



---

# Resource Discovery Management Scheduling

Manish Mahajan



# Introduction

---

- Ever increasing number of powerful regional and local computing environments
- Aggregated capacity of these new computing resources is enormous
- Yet, exploitation of these resources fallen short of expectation as few users reach or access resources beyond their home institution



## Problem

---

- “Potential barrier” due to associated mechanisms, policies, failure modes, performance uncertainties across administrative domains
- Overcoming these potential barrier requires new methods and mechanism that meet some key user requirements for computing in a “Grid” that comprises resources at multiple locations



## User Requirements

---

- Users want to discover acquire and reliably manage computational resources dynamically
- Do NOT want to be bothered about location of these resources, mechanisms required, tracking status of computational tasks, reacting to failure
- DO care how long their tasks are likely to run and cost to run them



# Different Schemes/papers/projects for resource discovery

---

- Condor-G
- GrADS (Grid Application Development Software)
- Resource Selection Framework
- Peer-to-Peer Approach to Resource Discovery
- SNAP



# Condor-G: Computational Management Agent for Multi- Institutional Grids

---

- Leverages software from Globus and Condor to allow users to harness multi-domain resources as if they all belong to one personal domain
  - Security, resource discovery, and resource access in *multi-domain* environments, as supported within the Globus Toolkit
  - Management of computation and harnessing of resources within a *single* administrative domain, specifically within the Condor system



## Briefly...

---

- Combine inter-domain resource management protocols of Globus and intra-domain resource management methods of Condor to give one personal domain view.
- User defines the tasks to be executed and Condor-G handles all other aspects (discovery, execution etc)
- Result is a powerful tool to manage variety of parallel computations in Grid environments
- The question addressed in this paper is how to build and manage a multi-site computation that uses required resources



## Issues faced

---

- Different sites may feature different authentication and authorization mechanisms, Operating systems, hardware architectures, etc
- User has little knowledge of resource characteristics at remote sites and this information is hard to obtain
- Failure problems inherent due to distributed nature of multi-site computing environment, networks, etc
- Tedious bookkeeping to track status of different elements in event of failure



## Above issues addressed by Condor-G by 'Separation of Concerns'

---

- *Remote resource access* issues: by secure discovery, allocation and management of resources using Globus protocols
- *Computation management* issues: via introduction of multi-functional users computation management agent (Condor System Component)
- *Remote execution environment* issues: via use of *mobile sandboxing* technology (Condor Component)



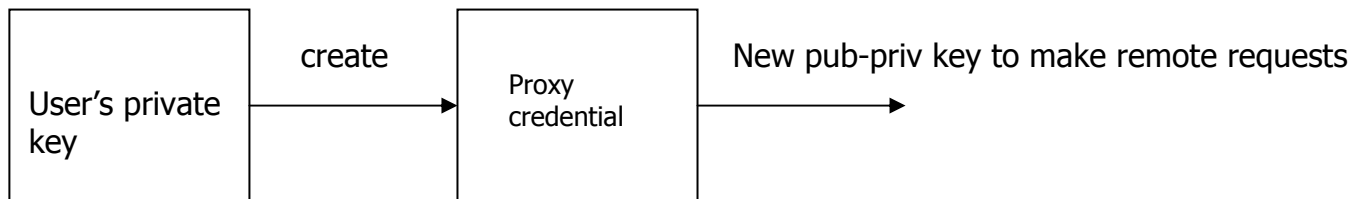
## Benefits of Separation of Concerns

---

- Easy for a remote resource to speak simple protocols rather than to require it to support complex distributed computing environment
- Careful design of remote access protocols can significantly simplify computation management

# Grid Protocols Overview

- Grid Security Infrastructure (GSI)
  - User authenticated just once using public key infrastructure (PKI) mechanisms to verify user supplied "Grid Credential"
  - One overall credential used to verify other local credentials
  - GSI's PKI mechanism:



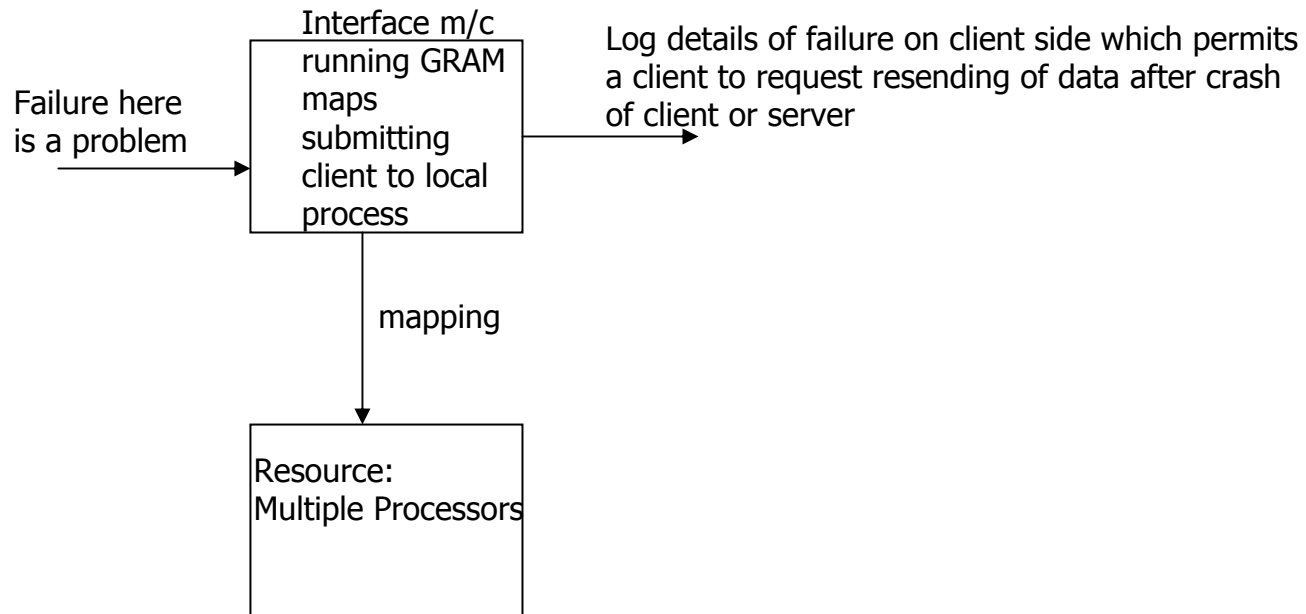


## GRAM Protocol

---

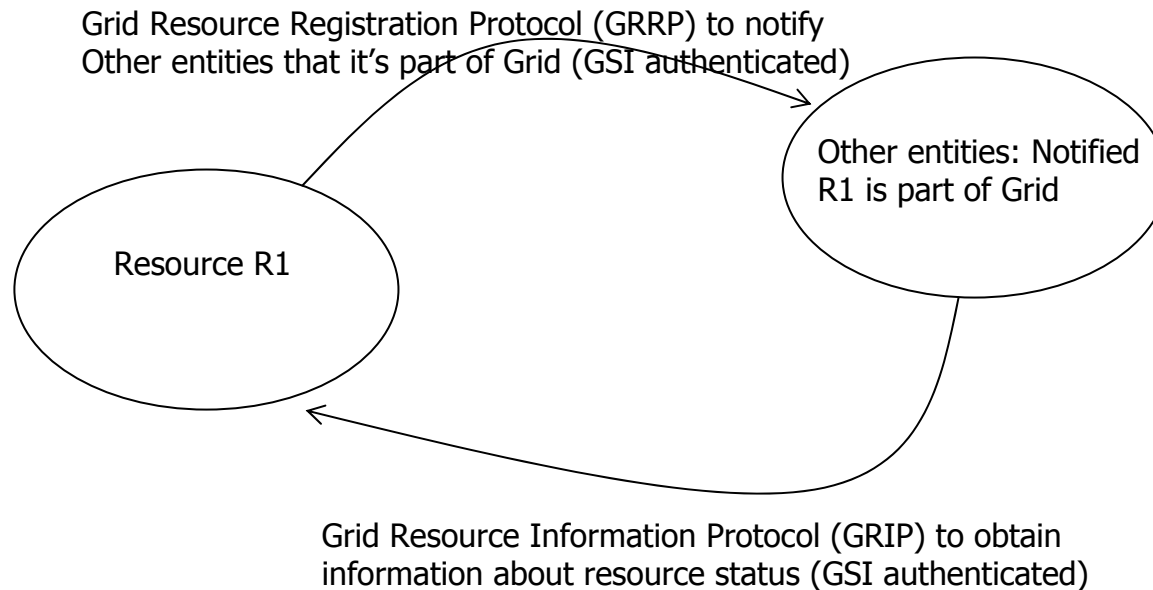
- Supports remote submission of computational request to a remote computational resource, and subsequent monitoring and control of resulting computation
- Security, two-phase commit and fault tolerance
  - GSI security mechanisms for all ops
  - Two-phase commit to achieve “exactly once” execution
    - Request/Response sequence number takes care of lost request or response
  - Log details of active jobs to stable storage at client side, allowing information to be retrieved if GRAM server crashes

# GRAM fault tolerance



# MDS (Monitoring and Discovery Service) Protocols

- Provides basic mechanisms for discovering and disseminating information about the structure and state of Grid resources e.g. Compute server configuration, CPU load
- GRRP/GRIP allow to construct directories that support discovery of interesting resources



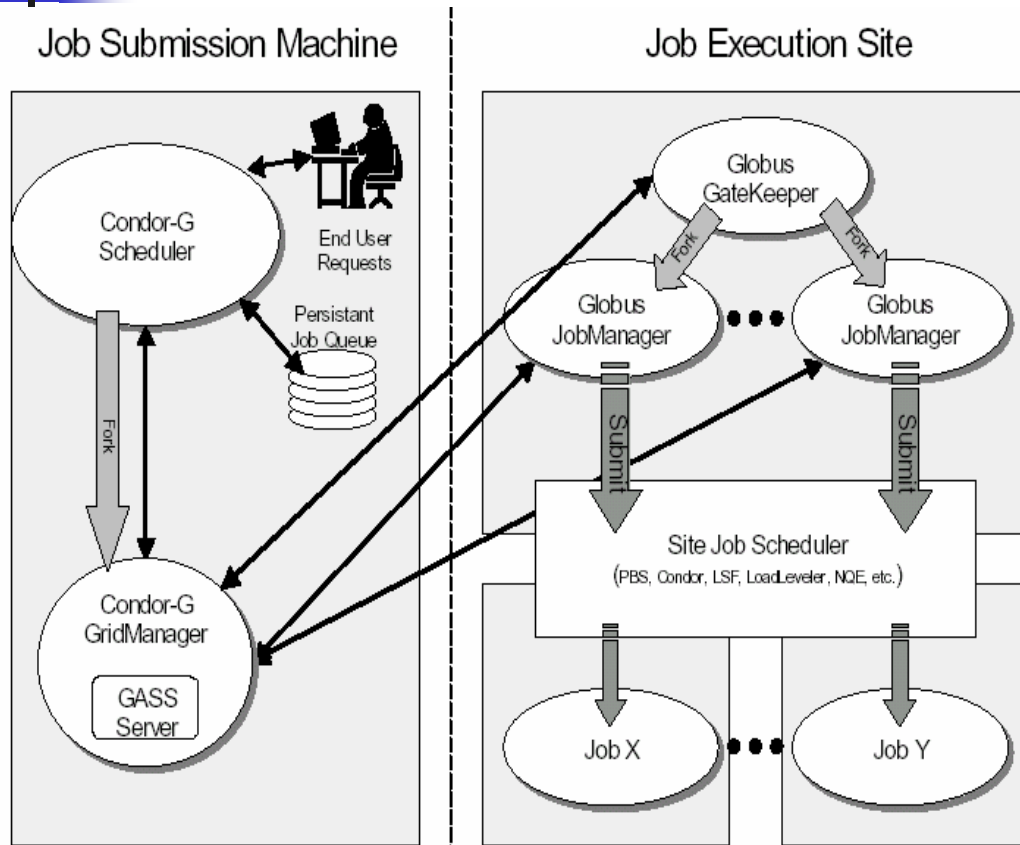


## GASS (Global Access to Secondary Storage)

---

- Provides mechanisms for transferring data between a remote HTTP, FTP, or GASS server
- Used here to stage executables and input files to a remote computer (GSI used for authentication)

# Condor-G Agent



- Stages job's standard I/O and executable using GASS
- Submit job to remote machine using GRAM
- Monitor job status and failure recovery using GRAM
- Authenticate via GSI



## Working of Agent...

---

- Scheduler takes jobs from user, creates GridManager
- GridManager (GM) handles all jobs for a single user
- Each GM job submission request (execution site via GRAM) creates one Globus JobManager (JM)
  
- JM uses GASS to transfer job's executable
- JM submits jobs to execution site's local scheduler
- JM updates GM which then updates scheduler where job status is stored persistently



## Types of failure

---

- Crash of JM,
- Crash of machine that manages GateKeeper and JM,
- Crash of GM,
- Crash of machine that runs GM
- Failure in the network



# Failure Management

---

- GM probes JM for all jobs
- If JM doesn't respond GM probes GateKeeper.
- If GateKeeper responds then problem with JM otherwise problem with whole network (cannot distinguish between the two)
- GM starts JM or waits to establish network connection
- If JM crashed then normal resume
- If network problem but JM finishes job okay, then GM is informed that job is done
  
- To recover from local failure, all state is stored persistently in scheduler's job queue.

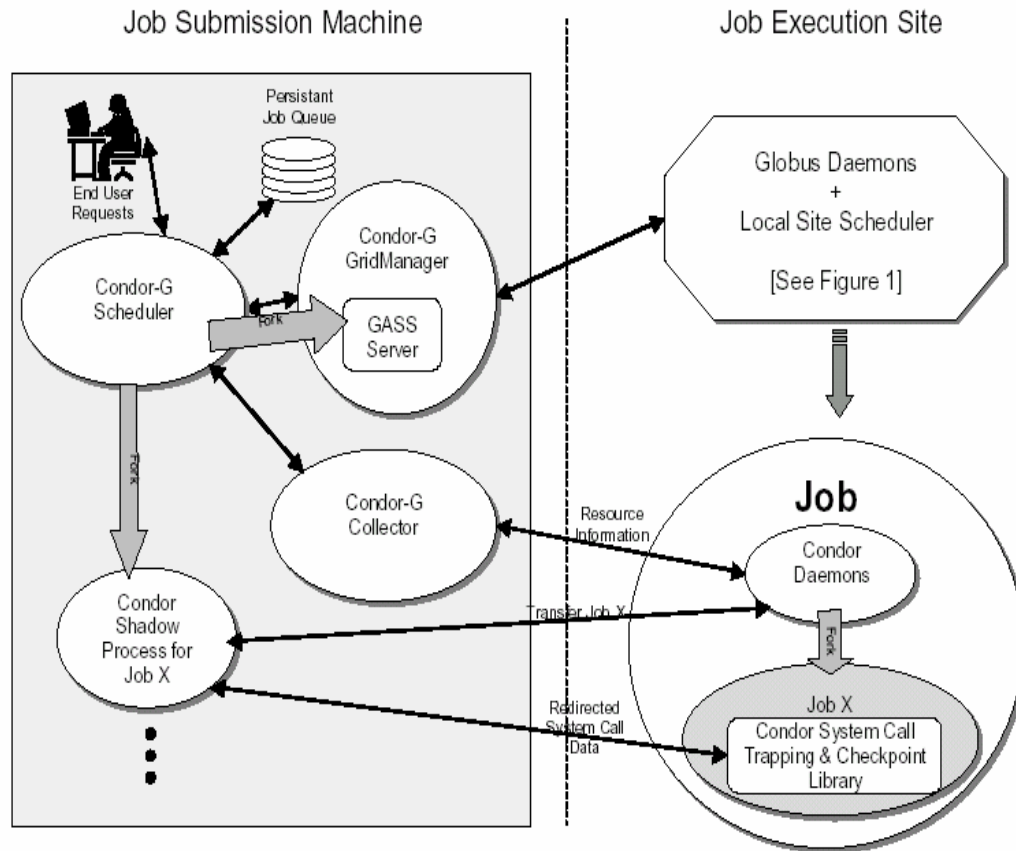


# Credential Management

---

- User's credentials have finite lifetime
- A long lived Condor-G computation must be able to deal with credential expiration
- Methods
  - Inform users by mail
  - Set credential alarms for a reminder
  - MyProxy for long-lived proxy credential

# Resource Discovery and Scheduling



- Simple approach is user supplied list of GRAM servers
- Sophisticated approach is to construct *resource broker* to build a list of candidate resources
- GlideIns: “Flood” candidate resources with requests to execute jobs. A simple but effective technique



## GlideIn Mechanism

---

- Monitor actual queuing and execution times to tune submission of subsequent jobs and to migrate queued jobs
- Mobile Sandboxing technique
  - Start Condor daemon at remote site (not user process)
  - Daemon advertises availability to Condor Collector
  - Scheduler queries Collector to learn of available resources
  - Condor-G matches locally queued jobs with resources to remotely execute them on these resources
  - Runs each user task received in a “sandbox” which increases portability and protects local system
  - Periodically checkpoints job to another location if requested



## Advantages

---

- Condor-G creates personal Condor pool out of Grid resources by “gliding-in” Condor daemons to remote resource.
- Daemons shut down gracefully
- Avoids users having to store binaries for all potential architectures on their local machines (by retrieving Condor executables from central repository)
- Minimize queuing delays by delaying the binding of application to resource till it is available
- Guarantee’s optimal/better queuing times by submitting GlideIns to all remote resources capable of serving the job



## Examples

---

- Used by a team of four mathematicians from Argonne National Laboratory to solve optimization problem involving 2,500 CPUs and 10 sites
- Caltech CMS Energy Physics Collaboration to perform large-scale distributed simulation and reconstruction of high-energy physics events
- GridGaussian project at NCSA to prototype a portal for running Gaussian98 jobs on Grid resources. This uses GlideIns to optimize access to remote resources.



## Related Work

---

- Condor, DQS, LSF, LoadLeveler, and PBS: In all these cases, different domains must run same resource management software
- Notably Condor-flocking and Condor-G: The prior uses special purpose mechanisms of Condor for inter-domain operation on remote resources that require authentication as against Condor-G that uses standard protocols
- Nimrod and Condor-G: The latter addresses issues of failure, credential expiry not addressed by Nimrod



## GrADS (Grid Application Development Software)

---

- Tremendous potential and enthusiasm to Grid paradigm, but...
  - Challenges of dynamic and Complex nature of Grid environment
  - Few software tools exist
  - Our understanding of algorithms and methods is limited
  - Middleware isn't suitable for a broad class of applications
  - Impressive applications have been developed by only specialists
- Overall: To have widespread acceptance there is a need to have relatively easy to develop grid applications



## GrADS Vision

---

- End user should be able to specify applications in high level, domain-specific problem-solving languages and expect these applications to seamlessly access the Grid to find required resources when needed.
- User is thus free to concentrate on HOW to solve a problem rather than how to MAP a solution on available Grid resources
- Discussion...such problems could be astro physics, oil exploration?



## GrADS and it's goals

---

- Framework for preparing and Executing Adaptive Grid Programs
- Goal
  - Provide good resource allocation
  - Support adaptive reallocation if performance degrades
- Strategy: Configurable Object program
  - Contains application code, strategies for mapping to resources and resource selection model that provides estimate of performance of application
  - Contract monitoring to remap application execution when performance degrades

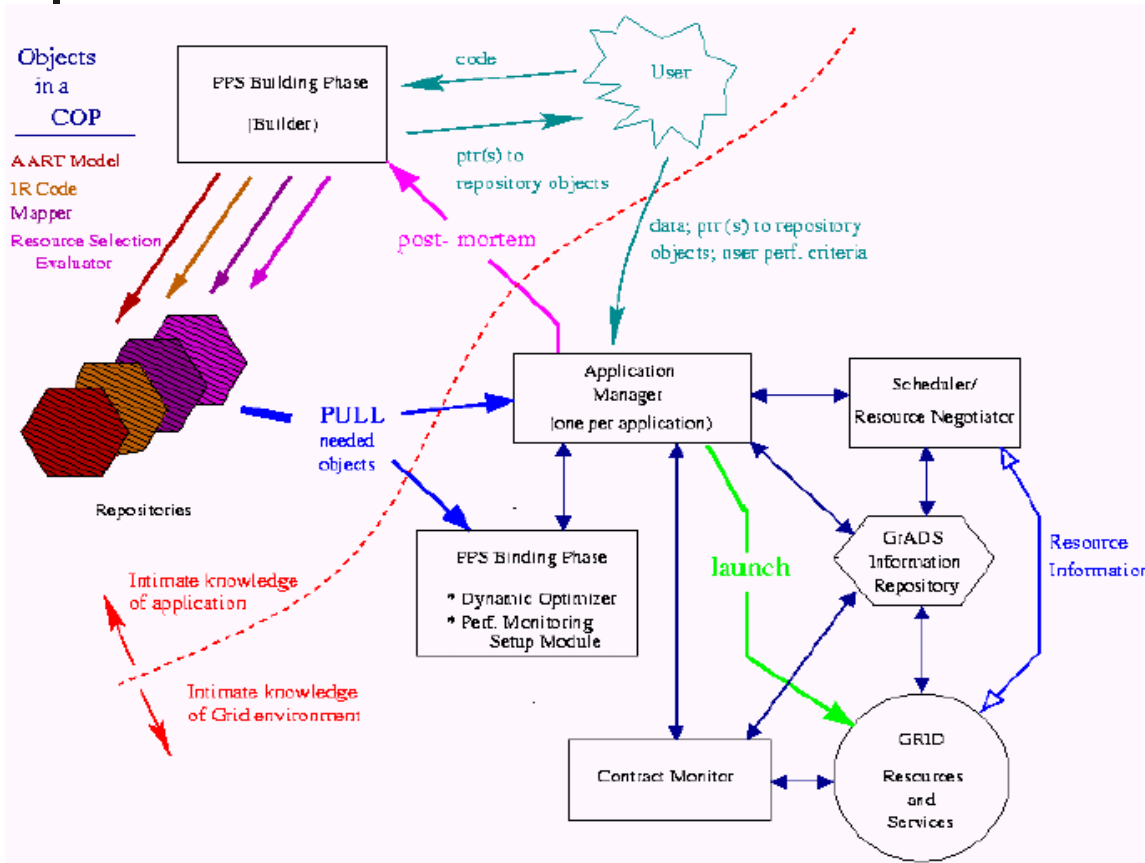


# Framework

---

- Application Manager is a process that...
  - Initiates resource selection
  - Launches the problem
  - Runs it
  - See its execution through completion
  - Responsible for “when” and “what” input or state for the application
  - Compare with Condor-G where GridManager and JobManager do this function

# Application Execution Scenario



- User provides source code with resource selection or run-time behavior
- Builder uses this information to produce Configurable Object Program (COP)





## Scenario continued...

---

- Builder constructs code, Mapping strategy (mapper) and a resource selection evaluator (RSE) performance model
- Builder provides Application Abstract Resource and Topology (AART) Model which is a model of resource space needed for application execution
- Purpose of AART is to kickstart resource selection process and provide information to mapper and RSE



## Continued...

---

- User starts Application Manager (AM)
- AM generates Resource Selection Seed Model from COP and AART
- This Seed is preliminary state necessary for mapper and RSE
- AM invokes Scheduler/Resource Negotiator (S/RN)
- S/RN is an optimization procedure that searches the best acceptable resources for application determined by RSE



## Continued...

---

- After resources are identified, AM launches appln
  - Stores state (checkpoint) in Program Execution System (PES) Repository
  - This helps to restart application after component failure
- AM invokes PPM which in turn creates optimized binaries and inserts monitoring sensors for performance monitoring
- AM starts Contract Monitor (performance monitoring and rescheduling) and launches the binaries
- Contract Monitor gathers sensor data to determine performance



## Resource Selection Framework

---

- The problems of first discovering and then organizing resources to meet application requirements are difficult
- Other projects that have addressed resource selection have done so either explicitly (job control language) or implicitly (submit job to particular queue)
- On others, the details of some applications are embedded making it difficult to use for other apps
- Problem is that neither user nor owner can control resource selection process in these systems
- Condor provides general resource selection mechanism but the matchmaker for condor was designed for selecting a single machine to run the job



## Problem addressed

---

- This paper presents a general purpose resource selection framework that can be used by different kinds of application
- Both application resource requirements and application performance models are specified declaratively while mapping strategies can be determined by user-supplied code



## Condor ClassAd Syntax

---

- ClassAd is mapping from *attribute names* to *expressions*
- Matchmaker takes two ClassAds and evaluates one with respect to the other
- Two ClassAds match if each ClassAd has an attribute named "requirements" that evaluates to true in the context of the other ClassAd e.g. `other.size > 3`, `rank`
- In resource selection, matchmaker evaluates a ClassAd request with respect to every available resource ClassAd and then selects matching resource with highest rank



## Set-Extended ClassAd Syntax

---

- Extends ClassAds by a Type specifier, boolean function *suffix* and three aggregation functions namely, Max, Min and Sum
- Aggregation functions are used to decide the rank of the resource set e.g. rank of resource set is decided by the longest subtask execution time i.e.  $\text{Rank} = 1/\text{Max}$
- User can use suffix function to constrain within particular domains the resources considered



## Set-Matching Algorithm

---

- Evaluates set-extended ClassAd against a set of ClassAds and returns ClassAd with highest rank
- Two phases
  - Filtering : individual ClassAd are removed from consideration based on individual requests
  - Set Construction: Select the ClassAd that best meets the application requirements
    - This is done using a greedy heuristic algorithm



# Greedy Algorithm

```
CandidateSet = NULL;
BestSet=NULL;
LastRank = -1; Rank = 0;
while (ClassAdSet != NULL)
{
    Next = {X : X in ClassAdSet && for all Y in ClassAdSet,
            rank(X+CandidateSet) > rank(Y+CandidateSet); }
    ClassAdSet = ClassAdSet - Next;
    CandidateSet = CandidateSet + Next;
    Rank = rank(CandidateSet);
    If (requirements(CandidateSet)==true && Rank > LastRank)
        BestSet=CandidateSet;
        LastRank=Rank;
}
if BestSet ==NULL return failure
else return BestSet
```

- The algorithm repeatedly removes the “best” resource remaining in the ClassAd pool and adds to “candidate set”
- “Best” is determined on basis of rank
- “candidate set” becomes new “best” candidate
- Stops when ClassAd pool is empty or failure if no such resource set

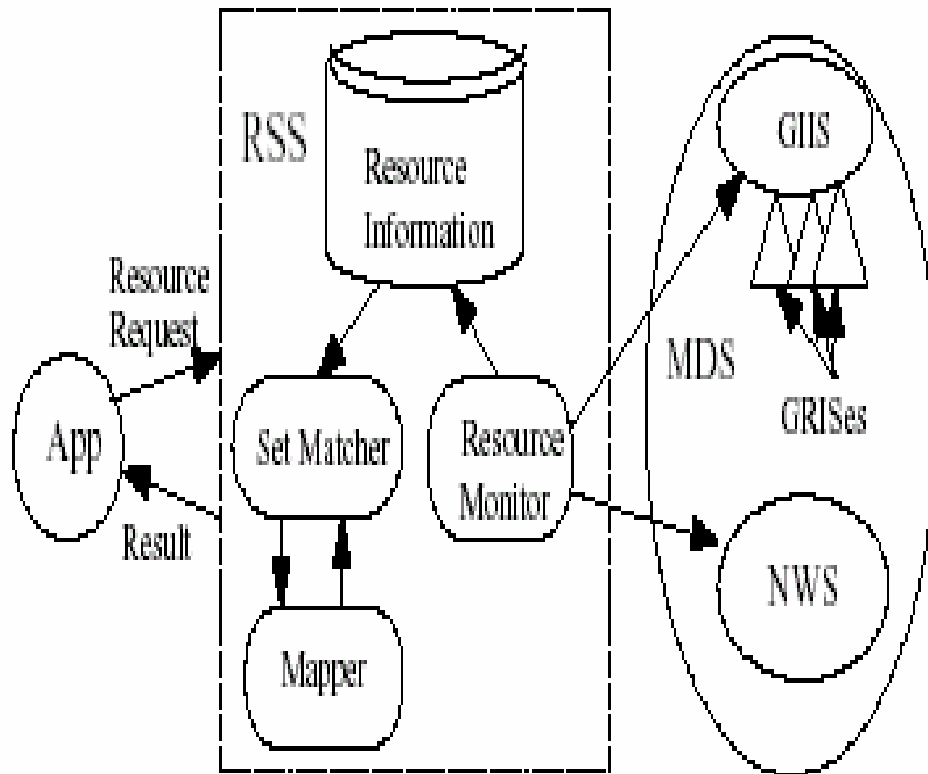


## Resource Selection Framework

---

- Based on set matching technique
- Accepts user resource requests
- Finds set of resources with highest rank based on resource information from a Grid information service
- An open interface allows users to customize the resource selector by specifying an application-specific mapping module

# System Architecture



- Resource Selector Service (RSS)
  - Resource Monitor
  - Set Matcher
  - Mapper



## Resource selector service

---

- Resource monitor queries MDS to obtain resource information and caches this in memory
- Set matcher matches incoming application requests with best set of available resources
- Mapper decides the resource topology and the allocation of application workload to resources



## Resource Request

---

- RSS accepts synchronous and asynchronous requests
- Syn requests: respond with best available resources or failure
- Asyn requests: specifies a request lifetime value and RSS responds if a matching resource set becomes available during that time



## E.G.

```
1. [  
2. ServiceType = "Synchronous";  
3. Type="Set";  
4. iter=100; alpha=0; x=100; y=100; z=100;  
5. A=370; B=254; startup=30; C=0.0000138;  
6. computetime = x*y*alpha/other.cpuspeed*A;  
7. comtime = (other.RLatency+ y*x*B/other.RBandwidth  
             +other.LLatency+y*x*B/other.Lbandwidth);  
8. exectime = (computetime+comtime)*iter+startup;  
9. Mapper = [type="dll"; libname="cactus"; func="mapper"];  
10. requirements =Suffix(other.machine, domains)  
    && Sum(other.MemorySize) >= (1.757 + C*z*x*y);  
11. domains={ cs.utk.edu, ucsd.edu};  
12. rank=Min(1/exectime)  
13. ]
```

- Lines 2,3: synchronous set
- 4-8: Job description including problem size and model
- 8: models execution time of every sub-task to find rank
- 9: gives name and location of mapping algorithm
- 10: specifies resources constraints
- 12: calculates rank



## Resource selection result

---

```
<virtualMachine>
  <result statusCode="200" statusMessage="OK"/>
  <machineList>
    <machine dns="tore2.cs.utk.edu" processor=2 x=20>
    <machine dns="tore3.cs.utk.edu" processor=2 x=15>
    <machine dns="tore6.cs.utk.edu" processor=2 x=15>
  </machineList>
</virtualMachine>
```

- Returned resource set includes three machines each with two processors
- Workload is allocated to the machines according to the ratio 20:15:15



# Performance

---

- Experiments were carried out in the context of a Cactus application that simulates the 3D scalar field produced by two orbiting sources
- The various properties tested were
  - Performance model
  - Mapping Algorithm
  - Mapping test strategy
  - Resource selection algorithm test
- Claim: This framework would adapt to different applications and computational environments. Further experiments on other application are needed to validate this



## Peer-to-Peer and Grids

---

- Computational Grids and P2P communities are two resource sharing environments that have polarized distributed systems research
- Is there a common ground between the two?
- This paper explores...



## Grids and P2P Differences

---

- Computation Grids are characterized by a large variety of services provided to (traditionally) scientific communities of hundreds of users
- P2P environments boast a larger community of users (hundreds of thousands of simultaneous users) but they offer limited, specialized services.
- Belief: design objectives of the two environments will eventually converge



## Grids Characterized by...

---

- Scale
- Some level of Centralization, e.g; centralized repositories for shared data
- Support for complex operations: program execution, resource monitoring, accountability etc
- Stability in resource participation: available for long predefined periods of time except for occasional failur
- Some level of homogeneity in usage behavior: due to incentive or policies that encourage fair sharing
- Homogeneity in resources: resources are often powerful with good network connectivity
- Existence of technical support personnel capable of fixing things



## P2P characterized by...

---

- Large scale: in order of hundreds of thousands
- Less centralization, e.g. Gnutella but Seti is centralized
- Specialized functionality: file sharing or embarrassingly parallel computations, often in the absence of trust enforcing mechanisms.
- Unpredictable resource participation: subject to influences of social, economical, or political nature
- Lack of implicit incentives for good behavior may led to uneven user behavior e.g. free riding
- Highly heterogeneous resources: modem-high bandwidth links
- Lack of technical expertise and administrative authority



## Common ground

---

- Possibility that the two environments will converge into large-scale dynamic sets of resources that can be exploited in a variety of ways
- Large scale: millions of resources
- Lack of global centralized authority
- Highly variable participation patterns: resources may join and leave network frequently but will be stable
- Strong diversity in
  - Shared resources: heterogeneous
  - Sharing characteristics: well defined public sharing policies
- Lack of homogeneous administrative support



## Resource discovery requirements

---

- Lack of centralized control: adopt a self configuring, distributed architecture
- Construct unstructured networks: no distributed hash tables where all nodes have equal responsibilities, which assumes homogeneous capabilities and trust
- No global naming scheme for describing resources
- Assumption: Every participant in the VO has one or more servers that store and provide access to resource information in a decentralized environment



## Four Axes of the solution space

---

- Membership protocol: refers to how new nodes join the network and learn about each other. Responsible for collecting and updating information about the currently participating nodes.
- Overlay construction function: selects the best (according to some metric) set of active collaborators from local membership list (known ones) and hence influences the topology of the overlay network



## Continued...

---

- Preprocessing refers to the off-line preparations for better search performance, independent of requests: e.g., caching is not a preprocessing technique, while prefetching is. E.g. advertise description of local resources to other areas of the network. Rewire the overlay network for better performance. Makes preparation for a more efficient search
- Request processing performs the search itself:
  - Local processing includes looking up the requested resource in local information
  - Remote processing: refers to request propagation rule by sending the (potentially locally processed) requests to other peers through various mechanisms



## Summary

---

- Grids will increase in scale and inherently will have to deal with more dynamism than today
- P2P systems will start to provide more complex functionalities integrating data and computation sharing with various security requirements



# SNAP (Service Negotiation and Acquisition Protocol)

---

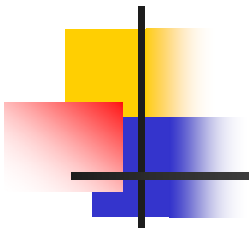
- A generalized resource management model in which resource interactions are mapped onto a well defined set of platform-independent service level agreements (SLAs)
- SNAP provides lifetime management and an at-most-once creation semantics for remote SLAs.



## Client and Owner Tussle

---

- Client needs to understand and affect resource behavior, often requiring assurance or guarantee on the level and type of service being provided by the resource
- Owner wants to maintain local control and discretion over how the resource can be used
- Owner often wants to restrict how much policy information is exposed to clients
- Service Level Agreement (SLA): A common means for reconciling these two competing demands.

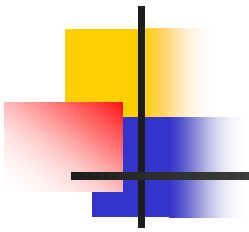
- 
- 
- SLA allows clients to understand what to expect from resources without requiring detailed knowledge of competing workloads or resource owner's policies
  - Problem: providing SLA across all diverse desired resources is not possible (e.g. in different administrative domains)



## Proposed types of SLAs

---

- Task service level agreements (TSLAs): negotiates for the performance of an activity or task. Characterizes a task in terms of its service steps and resource requirements
- Resource service level agreements (RSLAs): negotiates for the right to consume a resource. Can be negotiated without specifying what activity the resource will be used for
- Binding service level agreements (BSLAs): negotiates for the application of a resource to a task. E.g. a promising network bandwidth might be applied to a particular TCP socket
- BSLA associates a task defined by TSLA with RSLA and the resource capabilities that should be met by exploiting the RSLA

- 
- 
- SNAP is a protocol for managing the process of negotiating access to, and use of, resources in a distributed system.
  - It defines a general framework within which reservation, acquisition, task submission, and binding of tasks to resources can be expressed for any resource in a uniform fashion