Survey and Taxonomy of Grid Resource Management Systems

Chaitanya Kandagatla
University of Texas, Austin

Abstract

The resource management system is the central component of a grid system. This paper describes a *taxonomy for classifying* resource management architectures and presents the survey of resource management architectures of popular contemporary grid systems.

1. Introduction

Grid systems are inter connected collections of heterogeneous and geographically distributed resource harnessed together to satisfy various needs of the users. Resource Management is central component of a grid system. It involves managing resources in the system. Its basic responsibility is to accept requests from users, match user requests to available resources for which the user has access and schedule the matched resources. This paper presents a taxonomy to classify the resource management systems. The taxonomy is based on the architecture of the resource management system.

Rest of the paper is organized as follows section 2 discusses various issues about resource management in grid systems; section 3 describes the taxonomy of architecture of grid resource management systems; section 4 presents the survey of resource management in popular grid systems and classifies them according the developed taxonomy, this is followed by the conclusions.

2. Resource Management Systems

Resource management is a complex task involving security, fault tolerance along with scheduling. It is the manner in which resources are allocation, assigned, authenticated, authorized, assured, accounted, and audited. Resources include traditional resources like compute cycles, network bandwidth, space or a storage system and also services like data transfer, simulation etc.

2.1 Issues in Grid Resource Management Systems (GRMS)

Resource Management in Grid systems is made complex due to various factors like site autonomy, resource heterogeneity etc. Various factors contributing to this complexity are discussed below.

Grid resource management system should preserve *site autonomy*. Traditional resource management systems work under the assumption that they have complete control on the resource and thus can implement the mechanisms and policies needed for effective use of that resource. But in Grid systems resources are distributed across separate administrative domains this results in resource heterogeneity, differences in usage, scheduling policies, security mechanisms. Another resource management issue made complex due to the site autonomy is Co-allocation of the resources. Co-allocation is the problem of allocating resources in different sites to an application simultaneously.

Different administrative domains employ different local resource managements systems like NQE, LSF etc. A grid resource management system should be able to interface and interoperate with these local resource managements.

A resource management system for the grid should support such negotiation. In a grid system resources are added and removed dynamically. Different types of applications with different resource requirements are executed. Resource owners set there own resource usage policies and costs. This necessitates a need for negotiation between resource users and resource providers.

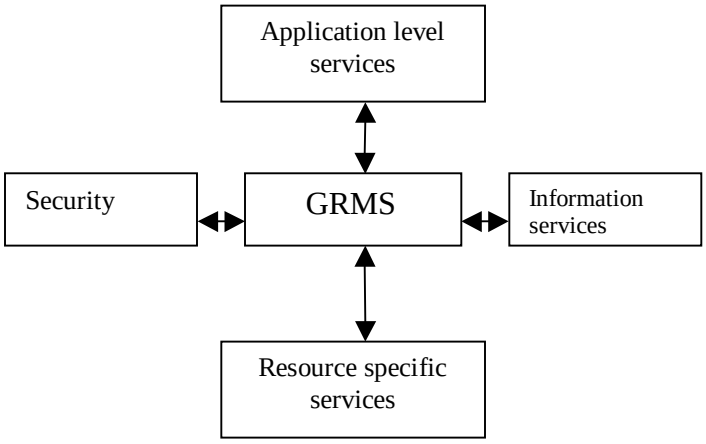
In a grid different type of applications from a wide range of domains are executed each with different resource management requirements. While some type of applications require to be scheduled as soon as possible even if it means reduced performance, some class of application need high performance. The resource management framework should allow new policies to be incorporated into it without requiring substantial changes to the existing code.

In addition to the above a resource management system for the grid should address the other issues like flexibility and extensibility, global name space, security, fault tolerance and Quality of service.

2.2. Context and Functions of GRMS

In a grid system, an end user submits to the management system the job to be executed along with some constraints like job execution deadline, the maximum cost of execution. The function of the resource management is to take the job specification and from it estimate the resource requirements like the number of processors required, the execution time, and memory required. After estimating the resource requirements RMS is responsible for discovering available resources and selecting appropriate resources for job execution. Finally schedule the jobs on these resources by interacting with the local resource management system.

A RMS is also responsible for naming the resources in the system, monitoring and reporting the job, resource status and accounting for resource usage. The RMS interacts with the security system to validate user requests, the information service to obtain information about resource availability, the local system to schedule jobs on the local resource management system



3. Taxonomy

Taxonomy based on the architecture of the resource management system is discussed below. It is divided into the following categories grid type, resource namespace, resource information (discovery, dissemination), scheduling model, scheduling policy.

3.1 Grid type

Most contemporary grids do not provide uniform support for all types of resources. Grid systems are classified as *compute, data* and *service* grids. In compute grids the main resource that is being managed by the RMS is compute cycles (i.e. processors), while in data grids the focus is to manage data distributed over geographical locations. The architecture and the services provided by the resource management system are affected by the type of grid system it is deployed in. Resources could be hardware (computation cycle, network bandwidth and data stores) or software resources (applications and databases).

3.2 Resource namespace Organization

The Grid RMS is responsible for naming the resources in the grid system. Resource naming affects other resource management functions like resource discovery and dissemination and also affects the structure of the database storing resource information. Three approaches to name space organization are *flat, hierarchical and graph* based.

A flat namespace is not very scalable in a grid system. In a hierarchical organization resource naming follows a system of systems approach, a name is constructed by traversing down a hierarchy. In graph based naming resources are linked together and a resource name is constructed by following the links from one object to another.

3.3. Resource Dissemination Protocol

Resource dissemination involves advertising the resources in the grid system. Dissemination protocols effect the amount of data transferred between systems and the also the status of the resource information database. Dissemination protocols can be classified as periodic or on demand.

Periodic dissemination involves batching resource status change information and updating the information database periodically. This can be further categorized as *push or pull dissemination*. In push dissemination resource information data is pushed to the information store periodically while in pull dissemination the information store collects status information the resources periodically.

On Demand dissemination protocols update resource information when a specific event is triggered or status change occurs.

3.4. Resource Discovery

Resource discovery is a very important function of the resource management. Discovery services are used by the scheduling system to obtain information about resource

available. Approaches to resource discovery can be classified as query based or agent based.

In a query based discovery the resource information store is queried for resource availability. Most contemporary grid systems follow this approach. In agent based discovery agents traverse the grid system to gather information about resource availability.

3.5. Scheduling model

Scheduler structure determines the structure of the resource management system and scalability of the system. Most contemporary grid resource management systems can be classified as having *centralized, hierarchical or decentralized structure.*

In a *centralized* organization all jobs are submitted to a single scheduler which is responsible for scheduling them on the available resources. Since all the scheduling information is available at one single position the scheduling decision are optimal but this approach is not very scalable in a grid system.

In a *decentralized* model there is no central scheduler, scheduling is done by the resource requestors and owners independently. This approach is scalable and suits grid systems. But individual schedulers should cooperate with each other in making scheduling decisions and the schedule generated may not be the optimal schedule. Based on whether or not schedulers cooperate they can be further classified as cooperative or non-cooperative schedulers.

In a *hierarchical* model the schedulers are organized in a hierarchy. High level resource entities are scheduled at higher levels and lower level smaller sub-entities are scheduled at lower levels of the scheduler hierarchy. This model is a combination of above two models.

3.6. State Estimation

Users submit the application and input data along with some QoS requirements like deadline to the management system. The management system should estimate the specific resource requirements for running the application. There are three approaches to this problem *theoretical prediction, history-based prediction, and testcase-based prediction.*

In *theoretical prediction,* application requirements are estimated based on an analysis of the application's programming model and problem domain. The efficiency of this approach depends on how well the computing model for the application is understood.

In *history-based* approach prediction is made based on previous runs of the applications. This approach is suitable for applications that are not likely to change over time or are typically executed on a given set of resources where past performance is a strong indicator of future performance of the application on those same resources.

In *testcase-based prediction* the application is tested for a limited set of cases on representative machines. This is suitable for applications that have unknown characteristics or applications that change over time.

3.7. Scheduling Policy

The intention here is not on classifying the various scheduling mechanisms but is on whether the scheduling system (broker) employs a *fixed* policy or an extensible scheduling policy.

Scheduling policy governs how resources are scheduled on the matched resources. In a grid environment there can be no single global scheduling policy, different administrative domains may set different resource usage policies so the RMS should allow for the policies to added or changed with minimal overhead.

4. Survey

Grid systems today are broadly classified as compute, data or service based grids. Resource Management in Condor, Globus, Legion (Compute Grids), European Data grid, PUNCH (On-demand Service grid), Nimrod-G (Economy driven meta-scheduler) have been surveyed. The motivation for choosing these systems was to study how the resource management architecture is affected by the type of the grid system.

4.1 Condor

Condor is a resource management system designed to support high-throughput computations by discovering idle resources on a network and allocating those resources to application tasks. The main function of condor is to allow utilization of machines that otherwise would be idle thus solving the wait-while-idle problem.
A cluster of workstations managed by condor is called a Condor pool. Jobs submitted by the users are queued by Condor and scheduled on available machines in the pool transparently to the user. Condor resource requests are specified in *Classified Ads* resource specification language. Condor selects a machine from the pool to run a user's job, it can also migrate a running job from one machine to another until it is completed.

Condor has a centralized scheduling model. A machine is the condor system (Central Manager) is dedicated to scheduling. Each condor work station submits the jobs in its local queue to the central scheduler which is responsible for finding suitable resources for the job execution. The information about suitable available resources to run the job (execution machine information) is returned to the job submission machine. A shadow process is forked on the submission machine for each job, which is responsible for contacting and staging the job on the execution machine and monitoring its progress. Condor supports pre-emption of running jobs, if the execution machine decides to withdraw the resources Condor can preempt the job and schedule it on another machine thus providing for resource owner autonomy.

The resource information needed for making scheduling decisions is also stored on the central Manager. Central Manager queries the data store for information resource availability. Resource dissemination is through periodic push mechanism. Each machine

in the Condor system advertises its resources and reports resource status to the central manager.

Negotiation is done through machine and job context .The resource owner i.e. the machine in Condor specifies the usage conditions or policy in the machine context submitted to CM. The resource requirements of the job are specified in the job context. The scheduler matches the requirements in job context to the usage conditions in the machine context.

Condor preserves job's submission machines environment at the execution machine through a *remote execution mechanism*. The system calls from User process can be redirected to the submission machine where they are handled by the shadow. The Shadow executes the system call locally and returns the result to the User Process.

Condor flocking is an enhancement to condor which enables different condor pools to be connected such that jobs submitted in one pool can run in other pools. This allows jobs to run on machines owned by different organizations.

In Condor a *checkpointing* mechanism is implemented which stores the current state of the job, checkpoint is saved in a checkpoint file on disk space accessible from the submission machine. *This checkpointing mechanism offers fault tolerance and recovery*, if a job executing on machine fails or if the resource is claimed back by the owner then the job can be restarted on another machine by reconstructing the state of the job from the checkpoint file.

4.2 Resource Management in Globus

Globus toolkit is a collection of tools that provides the basic services and capabilities like security, resource management, information services etc required for grid computing. Resource Management System of Globus consists of *resource brokers*, *resource co-allocators* and *resource manager* or GRAM. The resource requests are specified in *extensible resource specification language (RSL)*.

Globus has a decentralized scheduling model. Scheduling is done by application level schedulers (like AppleS) and resource brokers. Application specific brokers translate the application requirements into more specific resource specification. Resource Brokers are responsible for taking high-level RSL specification and transforming them into more concrete specification (this process is called specialization).Requests can be passed to multiple Brokers. Transformations done by the brokers' results in a request in which the locations of the resources are completely specified.

The Resource Brokers discover resources by querying the information service (MDS) for resource availability. MDS is a LDAP based network directory (Metacomputing Directory Services). MDS consists of two components Grid Index Information service (GIIS) and Grid resource information service (GRIS). GRIS provides resource discovery services. GIIS provides a global view of the resources by pulling information from the GIIS's. Resource information on the GIIS's is updated by push dissemination. Globus has a hierarchical name space organization.

The transformed resource requests from resource brokers are passed to the co-allocator. Co-allocator takes care of multi-requests, multi request is a request involving resources at multiple sites which need to be used simultaneously, and passes each component of the request to appropriate resource manager and then provides a means for

manipulating each resultant set of managers as a whole. The Co-allocation of resources is done by the DUROC component Globus.

The resource manager interacts with local resource management systems to actually schedule and execute the jobs. The implementation of the resource manager in Globus is called GRAM. GRAM authenticates the resource requests and schedules them on the local resource manager. Each user is associated with a UHE (user hosting environment) on the execution machine. All the jobs from a user are directed to the user's UHE, which starts up a new Managed Job Factory service (MJFS) instance for every job. The MJFS communicated with the clients by starting up two instances of File Stream Factory Service (FSFS) for standard input and output. MJFS and FSFS are persistent services.

When a system starts up after a fault all the UHE which were running before crash are started up. There is also a Sweeper task which runs every two hours and recovers any crashed UHE. The information about the UHE is obtained from gridMapfile which stores the status (Active, Inactive) of the UHE's in the system, if a nonexistent UHE is marked active in the gridMapfile it indicated that the UHE crashed. After a UHE is restarted all the persistent services in it (MJFS and FSFS) are recovered.

Globus' GARA component provides QoS guarantees through advanced resource reservation. Nexus provides the communication system between resources. Globus toolkit provides the middleware services for resource management while the scheduling tasks are done by individual resource broker like AppleS and Nimrod-G etc.

4.3 Resource Management in Legion

Legion is a reflective, object-based operating system for the Grid. It offers the infrastructure for grid computing. Legion provides a framework for scheduling which can accommodate different placement strategies for different classes of applications.

Scheduler in Legion has a hierarchical structure. Users or active objects in the system invoke scheduling to run jobs, higher level scheduler schedules the job on cluster or resource group while the local resource manger for that domain schedules the job on local resources. Scheduling in Legion is placing objects on the processors. The resource namespace is graph based.

Scheduling framework acts as mediator to find a match between the placement requests and processors. When a job request is submitted, appropriate scheduler for the job is selected from the framework. Selection of the scheduler can be made by the user explicitly or attributes of the Class Object of the application can be used to specify the scheduler for the objects of that class. Object specific placement constraints are also specifies as attributes on the class object. Resource owners also specify security and resource usage policies using class Object attributes. Scheduler Object uses this information in making scheduling decision. Co-allocation of resources is not supported.

The enactor object is responsible for enforcing the schedule generated by the scheduler object; more than one schedule is generated, if a schedule fails another one is tried until all the schedules are exhausted.

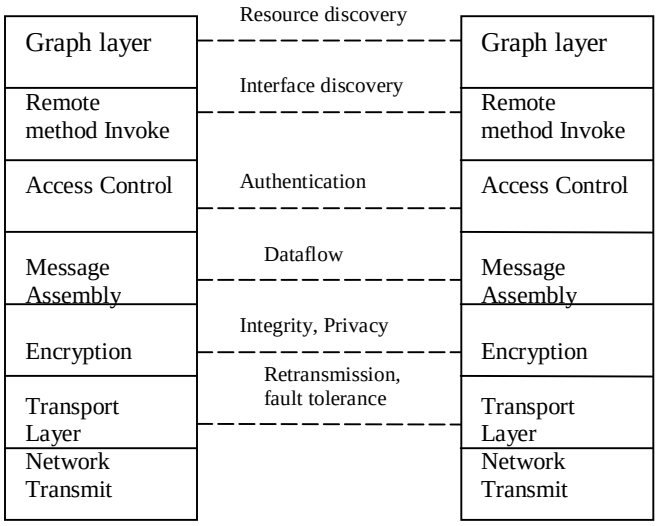
Information about resources in the grid is stored in database object called a collection. For Scalability there could be more than collection object and collections can

send and receive data from each other. Information is obtained from resources either by pull or push mechanism. Users or Schedulers query the collection to obtain resource information.

Legion supports resource reservation and object persistence. When the scheduler object contacts a host object (processor or local resource management system), the host returns a reservation token to the scheduler if the job can be executed on its resources. Every object is associated with vault object. Vault object holds associated object's Object Persistent Representation (OPR). This ensures that even if the object fails, it can later be re-constructed from the OPR.

Communication between any two objects goes through the Legion Protocol stack which involves constructing program graphs, making method invocations, checking authorization, assembling or disassembling messages, encrypting, re-transmitting messages etc. This frameworks allows for implicit security and fault-tolerance

Fig



Legion objects are *fault tolerant* since they can be reconstructed consistently using OPR stored in vault object. Some objects are also replicated for performance or availability.

4.4. PUNCH

Punch is a demand-based grid computing system that allows end users to transparently access and use globally distributed hardware and software resources. The resource management system in PUNCH has a pipelined architecture. With individual components in the pipeline replicated and geographical distributed for scalability and reliability. PUNCH employs a non-preemptive, decentralized, sender-initiated resource management framework.

The global namespace organization is hierarchical. Users specify the application and input data files to the management system through a web-based interface. The application unit on the client side estimates the resource requirements for executing the application using machine learning based prediction techniques and submits the resource requirements to a query manager of the resource management pipeline.

The resource management pipeline consists of the query manager, resource managers, and pool managers. Pools are dynamic aggregation of resources based on some criteria (e.g. processor architecture). Pools are created on the fly if a request can not be mapped to any of the pool managers.

Scheduling is performed in a decentralized manner by resource and pool managers. A query manager parses the resource request, transforms it into an internal representation and forwards it to a resource manager based on some criteria. Individual Resource managers try to map the requests to a pool manager from the list of pool managers stored its local database. If a request cannot be mapped to any of the pools, a new pool is created. When a new pool manager is created, it queries a resource information database for information of all the resources which satisfy the pool's criteria.

Scheduling policy is extensible. Each pool manager has one or more scheduling processes associated with it; the function of these processes is to sort the machines in its pool cache according to some specified criteria (average load, available memory) and to process queries sent by resource managers. Pool managers can be configured to utilize different scheduling policies.

The resource information is initially pushed to the information database and a monitoring service monitors the status of the resources and updates the information database (data pull).

PUNCH supports QoS negotiation by passing the request to multiple resource managers and utilizing the best response. It doesn't support advanced reservations or co-allocation of resources. PUNCH is intending for large number of short jobs or bursty submissions. PUNCH employs a non-preemptive, decentralized, sender-initiated resource management framework.

4.5. European Data Grid

EU datagrid was designed to provide distributed scientific communities access to large sets of distributed computational and data resources. The three main application areas of EU datagrid are the High Energy Physics (HEP) where geographical distributed researcher analyze the same data generated by single source but replicated distributed data stores, Earth Observation data are collected at distributed stations and also maintained in distributed databases, while in Molecular Biology research large number of independent datasets are used which need to be integrated into one logical system.

The main architecture of the datagrid is layered. The datagrid project develops datagrid services and depends on the Globus toolkit for core middleware services like security. The datagrid services layer consists of workload management services which contain components for distributed scheduling and resource management, Data Management services contains middleware infrastructure for coherently managing

information stores and monitoring services provided end-user and administrator access to status information on the grid.

The workload management package consists of a user interface, resource broker, job submission service, book keeping and logging service. A job request from user is expressed in a Job Description Language based on the Classified Ads of Condor. The resource broker (RB) given a job description tries to find the best match between the job requirements and available resources on the grid, considering also the current distribution of load on the grid. RB interacts with data replication and meta-data information services to obtain information about data location.

The information service a LDAP based network directory. Resource discovery is by queries and employ periodic push for dissemination. Global namespace hierarchical and scheduling is decentralized but instead of having a resource broker for each end-user, each virtual organization is provided resource broker.

It does not support advanced reservation or co-allocation of resources. It does not address failures originated by jobs which it simply reports to end user. But the state of the resource broker queues and job submission service queues is persistent and can be recovered fully after a crash.

4.6. Nimrod-G and GRACE

Nimrod-G is grid-enabled resource management and scheduling system based on the concept of computational economy. It was designed to run parametric applications on computational grid. It uses the middleware services provided by Globus Toolkit but can also be extended to other middleware services.

Nimrod-G uses the MDS services for resource discovery and GRAM APIs to dispatch jobs over grid resources. The users can specify deadline by which the results of there experiments are needed. Nimrod-G broker tries to find the ***cheapest*** resources available that can do the job and meet the deadline. Nimrod uses both static cost model (stored in a file in the information database) and dynamic cost model (negotiates cost with the resource owner) for resource access cost trade-off with the deadline.

GRACE provides middleware services needed by the resource brokers in dynamically trading resources access costs with the resource owners. It co-exists with other middle-ware systems like Globus. The main components of the GRACE infrastructure are a Trade Manager (TS), trading protocols and Trade Server (TS). TM is the GRACE client in the Nimrod-G resource broker that uses the trading protocols to interact with trade servers and negotiate for access to resources at low cost. Trade Server is the resource owner agent that negotiates with resource users and sells access to resources. TS uses pricing algorithms as defined by the resource owner that may be driven by the demand and supply. It also interacts with the accounting system for recording resource usage.

It has an extensible application-oriented scheduling policy and scheduler uses theoretical and history based predictive techniques for state estimation. Scheduler organization is decentralized and the namespace is hierarchical.

5. Summary

Most common resource management architecture among the surveyed grid systems was a hierarchical namespace with decentralized scheduler structure, query based resource discovery, push or pull dissemination. Object based architecture like Legion adopted graph based namespace.

The resource management systems varied in there support co-allocation of resources, for QoS which is achieved in most through advanced reservations, support for fault tolerance which was mostly through replication of the components or recovering using persistent state.

6. Conclusions

In this paper various issues in resource management in grid systems have been discussed. A taxonomy based on architecture for classifying grid resource management systems has been described. The results of the survey of resource management systems in various contemporary grid systems have been presented.

7. References

[1]. I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2]. Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz (co-editors), Grid Resource Management State of art and Trends, Kluwer Publishers, fall 2003.

[3]. I. Foster, C. Kesselman, S. Tuecke, The anatomy of grid enabling scalable virtual organization, lecture notes (2001).

[4]. I. Ekmecic, I. Tartalja, and V. Milutinovic, *A survey of heterogeneous computing: Concepts and Systems,*Proceedings of the IEEE, Vol 84, No 8, Aug 1996, pp. 1127-1144.

[5] H.G. Rotithor, *Taxonomy of dynamic task scheduling schemes in distributed computing systems*, Proceedings of Computer Digital Technology, Vol 141, No 1, January 1994, pp. 1-10

[6]. T.L. Casavant and J. G. Kuhl, *A taxonomy of scheduling in general-purpose distributed computing systems*,

[7]. T. Braun, J. Siegel, et al, *A Taxonomy for describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems*, IEEE Workshop on Advances in Parallel and Distributed Systems, in Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems 1998, pp. 330-335

[8]. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[9]. R.J.M. Boer, *Resource Management in the Condor System*, Master Thesis, 1996.

[10]. S. Chapin, J. Karpovich, A. Grimshaw, *The Legion Resource Management System*, Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999.

[11]. A. Natrajan, M. HumphreyN. Kapadia and J. Fortes, *PUNCH: An Architecture for Web-Enabled Wide-Area Network-Computing*,

[12]. D Royo, N. Kapadia, J. Fortes, and L. Cerio, *Active Yellow Pages: A Pipelined Resource Management.*

[13]. W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, *Data Management in an International Data Grid Project.*

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

[15]. R. Buyya, J. Giddy, D. Abramson, *An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications*, Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000), Kluwer Academic Press, August 1, 2000, Pittsburgh, USA.