

Resource Usage Policy Expression and Enforcement in Grid Computing

Jun Feng, Glenn Wasson and Marty Humphrey

Department of Computer Science
University of Virginia

(jf4t, gsw2c, humphrey)@cs.virginia.edu

Abstract -- To date, not enough attention has been paid to issues surrounding the description and enforcement of policies for controlling Grid resources. These policies define the permitted or desired usage scenario(s) allowed by resource providers, virtual organizations, or even the governing body for an entire Grid. Most existing Grid systems have either “in-spirit” usage policies with no actual enforcement (e.g., all resource providers are assumed to contribute in kind), or have implicit resource usage policies whose intent can only be manifested by examining the ad-hoc policy enforcement. Moreover, systems that *do* define some resource usage policies typically consider only CPU resources, without mentioning other Grid resources such as disk and bandwidth. Unless sufficient resource usage policies and enforcement mechanisms are created, resource providers will be increasingly reluctant to participate in Grids out of fear that their local resources will be overrun. In this paper, we identify the requirements for a resource usage policy language, and then propose an event-centric model by which to implement these policies. We describe the language structure, its implementation on top of the XML access control language XACML and a policy service that processes the language. Because decisions based on this type of policy typically require information from outside the security context of a single Grid request, we extend XACML for general timer-based and event-centric processing necessary to enforce such Grid resource usage policies. We evaluate our prototype implementation on a Grid consisting of three data repositories by showing that a usage policy-controlled Grid environment can be achieved with only minimal overhead.

I. INTRODUCTION

The combination of implicit and explicit policies govern resource usage in today’s Grids. The multi-institutional nature of grids means that it is typically not possible to set a single policy for the Grid as a whole. Instead, the overall policy is the aggregate of the policies defined by resource providers as well as the virtual organizations (VOs) to which they provide resources. While a VO is typically viewed informally as a collection of users and resources addressing a common problem or purpose, another definition is that the VO defines what is shared, who is allowed to share or consume, and the conditions under which sharing occurs [1].

Policy in this context is broader than simply access control or authorization policy. For example, a storage resource provider may want to provide at most 20% of its disk space to

a particular VO. Another example is that a VO may want all of its participants to perform roughly an equal share of the VO’s workload (jobs, storage, services, responsibilities, etc.). These two policies deal with the “usage” of resources, instead of the “access” to resources. Such *resource usage policies* cannot be expressed and enforced using existing Grid access control systems, which largely consider either only the requestor’s identity (e.g., Grid-mapfile) or VO membership (e.g., VOMS [2]). Additionally, such resource usage policies cannot be enforced on strictly a per-authorization decision basis. For instance, a site administrator may want to suspend all ongoing Grid computation tasks if someone locally logs into the machine, or a storage resource provider may want to purge all Grid data that is 7 days old at 3:00AM every day. These usage policies cannot be enforced simply by allowing or denying the *next* request from a Grid client.

Arguably, the focus of the Grid community to date has been the creation of productive and robust *mechanisms* (e.g., for Grid data discovery and movement, remote execution, etc.) but too little attention has been placed on explicit *policy* creation, management, and enforcement. Resource providers in most existing Grid systems have implicit resource usage policies (with ad-hoc enforcement mechanisms). Providers that *do* have some support for resource usage policies typically consider only CPU resources, and largely ignore other resources such as disk and networks, probably because enforcement mechanisms for other types of resources do not readily exist [3][4]. While policy-based control for resource providers can exist albeit with very limited capabilities, it is typically even less for VOs. In practice, a VO is a combination of a common software package on particular resources and a “VO server” (e.g., VOMS) that issues tokens to those users attesting to their membership in the VO. In such VOs, the “operational policy” of the VO – which defines the responsibility of resource providers and users to contribute and consume respectively – is, at best, an informal agreement (e.g., an Acceptable-User Policy, or AUP) that lacks a concrete connection to the underlying Grid software. We believe that unless sufficient software support for explicit usage policies for resource providers and VOs is researched and developed, resource providers will be increasingly reluctant to engage in Grid-level collaborations for fear of discovering either in real-time or after-the-fact that it can be very difficult to limit the consumption of their resources by Grid users.

This material is based upon work supported by the National Science Foundation under Grant No. SCI-0438263 and by the Department of Energy Early Career Principle Investigator (ECPI) Program (PI: Marty Humphrey).

In this paper, we propose an event-based policy model by which to describe and implement resource usage policies for Grids. Our policy language leverages and extends the capabilities of XACML [5] to include timer-based and event-centric processing. By supporting user-defined customizable attributes, our policy language is independent of domain-specific knowledge and is sufficiently flexible to express policies for different resource types and application domains. Our implementation consists of a policy engine to interpret policies and a policy service to store and manage policies.

The main contributions of this paper are:

- The identification and enumeration of resource usage policy requirements through careful analysis of the characteristics of some (implicit) Grid usage policies
- An event-based policy model to express usage policies for Grids
- Implementation of this policy language and enforcement mechanism that leverages XACML.
- Evaluation of this policy system showing that a resource usage policy-controlled Grid can be achieved with minimal overhead and complexity.

The rest of paper is organized as follows. In Section II, we detail the requirements of resource usage policies. We then present the language model in Section III and the design of a policy service in Section IV. Section V contains the evaluation of the policy engine and the policy system. Section VI is the related work and Section VII concludes.

II. RESOURCE USAGE POLICY REQUIREMENTS

Large-scale grid environments are complex and can involve many users, data and computational resources, network channels, and administration domains. This complexity can make it difficult to describe all of the entities and relationships required to provide usage policies within such systems. In this section, we use a representative small Grid environment to introduce some possible usage policies to drive the discussion of requirements for a usage policy language.

This representative Grid environment consists of three independent organizations within a university setting, Information Technology and Communication (ITC), the Computer Science Department (CS) and the Physics Department (Phys). These three organizations each contribute some of their resources to form a Campus Grid. ITC provides the public computing infrastructure of the campus, including student lab desktops and public clusters, while CS and Phys each has a separately owned cluster. All three organizations have data repositories in their domains to host data staged in/out of their clusters. In addition, temporary files can also be stored on ITC public machines for a short period if sufficient security mechanisms are employed.

In this environment, the resource providers ITC, CS and Phys might want to implement policies such as:

P1: Provide at most 20% of the storage space of disk C on machine “opteron8” for Grid use.

P2: Purge all Grid data over 7 days old under /scratch every morning at 3:00AM.

P3: Maximum upload/download speed limit per session is 20Mb/s, maximum number of concurrent sessions is 5, and maximum of 2 parallel streams per session.

P4: Admit new jobs only if the average CPU utilization in the past 5 minutes is under 70%.

P5: If someone logs into an ITC public desktop, suspend all ongoing Grid tasks on that machine.

In addition to these resource-specific usage policies, resource providers can have service level agreements (SLAs) amongst them when they form a Grid. These SLAs can be interpreted as site-level policies governing (the sharing of) all the resources inside an organization. Some possible organization policies are:

P6: ITC will provide up to 80GB storage space to CS and Phys.

P7: Data from CS and Phys will be replicated at two different locations inside ITC.

P8: CS will provide up to 40GB UNIX-based storage space to Phys and ITC.

P9: Phys will provide up to 40GB Windows-based storage space to CS and ITC.

There can also be policies that govern the entire VO (ITC, CS and Phys). Consider the VO policies from [6]:

- Each site receives VO utilization “credit” equivalent to the resource utilization they provide to other sites. This credit can then be “spent” for resource utilization in the VO (the *you-get-what-you-give* (ygwyg) policy).
- VO workload should be divided equally among the participating organizations (the *1/N* policy).

The “ygwyg” policy implies that an organization can utilize as much of the VO’s resources as they wish provided they “repay” the VO by providing access for other participating organizations to the resources they control. The “1/N” policy says each organization should perform 1/Nth of the work of the VO. With regard to storage resources, two possible variations of these policies are:

P10: For each participating site, the ratio between the consumed grid storage space and contributed grid storage space should be within the range of (0.5 – 2) and can be unbounded for up to one hour. For a particular site A, the consumed Grid storage space is calculated as the space occupied by data that originated from A and stored on other sites; the contributed storage is calculated as the total storage space that A has allocated for the Grid.

P11: In a Grid with N participating organizations, for each organization, the total space occupied by Grid data on the organization’s resources divided by the total Grid data in the VO should be within the range of (0.9/N – 1.1/N) and can be unbounded for up to one hour.

Through discussions with Grid deployers and operators, we believe that these policies are representative of policies that many Grids would *like* to implement but currently cannot make operational. From these policies, we derive common characteristics of usage policies.

First, many of these usage policies require information outside of the security context of a request to evaluate. For instance, to evaluate policy P1, a Policy Decision Point (PDP) will need information such as the size of the incoming file and the space consumed by the Grid on local disk. For policy P4, a PDP will need historical CPU usage information. In other words, a PDP may need to refer to many different kinds of information when making a decision. This means that any such *policy language* must be extensible in that it must be able to describe resource- or domain-dependent information, as well as how a PDP can gather that information (since most information will not be present in a client’s request message).

Second, the enforcement mechanisms can be substantially different for various usage policies, and as such, interactions between a PDP and the policy enforcement point (PEP) can be different as well. Some policies can be enforced using authorization decisions, i.e., by permit/reject certain users/operations on resources when the usage limit is reached. Policies P1, P6, P8 and P9 fall into this category. This type of usage policy fits nicely into the request-response access control model. Other policies, such as P2, P10 and P11, can be enforced by periodically executing management tasks to restore the system to a state consistent with the policy (by, for example, moving or deleting files). This type of policy requires that any such policy language allow the specification of time windows within which policy statements must be true. Policies such as P3 (partly), P5 and P7 can only be enforced by taking action when situations occur after a requested operation has begun. In other words, it cannot be determined at request time if a request will violate usage policy. For policy P3, P5, and P7, these “situations” could be, respectively, that the user’s transfer rate has reached 20Mb/s, that a user has logged-in, or that a file has been uploaded. This type of usage policy requires the policy language to be able to describe “events” that the PDP or PEP can receive and the effect of those events on policy evaluation/enforcement.

Third, usage policies can be either configuration policies or conditional policies. Configuration policy defines an intended configuration of the Grid system. The goal of enforcement of this type of policy is to maintain that state. Conditional policies, on the other hand, require the policy system to monitor for specific trigger conditions and take certain actions when those occur. Once these actions are taken, the policy is satisfied. In other words, conditional policies provide a goal state that must be reached, while configuration policies provide a goal state that must be maintained. Policies P1 - P4 are configuration policies, while policy P5 is a conditional policy. A resource usage policy language must be able to express both configuration and conditional constraints.

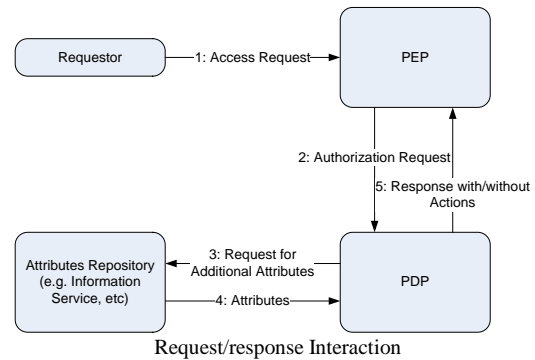
III. DESIGN OF THE RESOURCE POLICY LANGUAGE

To meet these requirements, we need the resource usage policy language to have the ability to convey access control policy as well as the notion of events and actions. In Section III-A, we first describe the details of the events and actions in our resource policy language. We note that we are not proposing a new “Grid information system”; rather, we are specifying the format and intent of the messages that could possibly originate from such Grid information systems as appropriate and used to enforce Grid resource usage policies. In Section III-B, we present the details of our resource policy language model, and explain how/why we significantly leverage XACML in the creation of our usage policy language as well as its subsequent implementation.

A. Events and Actions

Actions appear in policy documents as the operations to be performed when certain conditions occur, and events are XML representations of facts about the current state of some aspect of the policy-controlled Grid system. We discuss each of these constructs below, beginning with events.

As indicated in Section II, a resource usage policy language must be able to support both request-response and event-based interactions. Figure 1 shows information flow diagrams for these two interaction styles. Each diagram is based upon the generic policy framework of the IETF/DMTF [9]. In the “event interaction” diagram, the event reporter represents any component that collects and reports data (events) to a PDP. Delivery of this information could be direct or possibly through a publish-subscribe (pub/sub) architecture. In Figure 1, the event reporter is shown as an independent system component, although the PEP itself can be an event reporter.



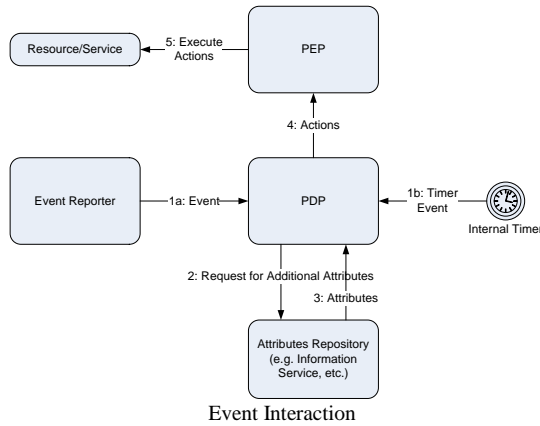


Figure 1: Data Flow Diagrams for Request/Response and Event Interactions

In our language, an event is an encapsulation of relevant attributes sent to a PDP when a given situation occurs. The XML structure of an event consists of four elements:

- *<Time>* - describes when the event happened. This element must possess “occurred-at” and “timeZone” attributes.
- *<Reporter>* - describes who reported this event. *<Reporter>* must contain a “reporter-id” attribute, a URI that points to an entity in Grid.
- *<Source>* - describes the source of the event. *<Source>* must have the “source-id” attribute, a URI that points to an entity in Grid. Note that *<Source>* and *<Reporter>* can point to the same entity.
- *<Situation>* - contains the data associated with this event. This element does not have any mandatory attributes; instead, it holds custom attributes supplied by the reporter. A special “Timer Event” is an event without a *<Situation>* element.

An event reporter can supply custom attributes when an event is generated. Each attribute is described using the *<Attribute>* element, a (typed) name/value pair. The *<Attribute>* element must have an *AttributeId* attribute (the name), a *DataType* attribute (the type) and a *Value* child element (the value). This extensible system allows the policy language to avoid inventing terms for each different resource type and application domain, thus making it very flexible. The policy engine does not have to know the exactly meaning of each custom attribute. It treats them just as named data values.

An example event is shown in Figure 2. This event says “The transfer rate at resource `gsiftp://opteron1.cs.virginia.edu` is 11.23MB/s”. For this event, *<Reporter>* and *<Source>* are the same entity “`gsiftp://opteron1.cs.virginia.edu`”. A custom attribute “download-rate” is used to describe the situation.

```
<Event>
  <Time occurred-at="2007-02-19T00:00:00.0000000"
  timeZone="EST"/>
  <Reporter reporter-id="
  gsiftp://opteron1.cs.virginia.edu"/>
  <Source source-id="
```

```
gsiftp://opteron1.cs.virginia.edu"/>
  <Situation>
    <Attribute AttributeId="download-rate"
    DataType="double">
      <Value>11.23</Value>
    </Attribute>
  </Situation>
</Event>
```

Figure 2: XML Representation of a Sample Event

Now we consider actions in the policy language. In our event-based usage policies, an action is a management task to be performed on a resource when certain conditions are met. An action is defined by a function name and a set of function arguments. The XML structure of the *<Action>* element is:

- “*action-id*” attribute, which defines the name of operation to be executed when this action is fired. The PEP should maintain a mapping from *action-id* names to executable code representing particular management tasks. Note that the question of how a policy author knows which actions a PEP is capable of executing is orthogonal to the policy language itself. A PEP could, for example, use advertisement mechanisms to allow discovery of its capabilities.
- *<Arguments>* element, which contains 0 or more *<Argument>* sub-elements. Each *<Argument>* element has “*ArgumentId*” and “*DataType*” attributes and a *<Value>* sub-element.

Figure 3 shows the XML representation of an example Action. This action’s intent is to “reduce the transfer rate on `gsiftp://opteron1.cs.virginia.edu` to 10.00Mb/s”, which it does by calling the *reduce* function passing the named arguments “*Item*”, “*Resource*” and “*Target*”.

```
<Action action-id="Reduce">
  <Arguments>
    <Argument
      ArgumentId="Item"
      DataType="string">
        <Value>transfer rate</Value>
      </Argument>
    <Argument
      ArgumentId="Resource"
      DataType="anyURI">
        <Value>gsiftp://opteron1.cs.virginia.edu</Value>
      </Argument>
    <Argument
      AttributeId="Target"
      DataType="double">
        <Value>10.00</Value>
      </Argument>
    </Arguments>
</Action>
```

Figure 3: XML Representation of an Action

B. Resource Usage Policy Language Model

Upon studying our requirements, and assuming the existence of the Events and Actions as described in Section III-A, we turned our attention to the design of the resource usage policy language proper. After careful consideration, we chose to

leverage and extend XACML for a number of very important reasons, in no particular order: we preferred to leverage existing work as opposed to creating something from scratch; a significant portion of our requirements fall into the request/response model of XACML; XACML is extensible; there were at least two open-source implementations of XACML available that we could modify (Open XACML and XACML.NET); and XACML already defined a wide range of data types along with a library of functions on those data types that we can easily leverage to construct boolean combination of functions. We do note, however, that we had to significantly take advantage of XACML’s extensibility to fully meet the requirements.

The language model of our resource usage policy language is depicted in Figure 4. Each policy statement begins with a “Scope” part that defines the requests/events to which this statement applies. If the policy is a conditional policy, then the “Condition” part is included to further refine the scope. The “Effect” part is used when the policy requires a “Permit” or “Deny” decision. An “action” part is included when the policy requires response to certain event types. Note that the “Scope” and “Condition” parts form a two-stage filter. For example, “Scope” could be used to filter messages from particular sources, while “Condition” could be used to filter the data in those messages. This is similar to the multi-stage filtering used in the XACML context model. The “Time”, “Reporter”, “Source” and “Situation” policy elements represent characterizations of “events” while “Subject”, “Resource”, “Action” and “Environment” represent requests and are leveraged from XACML. Figure 5 shows a complete example resource usage policy.

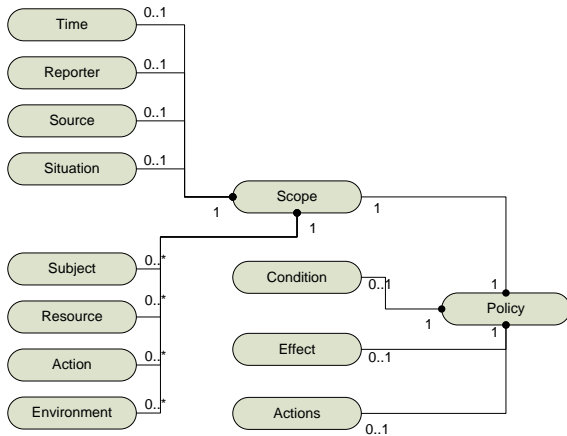


Figure 4: Policy Language Model

```

<Policy>
  <Scope>
    <Source>
      <SourceMatch function-id="anyURI-equal">
        <AttributeSelect
          AttributeId="source-id"
          DataType="anyURI"/>
        <Value>gsiftp://opteron1.cs.virginia.edu</Value>
      </SourceMatch>
    </Source>
  </Scope>

```

```

<Condition>
  <Apply function-id="double-greater-than">
    <SituationAttributeSelect
      AttributeId="download-rate"
      DataType="double"/>
    <Value>10.00</Value>
  </Apply>
</Condition>
<Actions>
  <Action action-id="Reduce">
    <Arguments>
      <Argument ArgumentId="Item"
        DataType="string">
        <Value>transfer rate</Value>
      </Argument>
      <Argument ArgumentId="Resource"
        DataType="anyURI">
        <Value>gsiftp://opteron1.cs.virginia.edu</Value>
      </Argument>
      <Argument AttributeId="Target"
        DataType="double">
        <Value>10.00</Value>
      </Argument>
    </Arguments>
  </Action>
</Actions>
</Policy>

```

Figure 5. Complete Policy Example

IV. DESIGN OF THE RESOURCE POLICY FRAMEWORK

In this Section, we describe how we designed and implement an end-to-end system built around the resource usage policy language described in the previous Section. Our design for a complete policy system includes four main components: a policy engine, a policy repository, a “plug-in repository” and a timer.

The policy engine implements the policy language and provides programming interfaces for constructing requests/events and evaluating policies.

The policy repository provides secure storage for policy documents and programmatic interfaces for instantiating policy objects from their XML representations. Besides this, the policy repository also does the first evaluation stage described in Section III-B. The policy repository also maintains the state of policies, either active or inactive. The active/inactive state allows multiple policies to be linked together to implement a more complex policy because policy actions can affect the state of other policies. Incoming requests and events are evaluated only against “Active” policies. This allows us to implement the “up to one hour” concept from the policy P11 in Section II (which is a variation of the 1/N resource sharing policy) using two policies. The first policy detects that the system is in an improper state, and the second policy corrects this improper state. The first policy is periodically evaluated when a timer-based event arrives at the PDP. When this policy detects that the Grid system is not consistent with the “1/N” policy, it generates an action to mark the second policy into the “Active” state in one hour.

The plug-in repository is used to hold pre-built software plug-ins that can determine the value of unknown attributes during the policy evaluation process. An example of a plug-in is code that is executed to retrieve/determine the amount of disk space used by each site in the VO. This plug-in could be

invoked, for example, when evaluating compliance with the 1/N policy. Each plug-in must implement a common interface defined by two functions:

```
void getAttribute(string attributeName, string  
dataType, out string value);
```

```
SupportedAttribute[] supportedAttributes();
```

The first function is invoked by the PDP to get one attribute's value using its name and data type. The second function tells the PDP which attributes a plug-in is able to fetch. The plug-in repository typically caches this information when the system initializes. In practice, all plug-ins are placed into a standard directory known to the PDP. The PDP's plug-in repository component inspects the directory, executes the supportedAttributes() function of each plug-in, then builds a name table to map attribute names to plug-ins. After this, whenever a policy is to be evaluated, and certain information is lacking, the PDP consults this mapping to determine which (if any) functionality to invoke to determine such information.

The plug-in architecture can also be used to execute some actions directly on the managed resource/service. The default behaviour in our policy system is to send actions to a PEP and let the PEP do the enforcement (Figure 1). However, one can have the plug-in interpret the actions and execute them accordingly.

One or more timer-based reporters can be deployed to enforce temporal constraints in the resource usage policies and/or periodically assess compliance with certain policies as necessary. The resource policy framework performs policy evaluation through the defined API of the compliant systems:

```
public string EvaluateRequest(string request);  
public void EvaluateEvent(string event);
```

The parameters "request" and "event" are string representations of the original request or event XML.

As mentioned earlier, during the evaluation process, a "MissingAttributes" exception is generated whenever attribute values are missing. The policy engine captures the exception, looks up the attribute name in the plug-in repository's name table to find the corresponding plug-in, and then invokes the plug-in to fetch that attribute's value. When the value arrives, the policy engine reformulates the original evaluation request including the additional attributes and begins the evaluation process again.

V. EXAMPLE SCENARIO AND EVALUATION

To make the abstract discussion of the resource policy framework from the previous Section more concrete, and to evaluate our policy system, in this Section we present an example scenario involving the use of our system and examine the performance of the mechanisms (i.e., the cost of evaluating resource usage policies in terms of system overhead). More specifically, we first study simpler policies and culminate by implementing P10 (you-get-what-you-give) and P11 (1/N) from Section II. We believe that these policies

are both implicitly being used in today's E-Science grids and therefore these experiments are useful to assess the cost of making these policies explicit.

We implemented our policy engine on top of XACML.NET [7], an open source XACML implementation on the .NET platform. XACML.NET already supports request-response based access control policies and so our modifications mainly involved support for event processing and interfaces for working with other components in the system, such as the policy and plug-in repositories. Our modified policy engine is capable of evaluating both access control and resource usage policies and thus does not represent an additional policy engine in the Grid software stack.

Our service that runs the policy engine is implemented as a .NET 3.0 Web service. The EvaluateRequest() function is a synchronous request-response function, always returning a decision to the client that generated the request message. The EvaluateEvent() function is a "one-way" asynchronous function in that event messages may or may not produce responses from the engine. Responses that do occur may not be sent to the "client" that generated the event message. A simple file-based policy repository is implemented, allowing us to easily control which policies are in effect by manipulating a configuration file.

To evaluate our policy system and engine, we set up a Grid testbed consisting of three Windows machines (each a Pentium 4 2.4GHz CPU with 1GB memory). Each machine represents a data repository from ITC, CS and Phys in the Grid environment described in Section II. On each machine, there is a GridFTP.NET [8] server, a policy service (the Web service running the policy engine) and an information service installed. We modified the GridFTP.NET server to send requests/events to the policy service and then execute any resulting enforcement actions. The information service is used as an "attribute repository" that collects system information such as the disk usage on the machine.

Experiment one measures the time taken at each stage of the policy evaluation process. Figure 6 shows the results of this evaluation. In this experiment, a request is generated by a client and sent to the co-located policy service for evaluation. Each policy used in this experiment is a version of policy P1 ("Provide at most 20% of the storage space of disk C for Grid use") scoped to a particular data resource on that machine. All data points are averaged from 20 runs.

By examining Figure 6 we can see that round trip time (the top-most line) becomes dominated by server-side processing (the next lower line) as the number of policies in the system grows. Server-side processing can be further broken down into the time to locate policies that are applicable to the request/event (first stage evaluation) and the time for evaluating these policies and generating responses (second stage evaluation). The majority of server-side processing time is spent locating applicable policies -- this time grows almost linearly as the number of policies increases. The actual evaluation of policies (second stage evaluation) remains fairly

consistent throughout the maximum tested number of policies, which is 64.

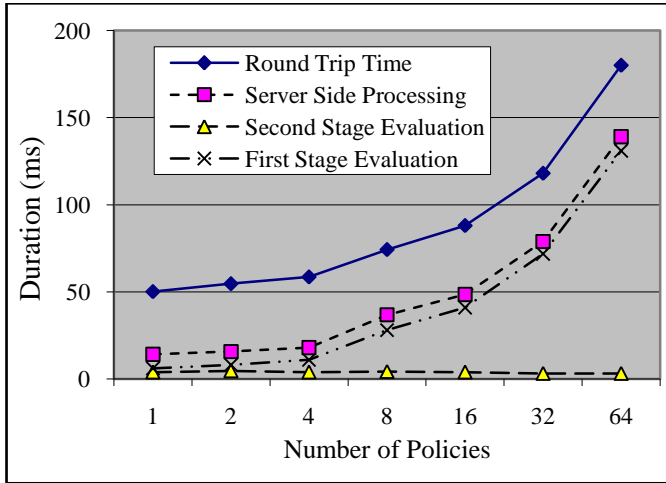


Figure 6: Analysis of the Policy Processing Time

A second method for evaluating the policy system’s overhead is to examine the impact of usage policies on small file operations. In this experiments, we measure how many 4K file uploads through GridFTP.NET on average can be finished within 60 seconds with various policy configurations. Table 1 shows the results. The first row shows the baseline performance when there is no resource usage policy being enforced in the system. When authorization policy is engaged, we see that the average number of uploads is reduced from 23 to 22.4. This is because there is now one authorization callout made by the GridFTP server, and one policy evaluation made by the policy service. The next row shows the effect of policy P1. This policy needs one additional attribute, “storage space consumed by Grid on local disk”, which is fetched by making a call out to the plug-in repository to retrieve the attribute’s value from an information service. When the attribute arrives, the evaluation restarts. Thus there are two callouts and two evaluations. Enforcing P1 reduces the average number of uploads to 21.6 when the information service is running locally and 13.9 when the information service is running remotely. Finally, we add site policies P6 through P9 to the system. Now, more attributes such as “how much space CS has provided to Phys” will be needed to evaluate these policies. This creates additional callouts, but still only two evaluations. Using these policies, the average number of uploads is reduced to 21.2 for a local information service and 9.7 for a remote service. The result shows that the dominant factor in enforcing these usage policies is the location of the information service and the cost of determining such information at run-time (if not cached or otherwise known). However, we must note that the impact of usage policies of a system’s performance highly depends on the nature of the policies. Although the policies used here can be analysed by their impact on performance, some usage policies, such as policy P2, may lack of obvious performance measure.

TABLE 1

IMPACT OF USAGE POLICIES ON SMALL FILE OPERATIONS

	# of 4K File Uploads in 60 Seconds	
	Local Info. Serv.	Remote Info. Serv.
No Policy	23	23
Authz Only	22.4	22.4
Local Usage	21.6	13.9
Local Usage & Site	21.2	9.7

Our second experiment examines two particularly important policies and their impact on the Grid testbed. These policies, P10 (you-get-what-you-give) and P11 (1/N), require information about the state of the Grid as a whole and so are more “heavy-weight” than the policies used in the previous experiment.

In this experiment, there is an event reporter that periodically collects attributes, such as “consumed Grid space for CS”, and sends them as an event to the policy service. The event is then considered in the context of the policy to decide if the Grid system still conforms. If the state of consumed Grid space is non-compliant, the policy engine will activate a second policy to enforce a correct system state in one hour. The enforcement mechanism is to migrate files (large files first) to different sites until the Grid disk usage at each site complies with the policy. Clients running on each machine upload files to the other two machines. Each client has a 75% chance to choose one desktop as a destination and a 25% chance to choose the other, to approximate the “hand scheduling” commonly used by today’s Grid users. The uploaded file size is randomly selected from 100 bytes to 10MB. Each user uploads files at a specific rate until the total allowable volume for that user is reached. Users A, B and C of ITC, Phys and CS respectively are each allowed to consume 100MB, 200MB and 300MB respectively.

Table 2 shows the total data movement volume (in MB), the data movement volume due to enforcement of policies, the total number of file operations and the file operations due to policy enforcement for three scenarios, no policy, policy P10 and policy P11. The table shows that when enforcing VO usage policies, the data movement volume increases by approximately 22% and 16%, respectively, and the number of file operations increases by 65% and 55%, respectively, for P10 and P11.

TABLE 2

IMPACT OF PARTICULAR POLICIES ON DATA MOVEMENT IN VO

	Data Movement		# of File Operations	
	Total	Enforcement.	Total	Enforcement.
No Policy	600	0	143	0
Policy P10	731	131	225	93
Policy P11	694	94	197	79

While this is a substantial increase in the number of file operations, we believe that through intelligent (and off-hours) movement of data and a service that maintains a logical-to-physical mapping for file data for individual users, the impact on users can be minimized. We assert that the benefit to the Grid overall for being able to express *and implement* such previously implicit resource usage and sharing policies far outweighs this potential cost.

VI. RELATED WORK

The IETF/DMTF common information model for the policy rules [9] defines each policy as a set of expressions of the form “if *condition* then *action*”. The event part of our policy language model does not fall neatly to this definition, although at the most abstract level, very roughly, it can be considered as close to “<event><condition><action>”. The Policy Research Group [10] at the Open Grid Forum (OGF) has also adapted this IETF/DMTF policy framework. However, to date, there has been no implementation of a policy system based on the OGF’s work.

Implicit policies are often encountered in resource management systems, especially schedulers. For example, the Maui Scheduler [11] can be thought of as a policy engine for controlling resource allocations to jobs while concurrently optimizing the use of managed resources. However, in Maui, policies can only be enforced at job submission time. Some Grid middleware, such as Legion [3] and Condor ClassAds [4], can be used to enforce resource usage policies at the individual host level. However, both Legion and ClassAds can only support CPU utilization policies.

Other meta-schedulers can use policies that cover different scopes, such as the Grid or VO level. Meta-schedulers, such as Silver [12] and CFS [13] can allow a few scheduling policies at the meta-scheduler level, such as round-robin and FIFO. CFS, in particular, can allow plug-ins to implement custom scheduling policies. Catalin et. al. [14] have identified usage policies (UP) for both sites and VOs. His research presents a policy model to express these UPs and a distributed architecture for UP-based scheduling in a Grid environment. Eric Elmroth et. al. [15] have identified resource partition policies, allowing local resource capacity as well as global Grid capacity to be logically divided across different groups of users. Wasson and Humphrey [6] focus on policies at the VO level. They identify three general policies regarding resource usage by which VOs might operate. We extend this work by providing the comprehensive framework around such resource policies. In addition, the expression of policies reported here is closer to the grammar of messages exchanged in the system and thus there was less translation between the policy author’s desired results and the statements used to achieve them.

We believe our work largely complements the goals of Autonomic Computing [16], an approach towards self-managed computing systems with a minimum of human interference. A particular instantiation of the Autonomic Computing ideas, the policy system PMAC [17], focused on automatic business policy decisions using an additional value

element in each policy. The main difference with this work is that our policy language and system focus on usage policies that regulate resource consumption by Grids, while Autonomic Computing systems generally focus on using policies to manage different configuration settings of business services in a closed enterprise environment.

VII. CONCLUSION AND FUTURE WORK

The lack of explicit resource usage policy management and enforcement is a significant problem in Grids, increasingly limiting the potential deployment of Grid technologies. Our approach to this problem is to provide a general policy model that expresses usage policies in an enforceable manner. We have identified the requirements for usage policies and presented an event-based policy model to express these policies. Our implementation of the policy language, framework that leverage XACML can provide a usage policy controlled Grid environment with minimal overhead.

There are several directions for future research. First, we can utilize emerging Web Service technologies such as WS-Resource Properties, WS-Eventing, WS-Management and languages such as SAML as standard mechanisms to publish/subscribe events, execute actions and convey attribute values. Second, the current policy language does not sufficiently address policy conflicts in which multiple (conflicting) actions are applicable in the same situation. Because resource usage policy evaluations do not always result in a “Permit” or “Deny” response, the common “deny-override” algorithm of XACML will not work. This creates an opportunity for investigating appropriate conflict identification and resolution algorithms for usage policies. Third, we are interested in providing policy management tools including policy authoring software, policy distribution protocols and other components for a comprehensive and robust end-to-end resource usage policy solution.

REFERENCES

- [1] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
- [2] R. Alfieri, R. Gecchini, V. Ciashini, L. dell’Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS: an Authorization System for Virtual Organizations, *1st European Across Grids Conference*, Santiago de Compostela, February 13-14, 2003
- [3] Andrew S. Grimshaw, Adam Ferrari, Fritz Knabe, Marty Humphrey. Legion: An Operating System for Wide-Area Computing, *IEEE Computer*, 32:5, May 1999:29-37.
- [4] Rajesh Ramen, Miron Livny and Marvin Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL
- [5] eXtensible Access Control Markup Language, Available at www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf
- [6] G. Wasson and M. Humphrey. Policy and Enforcement in Virtual Organizations. In *4th International Workshop on Grid*

- Computing (Grid2003)* (associated with Supercomputing 2003).
Phoenix, AZ. Nov 17, 2003.
- [7] XACML.NET, Available at <http://mypos.sourceforge.net/>
 - [8] J. Feng, L. Cui, G. Wasson, and M. Humphrey. Toward Seamless Grid Data Access: Design and Implementation of GridFTP on .NET. *2005 Grid Workshop (Associated with Supercomputing 2005)*. Nov 2005. Seattle, WA.
 - [9] B.Moore et.al., Policy Core Information Model – Version 1 Specification, RFC 3060, February 2001
 - [10] Policy Research Group, at Global Grid Forum
http://www.ggf.org/L_WG/wg.htm
 - [11] MAUI, Maui Scheduler, Center for HPC Cluster Resource Management and Scheduling, www.supercluster.org/maui
 - [12] Moab Grid Scheduler (Silver), Available at <http://www.supercluster.org/projects/silver/>
 - [13] Community Scheduler Framework, Available at <http://www-128.ibm.com/developerworks/grid/library/gr-meta.html>
 - [14] C. Dumitrescu, I. Foster, M. Wilde, Policy-based Resource Allocation for Virtual Organizations, *Grid Computer Workshop 2004*
 - [15] Erik Elmroth and Peter Gardfjall, Design and Evaluation of a Decentralized System for Grid-wide Fairshare Scheduling, *First IEEE Conference on e-Science and Grid Computing, 2005*
 - [16] Autonomic Computing.
<http://www.research.ibm.com/autonomic/>
 - [17] Policy Management for Autonomic Computing (PMAC) at <http://www.alphaworks.ibm.com/tech/pmac>