Agreement-Based Interactions for Experimental Science

Katarzyna Keahey[1], Takuya Araki[1,2], and Peter Lane[1]

[1] Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439 USA
{keahey,araki,lane}@mcs.anl.gov
[2] NEC Internet Systems Research Laboratories, Kanagawa 216-8555, Japan

Abstract. Enabling quality of service (QoS) in the Grids requires not only resource management strategies but also the development of protocols enabling structured negotiation for the use of resources. Such protocols will allow the creation of policies dynamically and automatically broadening the scope of Grid applications. In this paper, we describe design, implementation and application of an agreement-based infrastructure. We then discuss its use in the virtual control room developed for the National Fusion Collaboratory.

1 Introduction

Over the last decade computational Grids [1] became a very successful tool at providing distributed environments for the secure and coordinated execution of applications. More recently we have seen an increased demand for Grid technologies in areas with stringent quality of service (QoS) requirements such experimental science [2, 3]. This resulted in stronger emphasis on providing QoS in Grid technologies [4] and focus on technologies enabling it. The most recent work in Grid computing [5-8] indicates that the next-generation Grids will include policy-based resource management, a variety of authorization services, and support dynamic resource procurement as well as adaptation to changing system conditions.

If such dynamic, need and opportunity driven environment is to be achieved, it is fundamental to *establish mechanisms and protocols* enabling clients to negotiate and renegotiate policies *dynamically* rather than rely on static policy sets. Agreement services provide such mechanism. A client can negotiate with an agreement service to meet specific objectives in a Grid environment. An agreement service evaluates the client's request in the context of a potentially complex set of policies. As a result of the negotiation, an agreement is created representing a concretization of those policies to suit the client's requirements. An agreement can be subsequently renegotiated, amended, or otherwise dynamically and automatically updated.

Providing Grid services based on such agreements and managing them to automatically adjust to agreement changes allows many advantages to clients as well as providers in the Grid. An agreement-based services infrastructure combines information, negotiation and execution services that allow clients to query the availability of a particular service in the context of their priority needs, as well as to compare offers from different providers. Service providers can use agreements as a provisioning target driving resource management as well as to estimate future demand and analyze client needs. Combining multiple agreements allows for the creation of agreements of

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 399–408, 2004.

arbitrary complexity. Automatic resource management based on such agreements allows for adaptation that can both leverage and counteract the changing conditions in the Grid.

In this paper, we describe the implementation of an agreement-based infrastructure loosely based on the WS-Agreement specification [9] currently developed at the Global Grid Forum (GGF). We describe terms for specific applications including combined agreements, dependency based agreements, and agreement templates. To manage uncertainty, we associate agreements with confidence levels representing the strength of the agreement to the client. Finally, we demonstrate how our implementation satisfies a client's point of view, working under to constraints of a virtual control room developed by the National Fusion Collaboratory for use in fusion experiments.

2 Agreements: Architecture and Implementation

In this section, we first give an overview of the architecture of our system loosely based on WS-Agreement [9]. We then describe our implementation of this architecture and our definition of agreement terms used to capture agreements for services described in section 3.

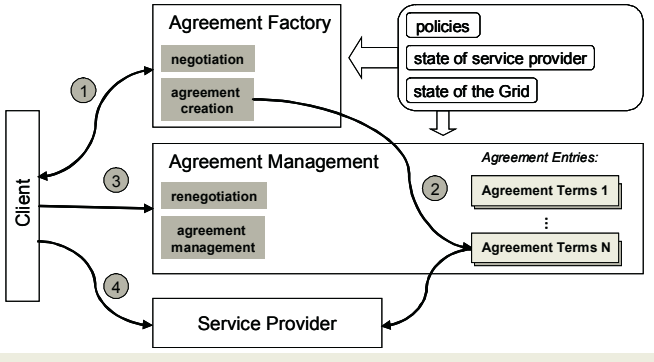

Fig. 1. Interactions in an agreement-based system

Figure 1 illustrates a basic interaction in an agreement based approach to resource management. The interaction starts with a negotiation process which can be viewed as a discovery phase in which clients advertise their needs to the agreement factory, and the factory represents what capabilities can be provided depending on policies, state of the Grid and other potential factors (1). This phase ends in the creation of an agreement when both sides commit (2). Agreements represent state that can be accessed and managed in terms of its lifetime and other properties. For example, in [9] they represented as Grid services [10] while we used an approach that is closer to Web Services Resource Framework (WS-RF) [11]. *Agreement terms* describe the objectives of a particular agreement. The client can manage (e.g., destroy or renegotiate) and monitor an agreement throughout its lifetime (3).

Once an agreement is created, it can be used to create or influence a service so that it meets specified objectives. Depending on the agreement terms, this may happen automatically (i.e., without requiring any further action from a client), or it may be

triggered by an event from a client (4). Further, a client may be required to explicitly point a provider at an agreement, or the agreement may be available to the provider through other means.

Although in practice both agreement management and the service provider may be implemented in the same service, the agreement format and interface are distinct and the same across multiple services. By standardizing it, we enable service providers to integrate agreements into their implementation model.

2.1 Implementation

We implemented agreement-based interactions using the Globus Toolkit 3 (GT3). While our implementation was influenced by WS-Agreement [9] and Web Service Level Agreement (WSLA) [12], our use case did not require a full implementation of it. Instead, we focused on defining terms and functionality required by the application and practical experiences with the system.

Instead of representing each agreement as a Grid service [10], we implemented the factory to create and maintain a table of current "agreement entries" exposed as factory Service Data Elements and managed as factory state. The factory also implemented the agreement management interface. Although based on an implementation of Grid services, this approach makes our implementation much closer to WSRF [11]; in general we found this model to be simpler and lighterweight.

Our negotiation process is simplified and emphasizes discovery. An agreement factory allows a client to retrieve an "agreement template" (based on the `AgreementTermType` in the section below) advertising some initial values of the agreements it supports: for example, a factory may support only services of a fixed description. The clients can then fill out some or all fields in this template and propose an agreement. By filling out more or fewer fields, the client can effectively ask a more or less concrete question about the availability of a specific service. The agreement may be rejected (if the terms specified by the client cannot be satisfied) by returning an exception. Alternatively, the factory can supply values for fields not filled out by the client and return it as provider's pre-committed offer together with an agreement handle identifying the agreement. Pre-commitment on the provider's side results in creating an "agreement entry" with a short expiration time that can be extended if the client commits (or expire if the client abandons negotiation). After receiving factory response, the client can either commit to the proposed agreement or try again. Our negotiation model is simpler than WS-Agreement as it does not implement multi-phase negotiations or support renegotiation once an agreement has been created. Further, it allows negotiation on the level of the whole agreement only. We also support a simpler commitment model: only the provider can pre-commit and client commit. Client's commitment extends the agreement time to the end of availability time.

Although for the purposes of our application domain an agreement should be claimed through an event (the requisite calculations are performed when the data becomes available) we decided to simplify this process in order to reduce the impact of service creation overhead on agreement claiming. Thus, agreement commitment causes the requisite application service to be instantiated as soon as its availability period starts. The client can then obtain the handle to the application service from the factory and claim the agreement from the application service which triggers the execution of desired actions.

2.2 Agreement Term Type

An agreement represents a commitment that services described by the service description will be provided during a specified time of service availability with a specified QoS (whenever applicable). At most one such service will be provided at a time, but it may be claimed multiple times as the availability period allows. Our agreement terms are described as follows:

```
<xsd:complexType name="AgreementTermType">
    <xsd:sequence>
        <xsd:element name="parties" type="tns:AgreementPartiesType"/>
        <xsd:element name="serviceInstanceHandle" type="xsd:anyURI"/>
        <xsd:element name="dependency" type="xsd:anyURI"
                minOccurs="0"
                maxOccurs="unbound"/>
        <xsd:element name="availability" type="tns:ScheduleType"/>
        <xsd:element name="expirationTime" type="xsd:dateTime"/>
        <xsd:element name="serviceLevel" type="tns:serviceLevelType"/>
        <xsd:element name="serviceDescription" type="xsd:anyType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AgreementPartiesType">
    <xsd:sequence>
        <xsd:element name="client" type="xsd:anyURI"/>
        <xsd:element name="provider" type="xsd:anyURI"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ScheduleType">
    <xsd:sequence>
        <xsd:element name="startTime" type="xsd:dateTime"/>
        <xsd:element name="endTime" type="xsd:dateTime"/><
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="serviceLevelType">
    <xsd:sequence>
        <xsd:element name="timeBound" type="xsd:duration"/>
        <xsd:element name="confidenceLevel" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
```

Listing 1. Generic term types used for any agreement in our system

The first three items of the schema correspond to the `wsa:ContextType` of the specification. They describe the parties of the agreement and include the Grid Service Handle (GSH) of the client and the provider. The `serviceInstanceHandle` element holds the GSH of the application service created as a result of the agreement. The `dependency` element contains the agreement handle(s) which the agreement is dependent on (see section 3.2 and section 3.4 for illustration).

The `availability` element defines the time period when the services specified in the agreement are available. The `expirationTime` element corresponds to the lifetime of the agreement (not the created service).

The `serviceLevel` element refers to QoS guaranteed by the agreement. The `timeBound` element describes guaranteed execution time. While some entities can

be managed in a deterministic fashion (CPU reservation for example), others are not (for example, data transfer over the Internet). In order to account for this uncertainty, the service provider accompanies the agreement terms with a confidence level with which it can provide the terms.

The `serviceDescription` element is a domain-specific element; its content depends on the service; examples will be described in the next section.

3 Agreements and Services

In this section, we describe service description terms for a few simple services and corresponding service implementations that can be used within an agreement-based framework described in the previous section. We describe how service levels were implemented and how the corresponding levels of confidence were estimated.

3.1 CPU Reservation Service

Agreements for this service allow clients to reserve a CPU resource. The `serviceDescription` term is defined as follows:

```
<xsd:sequence>
    <xsd:element name="CPUUtilization" type="xsd:int"/>
    <xsd:element name="hostname" type="xsd:string"/>
</xsd:sequence>
```

This service description is sufficient to reserve one CPU per host and possibly time-sharing that CPU with other jobs. Thus, CPUUtilization describes the percentage of CPU to be used. When the agreement is claimed, the job ID of a job to which the reservation should be applied is passed as an argument to the claim. Note, that the services description here is not related to QoS: the agreement simply states that a certain service will be provided.

The implementation of the resource reservation service is similar to GARA [6]: to implement resource reservation, this service utilizes DSRT [13], which has functionality to allocate specified percentage of CPU cycles to a certain process. The service maintains a reservation table; when an agreement is proposed, the table is first consulted to make sure there is a slot available, and if a specified percentage of CPU cycles is free during the period of availability, the proposed agreement is accepted.

To meet the needs of more complex hardware configurations, these service description terms can be extended (see [14] for a more extensive definition of terms). We are currently working on generalizing this infrastructure to allow reservations in clusters using for example the Maui scheduler plug-in in conjunction with the PSB batch system [15].

3.2 Application Execution

The terms below allow a client to make agreements for the execution of an application. The service description represents the name of the program and concrete argument values. For the specific application used in our experiment (the magnetohydrodynamics equilibrium fitting code EFIT [16]) the `serviceDescription` is as follows:

```
<xsd:sequence>
    <xsd:element name="application" type="xsd:string"/>
    <xsd:element name="timeSteps" type="xsd:int"/>
    <xsd:element name="executionMode" type="xsd:string"/>
</xsd:sequence>
```

The agreement for this service guarantees execution of a specific service description with certain service level as described in listing 1 (in this case: the execution time is bounded by a certain value). It is important that the service description externalizes all the arguments that QoS may depend on; in this example both `executionMode` and `timeSteps` influence the execution time of our application.

To meet the QoS, this service reserves CPU resources using the CPU reservation service. In the current implementation, resource reservation is made by job execution agreement factory when the level of the execution service is negotiated, but we also envision scenarios where the client can use a preexisting reservation as input to negotiation. The GSH of CPU reservation service is stored as the `dependency` element of `AgreementTermType` (see listing 1). It should be noted that the agreement does not indicate in what way or to what extent the service depends on the dependency agreement; the knowledge of how to "consume" the dependency is application-specific.

The time bound on which the service will finish, is calculated by combining information about the resource reservation and prediction of execution time based on historical data and scaled to the number of timesteps and the CPU share. The confidence level of the time bound is modeled as prediction error. Thus, although the terms of the agreement are based on resource management, they are to some extent informational, that is, the estimate of execution time is based on prediction rather than adaptive management of the application.

When the agreement is claimed, the service starts executing the job using GT3's Grid Resource and Allocation Manager (GRAM) service. The CPU is claimed by associating the job ID of the job started in this way with the reservation and the execution time of the application is monitored by the provider and reported after the execution finishes.

3.3 Data Transfer Service

The Data Transfer Service is implemented based on the GT3 reliable file transfer (RFT) [17] service and uses RFT's `transferRequest` as part of its service description. Among other qualities, `transferRequest` contains information about the source and destination of the transfer, needed to calculate QoS. The exact parameters are as follows:

```
<xsd:sequence>
    <xsd:element name="transferRequest"
        type="rft-types:TransferRequestType"/>
    <xsd:element name="size" type="xsd:int"/>
</xsd:sequence>
```

Again, estimates of execution time (transfer time, in this case) are based on a simple prediction depending on historical data for this transfer, and confidence level on associated error. Although we have explored more sophisticated ways of QoS enforcement for data transfer [18], we have not yet integrated them into this system.

Since fusion codes produce multiple files as a result of a run, the data transfer service has been customized to operate on directories of data rather than individual files: the data is tarred before RFT is invoked and untarred at destination. As with application execution, the transfer time is monitored by the provider and reported after the service finishes.

3.4 End-to-End Application Execution

Providing an end-to-end application service based on remote execution requires coordinating several subsidiary services. In our case the workflow scenario is very simple and consists of application execution and data transfer of output data. The `serviceDescription` exposes interface similar to application execution:

```
<xsd:sequence>
    <xsd:element name="application" type="xsd:string"/>
    <xsd:element name="timeSteps" type="xsd:int"/>
    <xsd:element name="executionMode" type="xsd:string"/>
    <xsd:element name="outputDestination" type="xsd:string"/>
</xsd:sequence>
```

As before, the service description externalizes arguments on which the QoS depends; in this case we add the argument describing the destination of the data to those on job execution. In this way, the end-to-end timebound calculation can be adjusted to the location in which the execution is eventually scheduled.

The end-to-end application service directly depends on the job execution and data transfer services for its service level. We currently store those dependencies as the dependency element of `AgreementTermType`. Negotiating a composite agreement is more complex as it requires the factory to in turn to negotiate subsidiary agreements. Further, dependencies between multiple components may impose an order on negotiating subsidiary agreements. The end-to-end time is calculated by combining execution times of the services in appropriate ways (in our case by adding, but in general we could use min/max, etc.) and using the confidence level of subsidiary services to calculate a weighted error. As with the other services, the workflow and its subsidiary services are instantiated when the availability period starts. Claiming an agreement on an application service will trigger claims on subsidiary services.

In principle, by externalicing the application description as a workflow rather than an opaque service we could both express a stronger dependency and subject much of what is currently embedded implementation to automatic management. While full implementation of this concept would require incorporating a workflow language into our agreement structure, we made some modest steps in that direction by for example externalizing service monitoring.

4 Case Study: Interactions in the Virtual Control Room

Our prototype infrastructure and services were put to the test in the virtual control room experiment at SC03 illustrating how Grids can be used in fusion science experiments. Fusion experiments operate in a pulsed mode producing plasmas of up to 10 seconds duration every 15 to 20 minutes, with multiple pulses per experiment.

Decisions for changes to the next plasma pulse are made by analyzing measurements from the previous plasma pulse (hundreds of megabytes of data) within roughly 15 minutes between pulses. This mode of operation could be made more efficient by the ability to leverage Grid resources to do more analysis and simulation in the short time between pulses. Hence, the ability to do time-bounded execution in the Grids is of critical importance.

The virtual control room experiment followed the script of typical experiment preparation and interaction. Before an experiment, a scientist can negotiate an agreement for the execution of a remote fusion code and request for data to be delivered to a specific location. This process allows the scientist to experiment with, and fine-tune the parameters for the execution of the code. Thus the agreement-based system is used not only to perform management actions but also to structure and automate experimental process that has grown more complex with the use of Grids.

The agreement formed in this way promises to deliver an end-to-end QoS on execution time of the service as long as the execution is requested within a certain availability window. Delivering the QoS entails combining data transfers with application execution and CPU management. At the time of the experiment, the client can request service execution against a previously formed agreement and expect it to be satisfied with the agreed on QoS.

In the experiment our implementation and services discussed earlier were used to obtain agreements and claim execution of an end-to-end EFIT application service. The agreement based execution was triggered form the SC03 show floor in Phoenix Arizona when experimental data became ready. The execution comprised: (1) reservation-based remote execution of EFIT at Princeton Plasma Physics Lab (PPPL), and (2) data transfer to the control room team at General Atomics in California. Servers of each site executed on Pentium 4 1.5GHz CPU under Red Hat Linux 7.1.

	application service	data transfer service	end-to-end execution
Measured	95 (92-99) sec	54 (52-57) sec	173 (167-180) sec
Agreement	95 sec, 90%	53 sec, 93%	172 sec, 92%

The table above shows how our actual execution values compared to what was promised in the agreement. The “measured” row shows the mean of 10 values and their range for each quality measured. The “agreement” row shows promised value and the level of confidence with which it is promised. The results show good agreement with estimated values. The overhead (difference between total execution time and sum of application execution and data transfer time) is large mainly due to the fact that while the time spent on the respective services was measured locally, the end-to-end execution time was measured from the SC show floor accumulating the high latencies of acknowledgement messages from the services. Despite that, the overall execution time was satisfactory.

5 Conclusions and Future Work

Although our implementation provides only a simple negotiation model, we found that it fulfilled the needs of our use case very well. The negotiation phase worked well as a capability discovery customized to the needs of a client. In fact, some of our

current agreements are used in "advisory" capacity and enable the scientist to do, in a structured way, what was previously done and an ad hoc manner: estimate times for codes that will be run during the experiment. Underpinning this interaction are the resource management actions ensuring the success of such preparations.

Given the dynamic and unreliable nature of a Grid environment, any guarantee must be qualified: resources may become unavailable or policies and priorities may change at any moment. Furthermore, while some qualities in the Grid can be managed (CPU reservations for example), others cannot: we cannot reserve bandwidth on the Internet or predict exactly the runtime of an application. For this reason, we have introduced *levels of confidence* used by the provider to represent the strength of a QoS guarantee. We modeled them as the probability that a certain QoS will be achieved. While this measure is correct from a provider's perspective, it is not very helpful for the client since it does not give it the means of verifying failure rate. However, with the addition of resource management it is possible to convert a provider's failure rate into a failure rate for a specific user. Such guarantee would be more appropriate from the perspective of our use case.

References

1. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.
2. Keahey, K., M.E. Papka, Q. Peng, D. Schissel, G. Abla, T. Araki, J. Burruss, S. Feibush, P. Lane, S. Klasky, T. Leggett, D. McCune, and L. Randerson. *Grids for Experimental Science: the Virtual Control Room*. in *Challenges of Large Applications in Distributed Environments (CLADE)*. 2004.
3. Pearlman, L., C. Kesselman, S. Gullapalli, B.F. Spencer, J. Futrelle, K. Ricker, I. Foster, P. Hubbard, and C. Severance, *Distributed Hybrid Earthquake EngineeringExperiments: Experiences with a Ground-Shaking Grid Application.* 13th International Symposium on High Performance Distributed Computing (HPDC-13), 2004.
4. Foster, I., *What is the Grid? A Three Point Checklist*. 2002: http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf.
5. Foster, I., C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002: Open Grid Service Infrastructure WG, Global Grid Forum
6. Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*. in *Proc. International Workshop on Quality of Service*. 1999.
7. K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, *SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems.* 8th Workshop on Job Scheduling Strategies for Parallel Processing, July 2002.
8. Pearlman, L., V. Welch, I. Foster, C. Kesselman, and S. Tuecke. *A Community Authorization Service for Group Collaboration*. in *IEEE Workshop on Policies for Distributed Systems and Networks*. 2002.
9. Czajkowski, K., A. Dan, J. Rofrano, S. Tuecke, and M. Xu, *Agreement-based Grid Service Management (OGSI-Agreement) Version 0.* https://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-Agreement_Specification/en/1/Draft_OGSI-Agreement_Specification.doc, 2003.

10. Tuecke, S., K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman, *Grid Service Specification*. 2003: Open Grid Service Infrastructure WG, GGF
11. Foster, I., J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana, *Modeling Stateful Resources with Web Services*. 2004: www.globus.org/wsrf.
12. Ludwig, H., A. Keller, A. Dan, and R.P. King, *A Service Level Agreement Language for Dynamic Electronic Services.* IBM Research Report RC22316 (W0201-112), January 24, 2002.
13. Nahrstedt, K., H. Chu, and S. Narayan. *QoS-aware Resource Management for Distributed Multimedia Applications*. in *Journal on High-Speed Networking, IOS Press*. December 1998.
14. Andrieux, A., K. Czajkowski, J. Lam, C. Smith, and M. Xu, *Standard Terms for Specifying Computational Jobs.* http://www.epcc.ed.ac.uk/%7Eali/WORK/GGF/JSDL-WG/DOCS/WS-Agreement_job_terms_for_JSDL_print.pdf, 2003.
15. Henderson, R. and D. Tweten, *Portable Batch System: External Reference Specification.* 1996.
16. Lao, L.L., H. St. John, R.D. Stambaugh, A.G. Kellman, and W. Pfeiffer, *Reconstruction of Current Profile Parameters and Plasma Shapes in Tokamaks.* Nucl. Fusion, 1985. **25**: p. 1611.
17. Madduri, R., C. Hood, and W. Allcock, *Reliable File Transfer in Grid Environments.* LCN, 2002: p. 737-738.18. Zhang, H., K. Keahey, and B. Allcock, *Providing Data Transfer with QoS as Agreement-Based Service.* submitted to IEEE International Conference on Services Computing (SCC 2004), 2004.