Globus Toolkit.

II. GRID CHARCTERSTICS-

These characteristics may be described as follows:

Large scale: It must be able to deal with a number of resources ranging from just a few to millions. This raises the very serious problem of avoiding potential performance degradation as the grid size increases.

*Geographical distribution:*It's resources may be located at different places.

Heterogeneity: It hosts both s/w and h/w resources that can be very varied ranging from data, files, software components or programs to sensors, VOL. 2, NO.5, MAY 2012 ISSN 2222-9833 ARPN Journal of Systems and Software ©2009-2012 AJSS Journal. All rights reserved http://www.scientificjournals.org190 scientific instruments, display devices, personal digital organizers, computers, super-computers and networks.

Resource sharing: resources in a grid belong to many different organizations that allow other organizations (i.e. users) to use or access them.

*Multiple administrations:*each and every organization may establish different security and administrative policies under which their owned resources can be accessed and used. As a result, the already challenging network security problem is complicated even more with the need of taking into account all different policies.

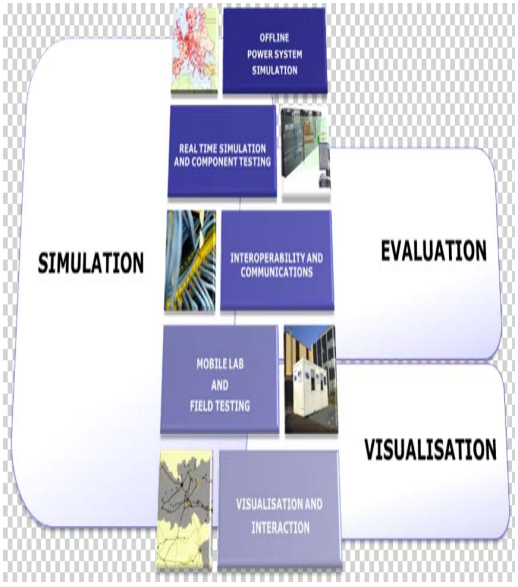
Resource coordination: resources in a grid must be coordinated in order to provide aggregated computing capabilities.

Transparent access: It should be seen as a single virtual computer.

Dependable access: It must assure that the delivery of services under established Quality of Service (QoS) requirements. The need for dependable service is fundamental since users require assurances that they will receive predictable and often high levels of performance.

Consistent access: It must be built with standard services, protocols and inter-faces thus hiding the heterogeneity of the resources while allowing its scalability. Without such standards, application development and pervasive use would not be possible.

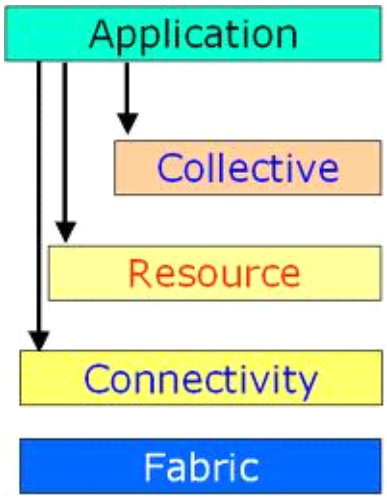
Pervasive access: It must grant access to available resources by adapting to a dynamic environment in which resource failure is common place. This does not imply that resources are everywhere or universally available but that the grid must tailor its behavior as to extract the maximum performance from the available resources.



III. GRID ARCHITECTURE-

Computational grids have to be designed so as to serve different communities with varying characteristics and requirements. Because of this reason we cannot have a uniform single architecture. But in general we can identify basic services that almost all the grids will provide although different grids will use different approaches for the realization of these services. This description of grid architecture does not provide a complete enumeration of all the required protocols and services but it identifies the requirements for general class of components.

This architecture organizes the components into layers as shown in Figure.



The Layers of the grid are as follows*:*

• *Fabric Layer*

This layer provides the resources, which could comprise computers (PCs running Windows NT or UNIX), storage devices and databases. The resource could also be a logical entity such as a distributed file system or computer pool. Excellent fabric functionality could mean that sophisticated sharing operations can be accomplished. For this, it should support enquiry mechanisms to discover their state, structure and capabilities. It should also have resource management mechanisms that provide some control of delivered quality of service.

• *Connectivity Layer*

This layer consists of the core communication and authentication protocols required for transactions. Communication protocols enable the exchange of data between fabric layer resources. Authentication protocols provide secure cryptographic mechanisms for identifications of users and resources. For communication transport, naming and routing are required. These protocols can be drawn from TCP/IP protocol stack.

• *Resource Layer*

This layer builds on the Connectivity layer communication and authentication protocols to define Application Program Interfaces (API) and Software Development Kit (SDK) for secure negotiation, initiation, monitoring, control, accounting and payment of sharing operations. The protocols, which the resource layers implement to achieve the above functionality are implemented with the functions provided by fabric layer. Resource layer protocols can be distinguished primarily into two classes, which are *Information Protocols* and *Management Protocols.*

1. *Information Protocol* -This protocol is used to obtain the necessary information about the structure and the state of the resource

2. *Management Protocol* -In order to negotiate the access to the shared resources this protocol is used.

• *Collective Layer*

This layer is different from the resource layer in the sense, while resource layer concentrates on interactions with single resource. This layer helps in coordinating multiple resources. Its tasks can be varied like Directory Services, Co-allocation and scheduling, monitoring, diagnostic services, and software discovery services.

• Application Layer

This layer consists of the user applications and programs and which call upon another layer.

**** Trust and Reliability***

The primary goal of grid computing is to encourage the interaction between every domain which is required for the user for communication without losing the control over own resources

which would be needed for the communication between the client and the domain and ensuring that the communication between domain-to-domain doesn't hampers the confidentiality of the user within the scope of the grid.

To achieve this the concept of "trust and reliability" must be addressed in order to make the grid computing more appealing. The concept of trust is divided into two categories: - identity trust and behavior trust.

Identity trust involves the verification of authenticity of every single entity connected to the network and further using various techniques in order to authenticate the entity and thereby increasing its reliability.

Behavioral trust covers he broader spectrum by checking the overall "trustworthiness" of the entity connected to the network and its further intentions.

In this paper we deal with the broader spectrum of trust and reliability aspect by concentrating of trust model based on behavioral trust.

The definition of trust that we will use in this paper is as follows:

Trust is the firm belief in the competence of an entity to act as expected such that this firm belief is not a fixed value associated with the entity but rather it is subject to the entity's behavior and applies only within a specific context at a given time.
With reference to [Mis96, GrS00, AbH00].

IV. Trust Model

4.1.*1* *Overview*

The trust model primarily comprises of a grid which is divided into several GRID DOMAINS (GD). there exists two virtual domains with each GD namely resource domain (RC) which signifies the resources within the grid domain (GD) and a client domain (CD) to signify the clients within the grid domain (GD).Firstly, the resources and clients within a grid domain inherit the parameters of resource domain (RD) and client domain (CD) they are associated with. Secondly, a value in the trust level table is modified by a new trust level value based on the computation on significant amount of transactional data. Third, limiting the number of contexts reduces the fragmentation of trust management space. In our study the contexts are limited to primary services such as printing, storage and computing.

- *Direct and Reputation Trust Weights*

Weights α and β are denoted as direct and reputation trust relationships respectively ranging between 0 to 1. Assigning values to these weights is up to the individual domain. For example: (a) Di might trust business partners or allies more than other domains. Therefore, $D_i$will give more weight to its business-partners and allies as recommenders or as domains to directly interact with, (b) D_i might have a policy stating that Di will only accept recommendations from domains whom D_i has a direct trust relationship with. I.e. D_i will assign a value of zero to recommenders whom it does not have direct interaction with.

Description of required trust levels

TRUST LEVELS	DESCRIPTION
A	VERY LOW TRUST LEVEL
B	LOW TRUST LEVEL
C	MEDIUM TRUST LEVEL
D	HIGH TRUST LEVEL
E	VERY HIGH TRUST LEVEL
F	EXTREMELY HIGH TRUST LEVEL

- *Decay Function*

The matter of fact is that trust decays with time. For instance, if D_i has not interacted with D_j for five years, then the TL between them today is likely to be weaker unless they have interacted since. In our trust model we have introduced the concept of decay functionto reflect the drop of trust within the model. Here the decay function is $\Upsilon(t\text{-}t_{ij}, c)$, we look at how old (in terms of time) is the

TL that resulted from the last transaction between D_i and D_j. Each domain might have different decay function and might be looking at other factors that accelerate or decelerate the TL decay.

- *Evaluating Trust Function*

TL resulting from a direct trust relationship means D_i is directly involved in a transaction with D_j. There are two (RTLs) *required trust levels* which are from client side and resource side. For example if two domains D_i and D_j are directly involved in a transaction so there may be a case where D_i might not want the mapping of its application onto the resources that are owned or managed by the domain it does not trust. Similar situation can stand for resource producer side where D_j does not want its resources being utilized by application which is not trustable. Therefore each D_i and D_j will specify a RTL which should not be violated. D_i evaluates the direct trust relationship with D_j based on the behavior of D_j Violation of RTL can lead to many abuses which are as follows:-
(a) using more resources than requested leading to improper utilization of resources, (b) leaving behind unused data and without doing "garbage collection" after using the resources, (c) going out of allocated boundary, and (d) instantiating tasks they are not required to instantiate. Such intrusions and violations can be detected by *audit data* [HaC92].

- *Trust Inheritance*

There is a *member weight* associated with every entity to indicate if the entity is a new, recent, or an old member with its domain and it is up to the individual domain to decide what constitutes an entity to fall in one of these *member weights*.

V. Problem Description

Scientists often seek specific data products, which can be obtained by composing available application components and executing them on the Grid.

As an example, suppose that the user's goal is to obtain a frequency spectrum of a signal S from instrument Y and time frame X, placing the results in location L. In addition, the user would like the results of any intermediate filtering steps performed to be available in location I, perhaps to check the filter results for unusual phenomena or perhaps to extract some salient features to the metadata of the final results. The process of mapping this type of user request into jobs to be executed in a Grid environment can be decomposed into *two steps*, as shown in Figure 1.

1. *Generating an abstract workflow:* Selecting and configuring application components to form an abstract workflow. The application components are selected by looking at the specification of their capabilities and checking if they can generate the desired data products. They are configured by assigning input files that exist or that can be generated by other application components. The abstract workflow specifies the order in which the components must be executed. More specifically, the following steps need to be performed:

a. Find which application components generate the desired data products, in our example a frequency spectrum of the desired characteristics. Let one such component be C n. Find which inputs that component takes, check if any inputs are available and if so let the corresponding files be I1 . . . Ij . If any input is required in formats that are not already available then find application components that can produce that input, let one such component be C n − 1. This process is iterated until the desired result can be generated from a composition of available components that can operate over available input data, namely C1 . . . C n and I1 . . . I m respectively.

b. Formulate the workflow which specifies the order of execution of the components C1 . . . C n. This is what we call an abstract workflow. Please note that at this level the components and files are referred to by their logical names which uniquely identify the component in terms of their functionality and the data files in terms of their content, but a single logical name can correspond

to many actual executable and physical data files in different locations.

2. *Generating concrete workflow :* Selecting specific resources, files, and additional jobs required to form a concrete workflow that can be executed in the grid environment. Each component in the abstract workflow is turned into an executable job by specifying the locations of the physical files of the component and data as well as the resources assigned to it in the execution environment. Additional jobs may be included in the concrete workflow. For example, jobs that transfer files to the appropriate locations where resources are available to execute the application components. More specifically, the following steps need to be performed:

a. Find the physical locations (i.e., physical files) of each component C1 . . . C n: C1-pf . . . C n-pf.

b. Check the computational requirements of C1-pf. . . . C n-p f and specify locations L1 . . . L n to execute them according to the required and available resources.

c. Determine the physical locations of the input data files I1 -pf . . . I m-p f, select locations that seem more appropriate given L1 . . . L n.

d. Augment the workflow description to include jobs K1 . . . Km + n to move component and input data files (C1-pf . . . C n-p f and I1-pf . . . I m-p f) to the appropriate target locations L1 . . . L n. Although Grid middleware allows for discovery of the available resources and of the locations of the replicated data, users are currently responsible for carrying out all of these steps manually. There are several important factors that make automating this process not only desirable but necessary:

• Usability: Users are required to have extensive knowledge of the Grid computing environment and its middleware functions. For example, the user needs to understand how to query an information service such as the Monitoring and Discovery Service (MDS), to find the available and appropriate computational resources for the computational requirements of a component (step 2b).

The user also needs to query the Replica Location Service (RLS) [13] to find the physical locations of the data (step 2c).

• Complexity*:* In addition to requiring scientists to become Grid-enabled users, the process may be complex and time consuming. Notice that in each step, the user makes choices when alternative application components, files, or locations are available. The user may reach a dead end where no solution can be found, which would require backtracking to undo some previous choice. Many different interdependencies may occur among components, and as a result it may even be hard to determine which choice to change and what would be a better option that leads to a feasible solution.

• Solution cost*:* Lower cost solutions are highly desirable in light of the high cost of some of the computations and the user's limitations in terms of resource access. Because finding any feasible solution is already time consuming, users are unlikely to explore alternative workflows that may reduce execution cost.

• Global cost*:* Because many users are competing for resources, minimizing cost within a community or a virtual organization (VO) is desirable. This requires reasoning about individual user's choices in light of other user's choices, such as possible common jobs that could be included across user's workflows and executed only once. While addressing the first three points would enable wider accessibility of the Grid to users, the last point of handling global cost simply cannot be handled by individual users and will likely need to be addressed at the architecture level. In addition, there are many policies that limit user's access to resources, and that needs to be taken into account in order to accommodate as many users as possible while they are contending for limited resources.

An additional issue is the reliability of execution. In today's Grid framework, when the execution of a job fails the recovery consists of resubmitting that job for execution on the same resources. (In Figure 1 this is shown as the "retry".) However, it is also desirable to be able to choose a different set of resources when tasks fail. This process needs to be performed at the abstract workflow level. Currently, there is no mechanism for opportunistically redoing the remaining tasks in the workflow to adapt to the dynamic situation of the environment. Moreover, if any job fails repeatedly it would be desirable for the system to assign an alternative component to achieve the same overall user goals. This would need to be performed at the application level, where there is an understanding of how different application components relate to each other. In Table 1 we describe three different levels of abstraction that a user can use to specify a workflow.

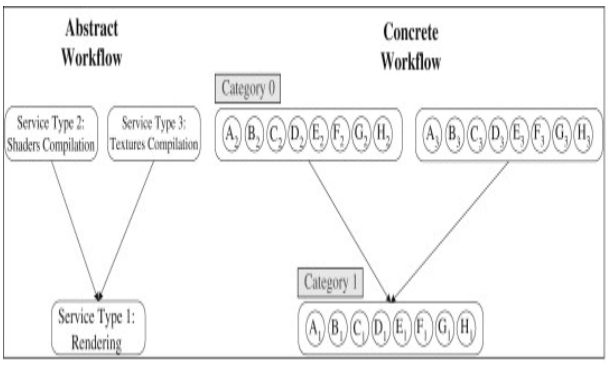
Table 1. Levels of abstraction used to describe workflows.

	Concrete workflow domain	Abstract workflow domain	Application domain
specification	Gridftp Host1://home/filea Host2://home/file1/user/ Local/bin/fft-i/home/file/	Fftfilea	Frequency spectrum of a signal s from instrument y and time frame x
Specification details	Resource level physical file execution	Logical file name, logical component name	Application specific metadata

The lowest level (concrete workflow) the user needs to specify explicit data movements and the exact executable and resources to be used. At the abstract workflow level the user needs only specify the workflow using logical files and logical component names. Finally at the top level, the application level, the user needs to specify only the metadata describing the desired data products.

In Section 3 we describe the implementation of a Concrete Workflow Generator (CWG). CWG performs the mapping from an abstract workflow to a concrete workflow. The system automatically locates physical locations for both components and data;finds appropriate resources to execute the components and generates an executable workflow of jobs that can be submitted to the Grid. Although this implementation isolates the user from many details about the Grid infrastructure, it still requires the user to spell out all the components and input files required. In addition, whenever several alternatives are possible (e.g., alternative physical files, alternative resources) it makes a random choice, so the final result is a feasible solution and not necessarily a low-cost one.

In Section 4, we describe the implementation of an Abstract & Concrete Workflow Generator (ACWG) which only requires from users an abstract description of the desired data products in terms of application specific metadata. The approach we used was to exploit Artificial Intelligence planning techniques that explore the solution space with search algorithms guided with informed heuristics.



VI. Application Experiences

As part of this work we have developed a configurable system, Pegasus (Planning for execution in Grids) and integrated it into Chimera. In Chimera the user first specifies the abstract descriptions of a component (the arguments it takes etc., the number of input and output files), which is defined using the Chimera Virtual Data Language (VDL). The language also includes derivations which are invocations of a particular component and contain logical file names (LFN) and parameters used to run that component. The derivations are used to construct abstract workflows. In the Chimera-driven configuration Pegasus receives an abstract workflow description from Chimera and uses CWG to produce a concrete workflow. Pegasus then submits the concrete workflow to DAG Man for execution and monitors the jobs described in the concrete workflow. We have used this configuration to map CMS workflows onto the Grid.

6.1. Applying CWG to CMS

The Compact Muon Solenoid (CMS) is a multipurpose particle physics detector currently being constructed at the European Center for Nuclear Research (CERN) in Geneva, Switzerland. When it begins operation in 2007, the CMS detector is expected to record data, produced by high-energy proton-proton collisions occurring within CERN's Large Hadron Collider (LHC), at a rate of 100 MB/s. After the data is recorded, it will be passed through various filter stages which transform and reduce the data into formats which are more easily analyzed by physicists. In order to better understand the response of the detector to different input signals, large scale, Monte Carlo simulations are performed which typically involve several different computational stages. These simulations are long-running, parallel, multi-stage processes that are ideally suited for Grid computation. Typically, a single workflow creates approximately 1 GB of data and requires 10 to 20 CPU/hours depending on the type of simulation. A typical production run may include thousands of workflows.

A variety of different use-cases exist for simulated CMS data production. One of the simpler use-cases is known as an *n-tuple* only production which consists of five stage computational pipeline shown in Figure 4.

The first is a generation stage that simulates the underlying physics of each event. The second stage is a simulation stage that models the CMS detector's response to the events created in the generation stage. The third stage, or formatting stage, copies the simulated detector data into an object-oriented database (OODB). The next stage, or reconstruction stage, transforms the data in the database, producing a "picture" of what a physicist would "see" as if the simulated data were actual data recorded by the experimental apparatus. The final stage, an analysis stage, selects user-specific information from the database and creates a convenient, easy to use file that can be analyzed by a researching physicist.

In an n-tuple only production, the last file, an n - tuple, is the only important piece of data and the intermediate data may be discarded. However, the log files for the intermediate data products are needed for quality assurance validation. Several small scale tests of CMS n-tuple only production pipelines have been successfully performed using CWG. CWG was also used for a large scale test which involved two- stage pipelines (generation

and simulation). CWG scheduled this work to be performed at a University of Florida computing cluster consisting of 25 dual- processor Pentium (1 GHz) machines. Over the course of 7 days, 678 jobs of 250 events each were submitted using CWG. From these jobs, 167,500 events were successfully produced using approximately 350 CPU/days of computing power and producing approximately 200 GB of simulated data.

6.2. Applying ACWG to the LIGO Pulsar Search

LIGO (Laser Interferometer Gravitational-Wave Observatory, www.ligo.caltech.edu) is a distributed network of interferometers whose mission is to detect and measure gravitational waves predicted by general relativity, Einstein's theory of gravity. One well-studied source of gravitational waves is the motion of dense, massive astrophysical objects such as neutron stars or black holes. Other signals may come from supernova explosions, quakes in neutron stars, and pulsars. Gravitational waves interact extremely weakly with matter, and the measurable effects produced in terrestrial instruments by their passage are expected to be miniscule. In order to establish a confident detection or measurement, a large amount of auxiliary data will be acquired (including data from seismometers, microphones, etc.) and analyzed (for example, to eliminate noise) along with the strain signal that measures the passage of gravitational waves.

The raw data collected during experiments is a collection of continuous time series at various sample rates. The amount of data that will be acquired and cataloged each year is on the order of tens to hundreds of terabytes. The gravitational wave strain channel is less than 1% of all data collected. Analysis on the data is performed in both time and frequency domains.

FIG: -The LIGO pulsar search

EXTRACT FREQUENCY-----CONSTRUCT IMAGE-----FINAL CANDIDATE--STORE EVENT DB

Requirements are to be able to perform single channel analysis over a long period of time as well as multi-channel analysis over a short time period.

To investigate the capability of ACWG to generate complex, metadata driven workflows, we integrated ACWG into Pegasus and applied it to a specific LIGO analysis, the pulsar search. In the ACWG-driven configuration, Pegasus is driven by the metadata of the search, which can be implemented as a pipeline depicted in Figure 5. The first element in Figure 5 is data archiving as instrumental data is stored into an archive. Next, since, the raw data comes from the instrument as short (16 second duration) Frames (a data structure used in the gravitational wave community) with all the channels, some processing geared towards the removal (cleaning) of certain instrumental signatures needs to be done. For example, naturally occurring seismic vibration can be subtracted from the data using the channels from the sensitive seismometer that is part of the LIGO data stream. For the pulsar search, the gravitational wave strain channel is extracted. The pulsar search is conducted in the frequency domain; thus Fourier Transforms are performed on the long duration time frames to produce datasets known as *Short Fourier Transforms (SFTs)*. Since the pulsars are expected to be found in a small frequency range, the frequency interval of interest is extracted from the STFs. The resulting power spectra are used to build the time-frequency image, which is analyzed for the presence of pulsar signatures. If a candidate signal with a good signal to noise ratio is found, it is placed in LIGO's event database.

VII. **Future Directions**

Finding good abstract and concrete workflows involves a wide range of issues that have been investigated in Artificial Intelligence planning, including hierarchical planning, temporal reasoning and scheduling, reasoning about resources, planning under uncertainty and interleaving planning and execution. In the near future we plan to evaluate approaches such as plan reuse and planning under uncertainty to increase the level of ACWG's performance and sophistication.

We also plan to investigate the applicability of our approach to service-level composition. In this section we describe some of our ideas.

7.1. Solution Reuse

One important research area that is likely to be effective for this problem is the reuse of solutions that were previously computed. Case-based planning is a powerful technique to retrieve and modify existing plans that need slight changes to be adapted to the current situation. These approaches have potential for ACWG because the network topology and resource characteristics are likely to be fairly stable and therefore high-quality solutions, which may take time to generate from first principles, will be good starting points for similar problems in the future.

7.2. Fault Avoidance

In the simple case, the planner creates a plan that is subsequently executed without a hitch. Often, however, runtime failures may result in the need to repair the plan during its execution. Planning systems can also design plans that either reduce the risk of execution failure or are more likely to be salvageable when failures take place. They can explicitly reason about the risks during planning and searching for reliable plans, possibly including conditional branches in their execution. Some planners delay building parts of the plan until execution, in order to maintain a lower commitment to certain actions until key information becomes available. These approaches are likely to have high impact in the Grid computing domain, since its decentralized nature means many factors are beyond the control of the planning agent. However current techniques for handling uncertainty have high complexity, and are not useable when more than a few potential failure points need to be considered.

7.3. Relevance to Open Grid Services Architecture

Although much work needs to be done in the area of workflow generation, we believe that the framework we designed is a good foundation for developing ever more sophisticated techniques, which will take into account an ever greater amount of information about the applications and the execution environment. Above Figure illustrates additional sources of information that we would like to integrate in the future within the workflow generation process. At the application level, we can describe the application components as services, which would facilitate the integration of our work with the new Open Grid Services Architecture (OGSA). These services can be composed into new more sophisticated services. Although OGSA provides a syntactic description of the services (via WSDL) it does not assign any semantic meaning to them. We propose to augment service-based component descriptions by developing ontologies of application components and data, which will describe the service behavior and add semantic meaning to the service interactions. Ontologies will allow us to generate abstract workflows more flexibly from user requirements that may be partially complete or specified at higher levels of abstraction than the current service descriptions. Additional information provided by performance models of the services can guide the initial composition.

We also see ontologies playing a very important role in generating concrete workflows. Ontologies of Grid resources would allow the system to evaluate the suitability of given resources to

provide a particular application service instance. The resources that are to be allocated to various tasks can often be characterized in a domain-independent way by how they are used.

For example, a computer system becomes available again once a task has been completed but a user's allocation of time on a particular machine is permanently depleted. Ontologies of resources capture these qualities. Such ontologies, along with others which can capture computer system capabilities and job requirements, are key in building planning domains for ACWG quickly and reliably from generic components. However, there has been little work in this area of engineering planning domains.

7.4. Incorporating Policy Descriptions

In addition, in order to generate a feasible workflow, information such as policies governing the members of a Virtual Organization must be provided. For example, given an allotment of resources, the VO might decide to grant more resources to particular individuals. At the same time resources themselves need to provide information about the policies that they enforce both at the VO and user levels. The resources also have to provide information about their current state. Given a feasible solution, CWG also needs Application domain – the high energy physics experiment

to provide an optimal solution, considering policies in part but also the overall behavior of the application.

VIII. **Conclusions**

In this paper we addressed the issue of composing complex applications and mapping them on the Grid resources. We have identified two important steps that need to take place. The first step is to map application requirements in terms of desired data products to an abstract workflow that specifies what application components can generate the data. The second step maps the workflow onto Grid resources.

We described two mappings: *CWG*, which takes an abstract workflow, generates a random feasible solution and performs modest optimizations, and *ACWG* which can perform both steps. We have exploited and adapted AI planning techniques in ACWG to express the mapping problem using application-specific metadata. ACWG makes use of operator-based plan generation and combines local heuristics and a global measure to look for high-quality plans. We applied CWG to an important

References

[1]**.**Farag Azzedin and Muthucumaru Maheswaran**Evolving and Managing Trust in GridComputing Systems**[*], University of Manitoba and TR*Labs*Winnipeg, Manitoba, Canada.

[2]**.** Ewa Deelman1, James Blythe1, Yolanda Gil1, Carl Kesselman1, Gaurang Mehta1, Karan Vahi1, Kent Blackburn2, Albert Lazzarini2, Adam Arbree3, Richard Cavanaugh3 And Scott Koranda,**Mapping Abstract Complex Workflows onto Grid Environments**

1*Information Sciences Institute, University of Southern California, Marina Del Rey, CA 90292, US.*

[3]**.** Sunil Taneja*and Ashwani Kush†, **A Survey of Routing Protocols in Mobile Ad Hoc Networks**
International Journal of Innovation, Management and Technology, Vol. 1, No. 3, August 2010
ISSN: 2010-0248