Quartermaster: Grid Services for Data Center Resource Reservation

Jim Pruyne and Vijay Machiraju
{jim.pruyne, vijay.machiraju}@hp.com
Hewlett-Packard Laboratories

1. Introduction

For many people, the term "grid" is synonymous with resource allocation and program execution. While it can easily be argued that this definition has never been accurate, the advent of the grid services paradigm has certainly expanded the scope of the term grid well beyond these boundaries. Nonetheless, resource allocation continues to be an important aspect of grid computing, and is what attracts many user communities. This has been increasingly the case with enterprise or application service provider (ASP) data center operators. They view the grid as a way of automating the management of their resources. Their desire is that grid enabling their environment will lead to increased utilization of assets and provide flexibility and efficiency in deployment while decreasing the cost of operation.

Grid services provide an attractive foundation for building the services required by data center operators. They build on web services technology which is often already found in these environments, support a dynamic environment as found in active data centers, and support decentralized identity management and authentication. This latter trait is increasingly important as data centers are consolidated. What were previously geographically and organizationally distributed resources are more and more being centralized into fewer locations and a small number of administrative domains. The grid's authentication mechanisms allow the operators to securely provide access to these centralized resources to remote users who may previously have maintained their own dedicated resources at a departmental or site-specific granularity.

Unfortunately, while the foundation exists, services that fully support the needs of data center operators do not exist yet. Commonly, resource management systems connected to today's grid are designed to support scientific workloads operating in a job submission or batch mode. A data center has requirements that are not met fully by such resource management systems. These requirements include:

- Support for diverse resources and complex topologies: Data centers contain a variety of resource types such as compute servers, large network attached storage arrays, and networking appliances such as routers, firewalls and load balancers. Each of these resource types has diverse characteristics and performance criteria, and must be cataloged and apportioned accordingly. Further, these resources may be interconnected in complex topologies for reasons of performance, reliability or security, which may be dependent upon the requirements of the applications running in the environment.
- Support for diverse application workload characteristics: The applications running in a data center, such as web or e-mail servers, have different demands than those usually found in a scientific domain. Their run-times are measured in weeks, months, or even years as compared to hours, days or perhaps weeks that typically characterize scientific codes. Further, during their lifetime, their resource demands fluctuate. This is often due to a predictable pattern in utilization either on a periodic (for example, an end-of-month spike in demand for payroll processing) or measurable long-term trend (such as an observed 5% per month increase in e-mail storage). Certain events, such as an e-mail virus may also cause an unpredictable, short-term need for increased resources. In all these cases, we must be able to change the quantity of resource allocated to an application while it runs. These applications also need guarantees, or reservations, to insure that they will be started on time.
- Security, identity management and isolation: As with any shared resource environment, data centers have strong requirements on security and isolation among their users. In consolidation cases, these concerns are often critical when convincing users to give up local ownership of resources and move into a centralized environment. The data center operators have their own concerns in these areas, particularly the ability to

securely shift resources from one user to another without exposing the identity of one user to another or allowing any residual state, such as disk storage blocks, to be seen by the new user.

The Quartermaster research project at HP Labs is developing a resource management system that satisfies the requirements of a data center. Ultimately, we strive to support both technical and enterprise workloads in a single environment. To these ends, we use grid services as the principal method of interfacing with our users. Our grid service interface for reservation, and discussion of our experience with grid services is provided in the next two sections.

2. Grid Services for Reservation

We consider reservation of resources to be a mutual discovery and negotiation process between a user and a resource management system. By "mutual discovery" we mean that during the interaction, both the user and the resource manager will learn what the other is seeking and what it has available. By "negotiation" we mean that these interactions will often be in multiple rounds, and that we wish to maintain some context between the parties so that we can track the progress toward a reservation. We also have two important modes of operation that we must support. First, it must be possible for a single user to coordinate reservations across multiple resource management systems. This permits co-scheduling or co-reservation when a single application requires resources spread across distinct management domains. Second, it must be possible to introduce brokers into the environment. A broker is a third party that is located between a user and the resource manager. It may provide additional services such as searching across multiple domains or using its own identity or priority on behalf of the user.

2.1 Our Initial Interface

Our initial service interface is described in [1], and is heavily influenced by SNAP [2]. The interface consists of two methods: `request` and `reserve`. The user provides the request method with a full description for the reservation including the resource types and time intervals when the reservation was to be valid. The resource management system can provide either a negative or positive reply. A negative reply indicates that the request could not be satisfied and may contain an alternative set of reservation parameters, based on those originally provided, that suggest an acceptable configuration. A positive reply indicates that a reservation is possible, but it may also contain updated parameter values that differ from those originally provided. The positive reply also contains an identifier and a time-out value. If the user wishes to make the reservation, it must complete the second phase of the protocol by calling the reserve method with the identifier prior to the time-out expiring. This second phase is what allows us to satisfy the requirements for coordinating reservations across multiple domains and permits a broker to inspect multiple sites on behalf of a single user request.

The basic structure of this interface is simple, containing only two methods, but it is unsatisfactory for two reasons. First, it does not truly utilize the power of grid services. We find ourselves re-inventing patterns that the grid service infrastructure could be providing for us. Most notably, we use identifiers to name the reservations when it would have been more consistent with the grid service model to use a factory pattern. Second, the request method signature is complex because all of the parameters for the reservation must be provided on every call. We, as others, also realized that we were creating a use-specific interface for a general pattern: negotiation. WS-Agreement[1] provides this general pattern, so this is the approach we are currently taking, and describe in detail below.

2.2 WS-Agreement Overview

WS-Agreement [4] is presently a draft specification for a general-purpose negotiation approach to service management. The term negotiation here has much the same meaning as above. It involves an iterative approach to reaching a desired state for the two parties. For the purposes of this discussion, it is important to understand two structures defined by WS-Agreement: the term type and the agreement type. For a more complete description of WS-Agreement, we refer you to the specification [4].

---

[1] WS-Agreement has been known as OGSI-Agreement. The name has been changed to better reflect its applicability to web services in general and is not specific to a grid context even though it does build on other specifications defined in the Global Grid Forum (GGF).

The term type (called `gsa:TermType` in the specification) represents one value or condition that the parties agree upon. The structure and content of the term are not specified by WS-Agreement. It is up to individual services that use WS-Agreement to define content of these terms that is specific to their domain. What is specified is a model of term states and how they interact to define the state of the agreement and what future actions can be performed on the term. The first type of state deals with the current commitment level for the term. This can take on the values Required, Optional, Observed, and Ignored. These names, and to some extent their meaning, are taken from the WS-Policy [5] specification. Required indicates that the parties must come to an agreement on a value for this term, but have not yet done so. Optional means that an overall agreement can be reached even if no agreement is made on this term. Observed means that both parties have committed to the current value of the term and will operate in an agreement based on it. Finally, Ignored means that this term has no meaning or control for the final agreement. The other sort of state is the negotiability of the term either Negotiable or Fixed. A term that is fixed cannot have its value changed. A Negotiable term's value may change over time as the result of on-going negotiation, even if the agreement as a whole has been put into effect.

The agreement type (named `gsa:AgreementType` in the specification) defines the contents of the agreement. Conceptually, it defines a collection of terms and the relationship among these terms. For example, an agreement could require all terms to be decided upon (that is, move to the Observed state), or it may provide an alternative between some sub-sets of the terms that must become Observed. This facility is inherited from WS-Policy which provides methods for grouping sub-terms and making assertions about their joint state such as exactly-one, all, or one or more of the terms must be satisfied. For the purpose of WS-Agreement, satisfaction means an agreed upon value for a term in the Observed state.

Agreements are created, and their lifetime managed as other grid services. A factory is used to create the agreement, and the parameter to the service creation is the agreement type. A new grid service, representing the agreement, is created when the factory determines that the provided terms are sufficient to begin negotiation. Alternatively, an exception is returned if the terms are not understood or have unacceptable initial values or states. The service representing the agreement has a lifetime managed as other grid services, using the termination time soft state management. This allows for the multi-site negotiation and brokering as in our previous protocol. Initial termination times will usually be set to a small value during which negotiation will take place, then extended to represent the commitment to continue operation under the agreement.

2.3 Resource Reservation via WS-Agreement

While WS-Agreement provides the framework and the structure for performing negotiation, it requires terms to be defined that are specific to the domain. We have created terms that satisfy our needs for reservation of resources in a data center. These definitions are currently being used in the Quartermaster prototype. Consistent with the definitions from WS-Agreement, each of these terms extends the gsa:TermType base type.

2.3.1 Start and End Time

The terms for start and end time for the reservation are defined as follows:

```
<xsd:complexType name="ReservationStartTimeTerm">
  <xsd:complexContent>
    <xsd:extension base="gsa:TermType">
      <xsd:sequence>
        <xsd:element name="startTime" type="xsd:dateTime"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ReservationEndTimeTerm">
  <xsd:complexContent>
    <xsd:extension base="gsa:TermType">
      <xsd:sequence>
        <xsd:element name="endTime" type="ogsi:ExtendedDateTimeType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Both start and end times for the reservation are specified. This provides exact information about the duration of the reservation distinct from the agreement thus permitting the agreement to be created well before the beginning of the actual reservation. We use the ogsi:ExtendedDateTimeType for the end time which permits us to specify an infinite end time for reservations that are meant to be permanent or at least of duration that is longer than we can specify at this time.

2.3.2 Resource Description

The types and configurations of resources requested are described using the following type and term:

```
<xsd:complexType name="ResourceType">
  <xsd:attribute name="type" type="xsd:anyURI" use="required"/>
  <xsd:sequence>
    <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ResourceDescriptionTerm">
  <xsd:complexContent>
    <xsd:extension base="gsa:TermType">
      <xsd:sequence>
        <xsd:element name="resource" type="ns:ResourceType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

For data center resources, we often encounter complex topologies of resources containing a variety of basic resource types (e.g. compute servers, virtual LANS, storage arrays, firewalls, etc.). Our specification for these resources is a complex language in its own right, and is not presently defined in a XML Schema compatible manner. For this reason, we refer to the description by reference using the xsd:anyURI type and allow it to be further refined or parameterized in this use with the sequence of any attributes. This structure may change if our resource description language migrates to the XSD type system.

2.3.3 Recurrence

As noted previously, in our environment we frequently see regular patterns of resource demand. We therefore define a structure for specifying a pattern for repetition in demand. We call this pattern a recurrence. The recurrence works in conjunction with specific demand intervals, defined in Section 2.3.4, to create a pattern for demand over time. The recurrence defines the base time to which the following intervals are relative. The defined recurrence types and their corresponding term definition are:

```
<xsd:simpleType name="RecurrenceType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="NONE"/>
    <xsd:enumeration value="PERIODIC"/>
    <xsd:enumeration value="HOURLY"/>
    <xsd:enumeration value="DAILY"/>
    <xsd:enumeration value="WEEKLY"/>
    <xsd:enumeration value="MONTHLY"/>
    <xsd:enumeration value="YEARLY"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="ReservationRecurrenceTerm">
  <xsd:complexContent>
    <xsd:extension base="gsa:TermType">
      <xsd:sequence>
        <xsd:element name="recurrence" type="ns:RecurrenceType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The RecurrenceType defines the type of pattern of recurrence that we have observed. The value NONE implies that there will be no recurrence to the demand. The PERIODIC value indicates that there will be a repetition, but it will be relative to the intervals defined in the following terms, not to any intervals based on clock or calendar time.

Finally, we define a set of regular recurrence patterns based on time intervals ranging from HOURLY to YEARLY. The ReservationRecurrenceTerm simply provides a single RecurrenceType for the reservation agreement.

2.3.4 Intervals and Demand

The final component of our reservation is the specific times and resource demands at those times. We specify these as a tuple containing one point in time and a resource quantity requested at that particular time. The quantity portion of that tuple is defined as follows:

```
<xsd:complexType name="QuantityType" abstract=”true”>
  <xsd:attribute name="units" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="ProbabilityMassFunctionQuantityType">
  <xsd:complexContent>
    <xsd:extension base="ns:QuantityType">
      <xsd:sequence>
        <xsd:element name="tuple" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:attribute name="value" type="xsd:float" use="required"/>
            <xsd:attribute name="prob" type="xsd:float" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

A quantity has a unit and also some numerical value as specified by a sub-type of the abstract base quantity type. We have defined a number of quantity sub-types including a constant, a range of values, or various statistical distributions. The most interesting and powerful quantity sub-type is the ProbabilityMassFunction (PMF) which we show here. The PMF is used to specify a non-deterministic resource quantity. Its use is described in more detail in [3], but the basis for this use is that while we may observe patterns of usage, the pattern may not be absolute. There may be some variance, and we specify this variance as a set of possible values and their associated probability. It is understood that the sum of these probabilities must be one. Quartermaster’s data structures and algorithms for reserving resources work with PMFs, and therefore each reservation is actually a probabilistic reservation.

A resource quantity is bound to a point in time with the following type definition and its associated term definition:

```
<xsd:complexType name="QuantityIntervalTuple">
  <xsd:sequence>
    <xsd:element name="quantity" type="rm:QuantityType"/>
    <xsd:element name="intervalStart" type="xsd:duration"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ReservationIntervalsTerm">
  <xsd:complexContent>
    <xsd:extension base="gsa:TermType">
      <xsd:sequence>
        <xsd:element name="intervals" type="ns:QuantityIntervalTuple" minOccurs="1"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Each interval is defined by its start time and the quantity of resources required during that interval. The start time is specified using the xsd:duration type. This essentially provides a time offset which is relative to both the beginning of the reservation and the beginning of the recurrence type. So, for example, a start time of zero, with a recurrence type of HOURLY would be setting the resource level at the beginning of every hour starting at the first hour boundary after the reservation’s start time. An interval start time that is greater than the recurrence length is simply ignored. Any number of interval-quantity pairs can be specified in the term. If the PERIODIC recurrence type is used, the last of these pairs must have a quantity equal to zero specifying the end of the period.

Finally, a complete reservation request takes one term of each of the types defined previously as follows:

```
    <xsd:complexType name="ResourceReservationType">
      <xsd:complexContent>
        <xsd:extension base="gsa:AgreementType">
```

```
        <xsd:sequence>
          <xsd:element name="resourceTerm" type="ns:ResourceDescriptionTerm"/>
          <xsd:element name="startTimeTerm" type="ns:ReservationStartTimeTerm"/>
          <xsd:element name="endTimeTerm" type="ns:ReservationEndTimeTerm"/>
          <xsd:element name="recurrenceTerm" type="ns:ReservationRecurrenceTerm"/>
          <xsd:element name="intervalsTerm" type="ns:ReservationIntervalsTerm"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

We do not, at present, provide any preference or alternative options in the reservation. It must specify exactly one term of each of the types. Usually, the simple case is easily defined. For example, a simple reservation for a constant number of resources from a start time to an end time would contain the resource description, the start and end times, a recurrence type of none, and an intervals term containing a single pair with the quantity (using the constant quantity sub-type which is not shown here) and an interval start-time of zero.

3. Reflections and Expectations

Our use of grid services is both consistent with and different from what we've commonly seen in other resource management systems. The data center is a diverse and complex environment in terms of the resources available and the applications than run there. It is clear, however, that grid services are helping us to meet this challenge. In particular, even the early development of WS-Agreement which itself builds heavily on grid services, has been an advantage as it addresses exactly the sort of problem that we face with regard to negotiation of reservations. Negotiation is a powerful, but also uncommon approach whose usability has not yet been proved. It may be that additional tooling will be needed to make it tractable. We will continue to be active in the development of WS-Agreement and expect that our reservation service will both evolve with it, and continue to exploit more of its features.

One of the challenges we presently face is determining how best to expose the resources Quartermaster manages. Just as we allow requests to specify constructions of resources, we wish to expose useful constructions. It is clear that this is a match for service data elements defined by grid services, but we have not yet defined a schema for exposing the environment in this manner. Also, while the authentication mechanisms provided seem to be key, we have not yet explored them in practice. We anticipate this being key both operationally to make the system more palatable to operators as well as a basis for policies involving priorities among users.

Ultimately, we hope that resource reservation will continue to be an area of standardization and collaboration based on grid services. The ideal situation is that a single service definition for reservation can be used in a variety of computing and application environments as well as support the needs of those that must coordinate resources across multiple domains or via intermediaries such as brokers.

References

[1] Rolia J., Pruyne J., Zhu X. and Arlitt M., "Grids for Enterprise Applications," in Proceedings of the 9th workshop on Job Scheduling Strategies for Parallel Programs. Seattle, June 2003.

[2] Czajkowski K., Foster I., Kesselman C., Sander V., Tuecke S., "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," in Proceedings of the 8th workshop on Job Scheduling Strategies for Parallel Programs, 2002, LNCS vol. 2357, pp. 153-183.

[3] Rolia J., Zhu X., and Arlitt M.: "Resource Access Management for a Resource Utility for Enterprise Applications," in Proceedings of the International Symposium on Integrated Management (IM 2003), March, 2003, pp. 549-562, Colorado Springs, Colorado, USA.

[4] Czajkowski K., Dan A., Rofrano J., Tuecke S. and Xu M., "Agreement Based Grid Service Management (OGSI-Agreement)," URL: https://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-Agreement_Specification/en/1/Draft_OGSI-Agreement_Specification.doc

[5] Box et. al., "Web Services Policy Framework (WSPolicy)," URL: http://www-106.ibm.com/developerworks/library/ws-polfram/