RC22650 (W0211-226) November 22, 2002
Computer Science

IBM Research Report

Meeting Service Level Agreements In a Commercial Grid

Avraham Leff, James T. Rayfield
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Meeting Service Level Agreements In a Commercial Grid

Avraham Leff

IBM T.J.Watson Research Center

<avraham@us.ibm.com>

James T. Rayfield

IBM T.J.Watson Research Center

<jtray@us.ibm.com>

Abstract

In this paper we give a definition of *commercial grids*, and identify the need to define and satisfy *service level agreements* as a requirement of commercial grids. We show that service level agreements impose unique requirements on a commercial grid infrastructure, specifically the need for a dynamic offload infrastructure. We discuss the requirements that such an infrastructure must meet, and then describe a prototype implementation in detail.

Keywords

Commercial grid, service level agreements, sla, dynamic offload

Introduction

Defining a Commercial Grid

We have identified several definitions of the term *commercial grid*, as opposed to standard *grid* [2] [3].

1. "Commercial grids charge for the service of hosting customer applications."

   Customers paying for a grid-hosted application will have different expectations than users of a free grid-service. In contrast to commercial grids, standard grids are used in university or national lab environments that do not run a grid "business". In addition, some companies are

experimenting with internal "intra-grids"; these resemble standard grids in that no real money changes hands in exchange for the service.

2. "Commercial grids coordinate *simultaneous*, in contrast to *sequential*, sharing of resources."

   This definition emphasizes the challenge of managing competitive demands for the same set of resources in environments where explicit negotiation between the grid's users is impractical. The assumption is that commercial grid customers will not tolerate being denied service or being rescheduled to some other time slot.

3. "Commercial applications (e.g., on-line transaction processing, e-commerce) run on a commercial grid; scientific applications (e.g., numerical computation) run on a standard grid."

   This definition focuses on application *type*, rather than (as the first two definitions do) on application *requirements*.

We believe that the last definition is accurate only to the extent that current grid applications are "non-commercial" according to both requirements specified by the first two definitions. For example, typical scientific grid applications include remote access to specialized scientific facilities; pooling of computing power to solve large numerical problems; and distributed analysis of large amounts of data. Currently, such applications are deployed in environments that do not charge "real" money for the service (definition 1). Such applications can coordinate shared usage of the grid fairly easily (definition 2) because the customer set is relatively small, and because sharing can be done by forbidding simultaneous access to the resource. However, as grid usage becomes more prevalent, we expect that the requirements of some "scientific" applications will require deployment to a commercial grid. As an example, consider the customer requirements for grid-deployed Monte Carlo simulations for risk analysis. The grid service-provider will likely charge for the service, and customers will expect, in return, that the service is always available (definition 1). Similarly, customers of the grid-service cannot be expected to negotiate among themselves about the order or the priority in which they will be served (definition 2). The hosting grid must transparently manage system resources such as available servers and network bandwidth so that simultaneous customer requests are met to everyone's satisfaction. Thus, although a "scientific" application, customer requirements imply that this application be deployed to a commercial grid.

While not using the the third definition, this paper does use the first (customer/provider expectations) and second (nature of customer sharing) definitions of commercial grid. However, under either definition, no hard and fast distinction exists between commercial and standard grids: the issue is only one of emphasis.

Commercial Grids & Service Level Agreements

Customers of a commercial grid are more likely to require a *service level agreement* (or *SLA*) than the standard grid customer. Because a commercial grid must satisfy simultaneous, paying,

customer demand for shared resources, a formal agreement is needed to explicitly specify what the grid will supply to a given customer. Informal agreements such as "you get to use these machines in off-peak hours" do not suffice when the customer is paying to use the machines and when other customers are using those machines at the same time. SLAs may be expressed in terms of metrics such as the maximum application response-time or minimum application throughput that will be provided to the grid customer.

Commercial Grids & Dynamic Offload Capability

There are several possible ways for commercial grids to meet their SLAs. In one approach, they can make sure that each customer is provisioned with sufficient resources, e.g. CPU and bandwidth, to fulfill the worst-case scenario guaranteed by the SLA. In certain environments, this approach can be very inefficient because traffic to busy web-sites exhibits considerable "burstiness", due to requests from different geographical areas or due to daily and seasonal variation[4]. To deal with worst-case burstiness situations a grid would have to keep large amounts of spare capacity available, with much of it unused for large periods of time. The cost of providing the grid service will therefore not be competitive.

Alternatively, the grid provider can arrange to statically offload the demand onto idle equipment using a manual process. This reduces the cost of provisioning each customer, but has a number of disadvantages. First, the reprovisioning time will be fairly slow, because manual intervention is required. This increases the chances of SLA violation. Second, the reprovisioning may result in a temporary service interruption, itself another likely SLA violation which is unacceptable in a commercial environment. Finally, this solution is labor-intensive, another cost factor.

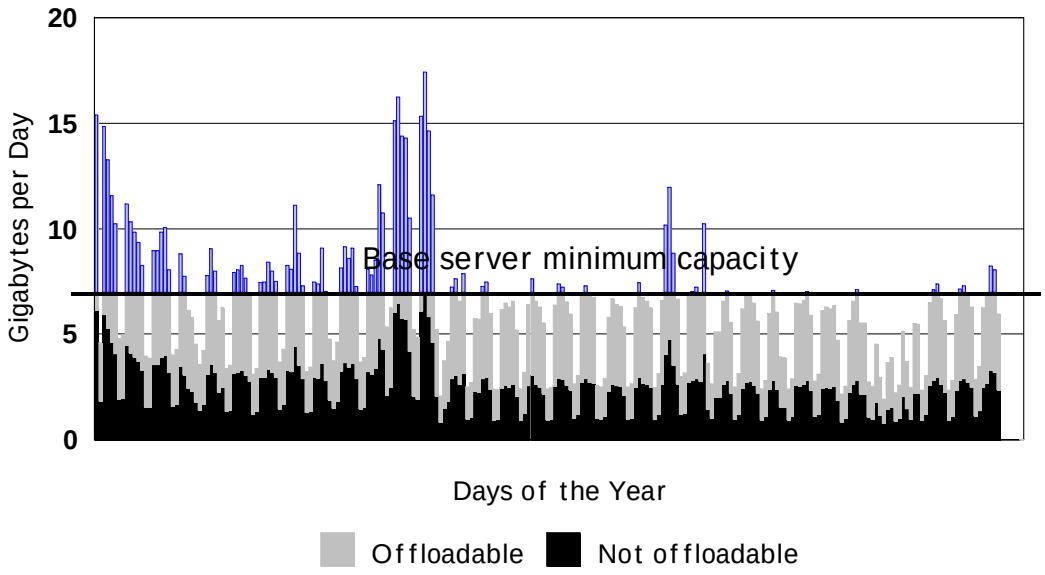

Figure 1: Server Load Over Time

A much more reasonable alternative is to use a dynamic, automated, offload capability. This approach can deal efficiently with situations similar to that shown by Figure 1. This shows the variation in a large company's workload over the course of a single year. The darkly shaded, bottom, section represents the portion of the workload that cannot be offloaded due to such considerations as security. At a minimum, therefore, the server must be provisioned with capacity to meet this demand. The lighter-shaded, upper, section of the figure represents workload that need not be satisfied by the server, and that could therefore be offloaded to other servers. The horizontal line denotes the portion of the workload that can be met by the server's required minimum capacity. All load positioned under this line should not be offloaded since the server already has the capacity to meet user demand. The interesting part of the figure is the load positioned above the horizontal line. This represents "peak" load imposed on the server on the busiest days of the year: this load cannot be met by the server's minimum capacity. One way to meet this peak demand is to greatly increase the server's capacity. The server could then avoid offloading any work; but, on the other hand, the extra capacity will remain unused for most days of the year. A more efficient approach is for the server to dynamically offload peak demand to other servers, with no need to over-provision its capacity for less busy days.

The situation of fluctuating demand on a single company's servers is magnified in a grid environment that hosts many applications. In the grid environment, the potential magnitude of peak-demand for cpu and bandwidth is magnified; worst-case provisioning means that large amounts of capacity will be wasted. Dynamic offload means that the grid can maintain a smaller pool of unused capacity that can be dynamically shared among customers as needed.

Note that, in a non-commercial grid, dynamic offload is less of a requirement. If customers are not paying for service, it's harder for them to demand better service. Similarly, the grid-provider is much more likely to do "worst-case" provisioning since they're not operating in a cost-competitive environment. This dichotomy is also true with respect to the other commercial grid definition. When grid customers coordinate resource sharing through a policy of sequential access, the pool of resources is fixed, and customers are simply encouraged to make the best use of the available resources. In a commercial grid where the grid provider is penalized when it does not meet its service commitments, dynamic offload is crucial to meeting SLAs. In Section 2 we list the requirements of a generic dynamic offload infrastructure. In Section 3 we discuss a prototype implementation of this infrastructure in detail.

Requirements of a Commercial Grid's Dynamic Offload Infrastructure

In this section we list, and motivate, the requirements of a dynamic offload infrastructure: these requirements are motivated by the grid-provider's need to efficiently meet SLAs. Specifically, we focus on the dynamic offload infrastructure needed to meet SLAs related to varying workload conditions. We assume that a pool of servers and bandwidth exists from which the commercial grid can draw resources under high load conditions, and to which it returns resources when the load decreases. The components include:

1. Ability to formally define a service level agreement.

   SLAs must be defined in a manner that allows for as little ambiguity as possible. Ideally, a SLA's definition should be directly readable by the code that predicts or detects violation of the SLA (see below).

2. Ability to predict that a SLA violation will occur, or at least to detect that a SLA violation has occurred. Otherwise, the grid cannot respond to the SLA violation.

3. Ability to transparently, and dynamically, scale up the resources in use (e.g., increase the number of servers) in response to a SLA violation (e.g., increased workload). This capability can be broken down further:

   a. Acquire the resource from a common resource pool.

   b. Provision the resource with the necessary software and configuration data. For certain resources, such as bandwidth, this step is fairly simple. For other resources, such as application servers, this step can be quite complex.

   c. Transfer load to the newly added resource.

4. If and when the workload decreases, the commercial grid must be able to transparently return resources to the pool so that it be used to meet another grid customer's requests.

Prototype Dynamic Offload Infrastructure

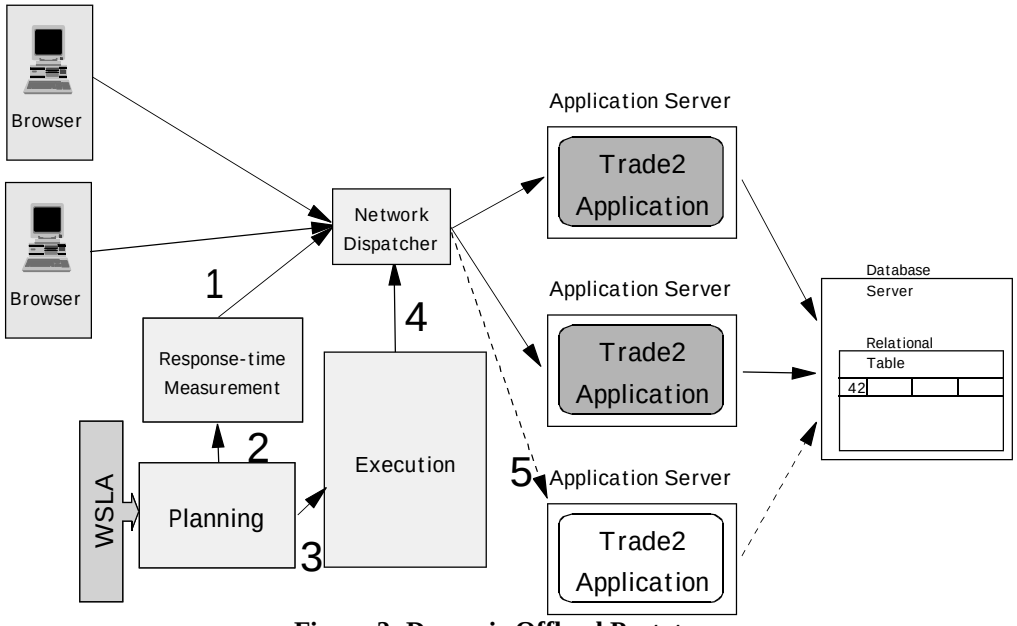

Figure 2: Dynamic Offload Prototype

We have built a prototype that implements the above requirements for a commercial grid's dynamic offload infrastructure. We downloaded *Trade2*, a J2EE application publicly available from IBM [9] and deployed it to our prototype. As described in its documentation, Trade2 "models an online brokerage firm providing Web based services such as login, buy, sell, get quote and more". We added a client-side HTTP load-generator program to enable us to dynamically adjust the load, defined as the number of concurrent requests to Trade2. As shown in Figure 2, multiple web-clients access Trade2; the application is running on two active application servers in the grid that access a shared database. The prototype infrastructure manages a single resource pool: a set of servers, each of which is pre-configured to run the application.

Defining the SLA

We use the WSLA framework [5] to define SLAs using XML. One advantage is that the SLA definition is considerably less ambiguous than a natural-language specification.

Example 1 shows the grid-provider's obligation to the customer with whom it has contracted to host Trade2. The xml states that the grid guarantees that no client interaction will exceed 200 milliseconds. Example 2 states that a violation notification notice should be sent to registered participants whenever a client interaction does exceed the specified threshold. Trade2-specific code is registered to deal with such SLA violations (see below).

```
<Obligations>
  <ServiceLevelObjective name="maxResponseTime">
    <Obliged>ACMEProvider</Obliged>
    <Validity>
      <Start>2001-11-30T14:00:00.000-05:00</Start>
      <End>2003-12-31T14:00:00.000-05:00</End>
    </Validity>
    <Expression>
      <Predicate xsi:type="Less">
        <SLAParameter>ResponseTimeSLA</SLAParameter>
        <Value>200.0</Value>
      </Predicate>
    </Expression>
    <EvaluationEvent>NewValue</EvaluationEvent>
  </ServiceLevelObjective>
</Obligations>
```

Example 1: Defining a Response-Time Obligation

Both the "obligation" and "action guarantee" contain references to WSLA elements defined elsewhere in the document: e.g., `ResponseTimeSLA`. Note that the SLA does not specify what actions should be taken in response to a SLA violation. SLAs are a contract between the

```
<ActionGuarantee name="increaseServers">
  <Obliged>ZAuditing</Obliged>
  <Expression>
    <Predicate xsi:type="Violation">
      <ServiceLevelObjective>maxResponseTime</ServiceLevelObjective>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
  <QualifiedAction>
    <Party>XInc</Party>
    <Action actionName="notification" xsi:type="Notification">
      <NotificationType>Violation</NotificationType>
      <CausingGuarantee>maxResponseTime</CausingGuarantee>
      <SLAParameter>ResponseTimeSLA</SLAParameter>
    </Action>
  </QualifiedAction>
  <ExecutionModality>OnEnteringCondition</ExecutionModality>
</ActionGuarantee>
```

Example 2: Defining the Response to a Max-Response-Time Violation

customer and the grid-provider. The SLA specifies *what* the problem is. It is the grid-provider's responsibility to *fix* the problem.

Monitoring & Enforcing the SLA

Another reason to define SLAs through the WSLA framework is that our system can benefit from the WSLA runtime that includes several SLA monitoring services [11]. These monitoring services receive a SLA specification as input, and in response, are automatically configured to enforce the SLA.

This part of our system (see Figure 2) consists of:

- A *measurement* module that collects data about the relevant SLA metric. In our system, the metric we need to monitor is the end-to-end response time of a single client request to Trade2. We implemented this function by re-using the client code used by the load-generator program. A measurement is performed by periodically interacting with the application, in exactly the same way that an actual application client does.

  A WSLA monitoring service polls the measurement module, and, based on the observed data, determines whether an *undershot minimum-response-time* or an *exceeded maximum-response-time* event has occurred. If either event occurred, the monitoring service informs the *planning* module (below).

- A *planning* module that is responsible for determining what action to take, if any, given the observed event.

  The logic in our prototype's planning module is straightforward. If an "exceeded" event occurred, it infers that not enough servers are currently assigned to the Trade2 application. Therefore, if the server pool contains an unassigned server, it should be assigned to host the application. Conversely, if an "undershot" event occurred, it infers that the grid has assigned too many servers to the Trade2 application. Therefore, a server currently hosting the application should be deactivated and returned to the grid pool.

- An *execution* module that is responsible for allocating a server from, or deallocating a server to, the grid server pool.

The WSLA monitoring service, the Trade2 measurement module, and Trade2 planning module all execute in the same address-space; they interact *via* standard Java method calls.

Server Pool Allocation & Deallocation

As mentioned above, the dynamic offload infrastructure manages a set of servers to keep the application's response time behavior between the "high" and "low" response-time thresholds defined in the SLA. This function is implemented in the following way. All client interaction with Trade2 is routed through a *Network Dispatcher* (or *ND*) [6]. Using one of several routing algorithms (e.g., round-robin) the ND maps a *cluster* address to an individual server in the cluster set.

ND has an API that allows servers to be dynamically added or removed from a given cluster. We implemented a shallow wrapping of this API using the Apache Axis [1] toolkit, because calls to this API need to occur across address-spaces. The web-services API was defined in WSDL [10], which enables automatic generation of the application-dependent client and server stubs. At runtime, the execution module sends SOAP [8] messages which are transported over HTTP to the ND web-services wrapper. By adding or removing servers from the grid pool, the execution module dynamically spreads the workload over more or fewer servers. This enables it to tune the response-time behavior of Trade2.

Putting it All Together

Referring back to Figure 2, the dynamic offload process flow proceeds as follows:

1. A response time probe is converted to an *undershot minimum-response-time* or an *exceeded maximum-response-time* event.

2. This event is delivered to the planning module which instructs the execution module to either allocate a server from, or deallocate a server to, the grid's server pool.

3. The execution module sends the corresponding web-service call to the Network Dispatcher.

4. The Network Dispatcher redefines the Trade2 server cluster as appropriate: subsequent client requests will be processed by servers in the redefined cluster.

An important feature of our prototype is that the application is developed independently of the dynamic offload infrastructure. In fact, the application is even developed independently of the commercial grid itself. The grid simply hosts the application on one or more of its servers, and the dynamic offload infrastructure manages the grid's server pool to meet the grid's SLA commitments.

Summary and Future Work

In this paper we defined the special characteristics of commercial, as opposed to standard, grids. One requirement of commercial grids in the ability to meet service level agreements. We showed that a dynamic offload infrastructure is critical for a commercial grid to meet its service level agreements. Finally, we described a prototype implementation of a dynamic offload infrastructure, and our experiences in deploying a web-application to this infrastructure.

We made a number of simplifying assumptions for our prototype. First, that all the servers in the pool were pre-installed and pre-configured with the application server (Apache Tomcat) and application software (Trade2). Further, we start the servers manually, and must keep them running throughout the grid's operation. The dynamic offload infrastructure simply manages which servers in the Network Dispatcher cluster are configured to run the application: the infrastructure does not actively manage the servers themselves. We are currently investigating the

use of OGSA [7] to provision and start servers dynamically.

In addition, we use a very simplistic service level agreement, consisting only of "minimum" and "maximum" response time thresholds. In a real commercial agreement, the response time guarantee would only apply if the request rate remains below a certain threshold. That is, if system load exceeds a given threshold, the grid provider need not satisfy the response-time criterion. This is because customers typically will not want to pay achieve good response time under unbounded or very high load conditions.

Also, we currently react only when an SLA is violated. Ideally, imminent SLA violations should be predicted, and the system should act proactively. This requires analytical modeling of the application as a function of system resources and load, and using past load observations to predict future load.

The final challenge we are investigating is how to manage server and application allocation in the presence of multiple customers and/or multiple types of applications. For example, if there are multiple customers, how can all the SLAs be efficiently satisfied simultaneously? Further, if insufficient resources are available to satisfy all SLAs, which SLAs should be violated first? These problems are quite difficult, although some initial work has been done in this area [12].

Addressing and solving such issues will be critical if commercial grid use is to become prevalent.

References

1. *Apache Axis*. http://xml.apache.org/axis/.

2. Ian Foster, Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers. 1998.

3. Ian Foster, Carl Kesselman, and Steve Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations* . International J. Supercomputer Applications. 15. 3. 2001.

4. Arun K. Iyengar, Mark S. Squillante, and Li Zhang. *Analysis and Characterization of Large-Scale Web Server Access Patterns and Performance* . World Wide Web. 2, 1, June 1999.

5. Alexander Keller and Heiko Ludwig. *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services*. IBM Research Report. RC22456. 25-32. 2002.

6. *IBM Network Dispatcher features.* http://www-3.ibm.com/software/network/dispatcher/about/features/keyfeatures.html.

7. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.* Ian Foster. Carl Kesselman. Jeffrey Nick. Steven Tuecke. http://www.globus.org/research/papers.html#OGSA. June 2002.

8. *Simple Object Access Protocol (SOAP) 1.1.* http://www.w3.org/TR/SOAP/.

9. *WebSphere Performance Benchmark Sample (Trade 2 Application) Download.* http://www-4.ibm.com/software/webservers/appserv/wpbs_download.html.

10. *Web Services Description Language (WSDL) 1.1.* http://www.w3.org/TR/wsdl.

11. *Web-Services Toolkit.* http://www.alphaworks.ibm.com/tech/webservicestoolkit.

12. *On Maximizing Service-Level-Agreement Profits.* Zhen Liu, Mark S. Squillante, and Joel L. Wolf. IBM Research Report, RC22271, http://domino.watson.ibm.com/library/cyberdig.nsf/Home.