Specifying and Monitoring Guarantees in Commercial Grids through SLA

Akhil Sahai, Sven Graupner, Vijay Machiraju, Aad van Moorsel
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto

HPL-2002-324
November 14th , 2002*

E-mail: {asahai, svgr, vijaym, aad}@hpl.hp.com

Grid, SLA, specification, web services, monitoring, OGSA, management

Grid computing has relied on "best effort" as the guiding principal of operation. However, commercial grids need to provide much stricter guarantees. These guarantees have to be specified in terms of service level agreements and have to be monitored and assured. We propose an architecture for specifying and monitoring service level agreements to achieve the above. The architecture relies on a network of communicating proxies each maintaining SLAs committed within the administrative domain of the proxy. SLAs are either negotiated between or specified to management proxies, and they are responsible for automated monitoring of data and for triggering evaluations of the registered SLAs. An unambiguous and flexible language for formalizing SLAs is presented to achieve the above. The management proxy allows reasoning about the overall status of SLAs related to an application context across multiple administrative domains by contacting and querying involved management proxies, obtaining measurement information from multiple proxies, if needed and performing a consolidated SLA evaluation. The process of measurement collection, and SLA evaluation is automated and based on web services technology. The scenario considers HP Utility Data Center as a typical Commercial Grid deployment environment.

Specifying and Monitoring Guarantees in Commercial Grids through SLA

Akhil Sahai, Sven Graupner, Vijay Machiraju, Aad van Moorsel
Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA
{asahai, svgr, vijaym, aad}@hpl.hp.com

Abstract

Grid computing has relied on "best effort" as the guiding principal of operation. However, commercial grids need to provide much stricter guarantees. These guarantees have to be specified in terms of service level agreements and have to be monitored and assured. We propose an architecture for specifying and monitoring service level agreements to achieve the above. The architecture relies on a network of communicating proxies each maintaining SLAs committed within the administrative domain of the proxy. SLAs are either negotiated between or specified to management proxies, and they are responsible for automated monitoring of data and for triggering evaluations of the registered SLAs. An unambiguous and flexible language for formalizing SLAs is presented to achieve the above. The management proxy allows reasoning about the overall status of SLAs related to an application context across multiple administrative domains by contacting and querying involved management proxies, obtaining measurement information from multiple proxies, if needed and performing a consolidated SLA evaluation. The process of measurement collection, and SLA evaluation is automated and based on web services technology. The scenario considers HP Utility Data Center as a typical Commercial Grid deployment environment.

1 Introduction

Grid computing emerged as a paradigm of sharing resources for collaboration and resource usage optimization purposes. A **Grid** is made up of a finite set of nodes. Where a node is a system that manages a set of resources. A node may be a single system or a cluster. A resource being managed can be a network, system, or an application. Being mostly used in academic environments, "best-effort" was (and is) a sufficient policy for committing resources to users performing their computational workload.

Moving into the commercial space, businesses will be bound by commitments. Monitoring and accountability are becoming increasingly important in networked environment. "Best effort" is no longer sufficient. Thus commercial grid need to develop appropriate monitoring and metering tools for observing resource utilization, managing performance degradations, availability and other parameters. Observed data need to be validated against commitments made to consumers. Automation of these processes is highly desirable due to the scale and complexity involved.

Commercial grid products such as offered from HP, IBM, Platform, or Sun, as yet do not focus on specification, measurement and validation of measurements against commitments (contracts) as specified in terms of service level agreements.

Several problems arise. SLAs determine the kind of data to be observed or monitored in the system. At any given point of time hundreds of SLA may exist. Each SLA in turn may have large number of metrics to be observed. A formalized representation of commitments in the form of SLA documents is required, so that information collection and SLA evaluation may be automated.

Another problem is that for a given application context multiple resource providers and resource consumers are involved. This could be the case when multiple cluster grids within an enterprise form an enterprise grid and provide common interface to consumers. Consolidation of management information is required when resources are spread across geographically distributed clusters. A similar problem arises when multiple enterprise grids collaborate together to provide a common interface to consumers. Since participants in a grid are inherently distributed, SLAs are distributed as well since validation of SLAs depends on local measurements. However, the SLA management system must have the ability to combine the distributed states of SLAs providing a consolidated view in the embracing application context.

Given the fact that SLA management has made some progress in web services [4], and the fact that grid technology is moving towards web services [6] technology (Open Grid Services Architecture), the proposed solution consequently is founded in web services technology.

The paper first surveys the status of SLA in the grid. Section 3 describes a typical Grid deployment infrastructure and describes a sample request for resource allocation in such an environment. In the subsequent section, a generic Grid management architecture and the place for SLA management in it is debated Section 5 describes the SLA specification and the monitoring engine. It also describes the measurement exchange protocol for obtaining measurements from multiple sites.

2 SLA and the Grid

In a grid environment users and resource providers, often belonging to multiple management domains are brought together. Users must be given some form of commitment and assurances on top of the allocated resources, such as performance, security, availability, latency, throughput, variance etc. (sometimes also referred to as quality of service) as well as in terms of responsibilities for dealing with erroneous conditions, fail-over policies or backups and more. Those terms need to be agreed upon before use and manifested in form of a SLA.

Negotiating a SLA is an exchange (protocol) of messages between user and provider, potentially involving some form of a middleman or broker. The result is an agreement of all terms that are important for either side.

The Service Negotiation and Acquisition Protocol (SNAP) [11] provides support for the following three types of SLAs:

- resource acquisition agreements (right to use/consume a resource),
- task submission agreements (inform a resource of the existence of a task), and
- task/resource binding agreements (enabling the task to consume an agreed quantity of a resource).

These notions are primarily focused on task submission techniques used in traditional grids and are related to allocation time negotiations in terms of resources and tasks. The Quality aspects as mentioned above are not negotiated. We propose an extension of scope.

A second extension we propose is the need to not only negotiate agreements, but also to maintain them in the system for their life-span meaning that agreed upon terms must be validated during use by appropriate measurements, consolidation and comparison of those measurement against the terms manifested in the SLA. We propose a SLA monitoring engine in the context of OGSA for this purpose.

A grid node controls a set of resources. The resources are allocated, de-allocated and managed over a hosting/deployment environment. The hosting environment could be a data center. For example, HP's Utility Data Center (UDC) that provides virtualized resource infrastructures could be an instantiation of a Grid deployment environment. It is important to understand a typical hosting environment to understand how service level agreements may be specified, and monitored.

3 Grid Deployment Infrastructure

The central notion of the UDC is called a "farm", a programmable hosting environment for applications. Farms can be dynamically requested, with resources contained in them, by users submitting farm resource descriptions to UDC's resource allocation manager[2]. Since farms may contain resource topologies, a special language is used to specify resource topologies. Resource topologies include not only the sets of required resources, but also their "wiring" topology, how resources are connected by protected subnets.

Figure 1 shows an example of a hosting environment [1] that can be described and instantiated when a respective document is sent to the resource manager, and the farm with all its wired resources is instantiated without further manual involvement.

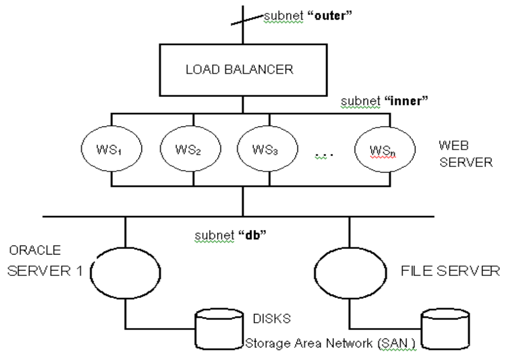

Figure 1: Resource topology of a simple two-tier application.

Alternatively, the shown resource topology can be described in Grid resource specification languages such as the Globus Resource Specification Language (RSL). Besides resources and resource attributes, topology information must be encoded in RSL. Topology information describes the connection relationships between resources. This information is needed by the UDC resource manager in order to configure resources properly. The RSL used in the example uses specific attributes understood by the UDC controller. The major extension compared to "standard" RSL is the encoding of topology information (relationships among components).

```
&( farm       (name "My-2-TierFarm") (version "1.1") )
  (subnet (id "outer") (name "outer") (ip "external") )
  (subnet (id "inner") (name "inner") (ip "internal")  )
  (subnet (id "db") (name "db") (ip "internal") )
  (lb (id "lb1") (name "lb1") (type "lb")
              (interface (name "eth0") (subnet "outer") )
              (interface (name "eth1") (subnet "inner") )
              (policy "round-robin")
```

```
          (vip (name “vip0”) (subnet “outer”)
                    (bind (id “bind1”) (name “WebTier:eth0”)
                              (virtualport “8080”) (realport “8081”) ) )
)
(serverrole (id “WebServerRole”) (name “WebServerRole”)
          (disk (target “0”) (drivesize “8631”) (drivetype “scsi”)
                    (diskimage (type “system”) (element “NT_IIS”) ) )
          (hw “cpu-x86-x2”)
)
(serverrole (id “OracleRole”) (name “OracleServerRole”)
          (disk (target “0”) (drivesize “8631”) (drivetype “scsi”)
                    (diskimage (type “system”) (element “HPUXOracle”) ) )
          (hw “cpu-pa-risc”)
)
(serverrole (id “FileServerRole”) (name “FileServerRole”)
          (disk (target “0”) (drivesize “8631”) (drivetype “scsi”)
                    (diskimage (type “system”) (element “RedHat”) ) )
          (hw “cpu-ia-64”)
)
(tier (id “WebTier”) (name “WebTier”)
          (interface (name “eth0”) (subnet “inner”) )
          (interface (name “eth1”) (subnet “db”) )
          (role “WebServerRole”)
          (minservers “5”) (maxservers “20”) (initservers “10”)
)
(tier (id “OracleServer”) (name “OracleServer”)
          (interface (name “eth0”) (subnet “db”) )
          (role “OracleServerRole”)
          (minservers “1”) (maxservers “1”) (initservers “1”)
)
(tier (id “bebop”) (name “bebop”)
          (interface (name “eth0”) (subnet “db”) )
          (role “FileServerRole”)
          (minservers “1”) (maxservers “1”) (initservers “1”)
)
```

Figure 2: RSL fragment of the shown resource topology.

In order to protect different farm instances from each other, two types of resources are virtualized for farms:

- network resources: by permitting the programmable rewiring of server machines and devices to create a virtual LAN network. Virtual wiring is achieved by programming switches connecting machines and programmatically connecting or removing machines to or from virtualized subnets, and
- storage resources: by not containing storage in machines, but programmatically attaching storage from external storage units to machines through SAN. Entire disk images with all persistent states of application environments, file systems, bootable operating system images, application software, etc. are maintained separated from machines and are attached to them during the farm instantiation process. Disk images appear on SCSI interfaces from where machines then boot images and further data.

4 GRID Management Architecture

The Grid conceptual architecture is presented in Figure 3. The Grid deployment infrastructure is where the resources (applications/software, machines, networks, farms) are provisioned. The OGSA infrastructure which is based on a web services infrastructure (.Net, J2EE based), provides the basic functionalities that deal with creation through factory, life-cycle management, obtaining management related information through manageability interfaces, and other services like notification and invocation. The OGSA meta-services are higher-level functionalities that deal with provisioning/allocation, clustering, policy specification, security and problem determination. Applications can be defined on top of the meta-services.

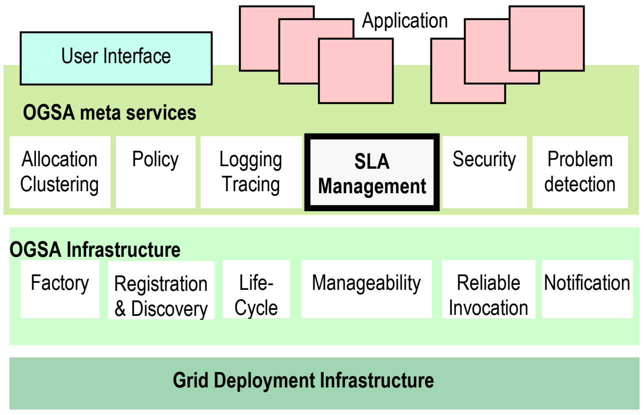

Figure 3: SLA Management in Grid – Conceptual Architecture.

SLA management in the OGSA conceptual architecture will be classified as an OGSA meta-service. SLA Management will in turn, need interfaces to the factory service, registration and discovery service for finding resources based on QoS requirements, life-cycle management service, manageability interface to collect measurement data, reliable invocation for controlling the resources and notification service for informing impacted parties.

For deployment of the architecture, a Grid proxy will be required that provides all the above-mentioned functionalities. The Grid proxy corresponding to a particular Grid deployment infrastructure will have to undertake a set of protocols with other Grid proxies. The Grid community has already agreed on (or discussing) a set of protocols. Some of these protocols are:

- ❑ GRAM protocols such as GRAAP are used for resource allocation and management.
- ❑ GIS is used for registration and discovery of services.
- ❑ GASS, deals with transparent access to data.
- ❑ GSI, deals with authentication and authorization.
- ❑ Grid FTP, is an extension to traditional FTP transfer facility.
- ❑ Other protocols, like SNAP etc.

Some of these protocols are executed at the initialization time while some need to be executed in run time.

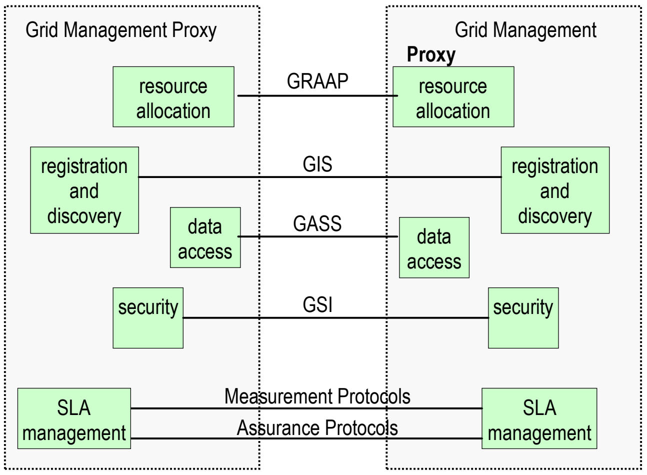

Figure 4: SLA management requires additional protocols between Grid proxies.

The grid proxies will interact with each other forming a Grid management proxy overlay as shown in Figure 5. The SLA Management engine in the proxy may need to execute some additional protocols amongst each other.

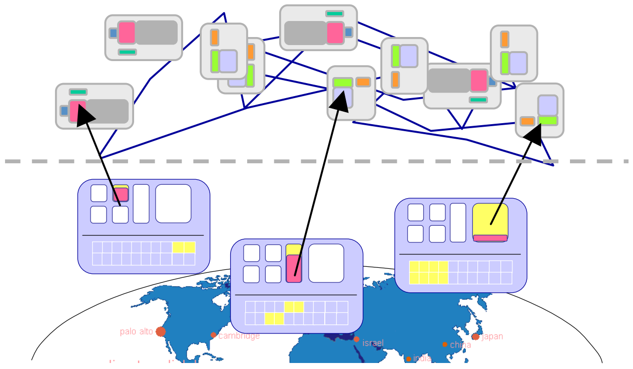

Figure 5: Grid-wide management proxy overlay.

Why does SLA management require additional grid protocols? In the simplest case, all the measurements needed to measure its SLAs and all the controls needed to assure SLAs will be within the same cluster and there is a single cluster in the enterprise. Even in that case, how do the consumer and provider agree on the outcome of the SLA? Since the two are in different management domains, the provider may argue that its SLA has been met while the user may argue that it has been violated. This requires agreement protocols that will repudiate and ensure that the two parties agree on the outcome.

Beyond this simplest case, there exist scenarios in which not all data required for managing SLAs can be measured locally. This can happen in two types of situations - one, when the provider's behavior is dependent on user's behavior, and two, when the provider's behavior is dependent on another provider's (provider-of-provider's) behavior. To understand the first situation, consider a utility infrastructure service provider (U) who offers resources such as servers and network. Consumers (C) can invoke operations on U to request a particular number of servers. The SLA between U and C states that the requested number of servers will be offered with a specified availability. Once C receives the servers, she completely owns what she can place on those servers (including operating systems and applications). Now, C can affect the behavior of U's servers since a failure in C's application can bring down U's server. In order to manage to its SLAs, U's management system should be able to figure out whether a server failure is attributed to a hardware failure (U's responsibility and hence should be counted in the SLA) or to an application failure (C's responsibility and hence should not be counted in the SLA). The only way to resolve this would be by making U's management system and C's management communicate status with each other or to a third party.

A more common situation is when a user has an SLA with a provider and the outcome of that SLA is dependent on a third provider. For example, consider a provider A that promises a certain number of resources to a customer B with some quality guarantees. However, A uses another provider C for some of the resources. As a result, A will not know whether the SLA with B has been met or not. The only way that A can measure its SLA is by consultation with B or C. A monitors its resources, C monitors its resources, they exchange the results to measure the end-to-end SLA between A and B. This would be the case when multiple geographically disparate clusters provide a common interface to a client through an enterprise grid. In such a case, the resources allocated for a client, may be distributed across geographically disparate clusters. In order to evaluate client SLAs, measurement data has to be collected and aggregated at one place necessitating a measurement exchange protocol. This case could also arise if multiple enterprise grids provide a common Grid interface to clients.

There are also scenarios where all the controls needed to manage SLAs are not available locally. Consider a user M that uses a provider N. The management system that manages M may detect that its SLAs with its own customers are going to be violated. It may also detect that the cause of those violations is N not meeting its SLA. However, it does not have any control actions it can execute over N's service. The only controls it has available are switching from N to a different supplier O, making up the difference caused by N's violations by changing its own execution, or negotiating for higher penalties with N. In a slightly different scenario, let us say that M's management system detects SLA violations, but this time, they are not caused by N. It may be able to prevent the violation from happening if it had access to some control actions on N, such as ability to ask for more resources, or escalate the quality guarantees on offered resources. In summary, SLA assurance may also be accomplished by management systems across multiple domains exchanging messages that invoke a limited set of explicitly stated control actions. This is in addition to the local control actions that can be taken by a management system.

In this article we will focus on specification of service level agreements between parties, the SLA monitoring engine and a Measurement Exchange Protocol (MEP) for SLA monitoring purposes.

5 Specifying and Monitoring Service Level Agreements

5.1 SLA definition

An SLA is typically signed between two parties, which have the role of provider and consumer respectively. A typical SLA [7] has the following components:

Purpose – describing the reasons behind the creation of the SLA

Parties – describes the parties involved in the SLA and their respective roles.

Validity Period- defines the period of time that the SLA will cover. This is delimited by start time and end time of the term.

Scope – defines the services covered in the agreement.

Restrictions – defines the necessary steps to be taken in order for the requested service levels to be provided.

Service-level objectives – are the levels of service that both the users and the service providers agree on, and usually include a set of *service level indicators*, like availability, performance and reliability. Each aspect of the service level, such as availability, will have a target level to achieve. Service Level objectives have day-time constraints associated with them, which delineate their validity.

Service-level indicators – the means by which these levels can be measured. Service Level Indicators (SLI) are the base level indicators.

Penalties – spells out what happens in case the service provider under-performs and is unable to meet the objectives in the SLA. If the agreement is with an external service provider, the option of terminating the contract in light of unacceptable service levels should be built in.

Optional services – provides for any services that are not normally required by the user, but might be required as an exception.

Exclusions – specifies what is not covered in the SLA.

Administration – describes the processes created in the SLA to meet and measure its objectives

5.2 Unambiguous and Precise SLA specification

The first enabler for automated SLA management is a **flexible** but **precise** formalization of what an SLA is. The flexibility is needed since we neither completely understand nor can anticipate all possible SLAs for all the different types of web service providers. This will also help create a generic SLA management system for managing a range of different SLAs. The precision is essential so that an SLA management system can unambiguously interpret, monitor, enforce, and optimize SLAs.

Examples of the lack of flexibility and precision in existing SLA formalizations are discussed in [3]. Detailed explanation of how we have addressed flexibility and precision in coming up with SLA formalization are also presented in [3]. Below is a summary of the formalization.

An SLA is specified over a set of data that is measurable. An SLA typically has a date constraint (start date, end date, nextevaldate) and a set of Service Level Objectives (SLOs). An SLO in turn has typically a day–time (Mon-Wed, 6:00PM-8:00 PM) constraint and a set of clauses that make up the SLO. A clause is based on measured data. This is referred to as a *measuredItem*. A measuredItem can contain one or more *items*. A measuredAt element determines where the measurements are taken (provider, consumer side). A clause evaluation is triggered either when an event happens, e.g. say a message arrives, an operation completes or at a fixed time, say at 6PM. We call this an *evalWhen* component of an SLO.

Once the evalWhen trigger arrives, a set of samples of measuredItem are obtained applying a sampling function. The *evalOn* component determines how this sample is computed. The sample set is a constrained set of measured data that is constrained by the evalOn component. Examples of evalOn components may be a number or a time period, e.g. the 5 longest running transactions, or all the samples for last 24 hours. A function (*evalFunc*) is thereafter applied on the sample set so obtained. An example of evalFunc would be average response time function < 5 ms. The evalFunc must be a mathematical function that is expressible in terms of its inputs and logic. The evalAction specifies what action to perform after the evaluation is done. The following grammar shows a portion of this formalization.

```
SLA  = dateconstraint SLO*
Dateconstraint = startdate enddate nextevaldate
SLO = daytimeconstraint clause*
Daytimeconstraint = Day* time
Clause = MeasuredItem evalWhen evalOn evalFunc evalAction
MeasuredItem = Item*
Item  = measuredAt constructType constructRef
```

Figure 6: SLA specification grammar

In order to explain in detail the concepts involved in SLA specification and monitoring, we consider an example scenario. The example scenario comprises of a customer, an application service provider (ASP.com) and a Utility data center (myUDC.com). The management proxies not only ask for resources as explained in section 3 but also decide on a SLA specification (based on resources described in the RSL specification) which in turn is monitored by the Grid proxies.

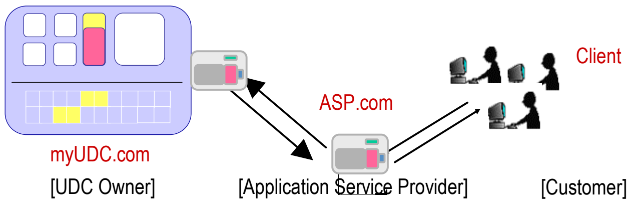

Figure 7: An example scenario.

Assuming that myUDC.com and ASP.com have a clause like *At month-end, the availability of the farm allocated to the user myASP.com, measured on the myUDC.com from Mon-Fri from 9AM-5 PM should be at least 99.9%,* can be specified as follows.

```
<sla>
  <slaId>1</slaId>
  <partnerName>ASP.com </partnerName>
  <startDate>Fri Feb 15 00 :00:00 PST 2002</startDate>
  <endDate>Mon Jul 15 00:00:00 PDT 2002</endDate>
  <slo>
    <sloId>1</sloId>
    <dayTimeConstraint>9:5:1:5</dayTimeConstraint>
    <measuredItem>
      <item>
        <constructType>udc.hp.com/farm</constructType>
        <constructRef> My-2-Tier-Farm</constructRef>
        <measuredAt>myUDC.com </measuredAt>
      </item>
    </measuredItem>
    <evalWhen>month-end</evalWhen>
    <evalOn>all</evalOn>
    <evalFunc>Availability:99.9:percent </evalFunc>
  </slo>
</sla>
```

Another SLA specified between ASP.com and UDC.com, dealing with Mean Time To Repair (MTTR) specified using trouble ticketing messages as described in the Web Service Description Language (WSDL) [13] of the services could be as follows

```
<sla>
  <slaId>2</slaId>
  <partnerName>ASP.com</partnerName>
  <startDate>Fri Feb 15 00:00:00 PST 2002</startDate>
  <endDate>Mon Jul 15 00:00:00 PDT 2002</endDate>
  <slo>
    <sloId>1</sloId>
    <dayTimeConstraint>0:24:1:7</dayTimeConstraint>
    <measuredItem>
      <item>
        <constructType>message</constructType>
        <constructRef>TroubleTicketLaunchMsg</constructRef>
        <measuredAt>myUDC.com </measuredAt>
      </item>
      <item>
        <constructType>Message</constructType>
        <constructRef> TroubleTicketClosedMesg</constructRef>
        <measuredAt>myUDC.com</measuredAt>
      </item>
    </measuredItem>
    <evalWhen>month-end</evalWhen>
    <evalOn>all</evalOn>
    <evalFunc>AvgRespTime:LT:10:min</evalFunc>
  </slo>
</sla>
```

The complete set of examples of how complex SLAs can be represented in the specification language are presented in [3].

5.3 SLA Monitoring

As the specification typically has startdate, enddate, daytimeconstraint, evalWhen, evalOn and evalFunc components to it, each of these constitutes a generic component that can be used by our SLA Management engine. In addition, we have also identified the most common variants of these generic components, which can be readily parameterized by the engine for a large number of possible combinations of SLAs. Using a new, evalWhen, evalOn, or evalFunc component in an SLA requires an administrator to first develop such a component within the framework of our engine and then to add it to the engine. The *model generator* creates a model for the Grid hosting environment in the model repository. All the measurements collected are attached to this model. The instrumentation in the hosting environment is responsible for collecting these measurements and passing them on to the *management* *handler* to be stored in the model repository. If the measurements are collected on the client side (as determined by the measuredAt components of the items in SLA clauses), then the *communicator* is responsible for receiving the measurements and storing them into the repository. SLM Engine process controller receives the SLA executes a monitoring process flow (as explained in subsequent section) and accordingly informs the SLA customizer which in turn customizes the alarms at the Alarm Manager (depending on the evalWhen and dateconstraint components). The Alarm Manager comprises of the SLO Validity Period Monitor, and triggers (time based and event based). The SLA customizer also creates an SLO object in the SLA/contract repository and registers it as the call back handler of the alarms. The SLO object maintains the state of the SLO (valid, active, invalid). If a registered alarm for start-date of an SLO arrives the state of the SLO is changed from init to valid. The SLO is invalidated when the end-date trigger arrives. In between as the evalWhen alarms are triggered (because of a time or an event happening) the SLO evaluator evaluates the SLO. The *SLO evaluator* obtains the required management information (based on start-date, end-date, evalOn, daytime Constraint and the evalFunc constituent of the specification) from the high performance database in memory. The SLO evaluator determines compliance/violations. The *SLA violation engine* maintains the record for violations, their timestamps, the levels of violation, and the clauses that are violated (both in memory and in log files). The management console can be used for looking and visual analysis of the current SLAs, SLOs, and their violation records. The violation records are used for triggering SLA assurance processes.

Measurement and modeling

In order to undertake SLA Management it is essential to measure data and model it. The measurement data is collected through manageability interfaces of the resources. Although a variety of instrumentation techniques and tools exist it is essential that network, system, software layer data be collected through uniform manageability interfaces that are CIM compliant. Currently, instrumentation is done by installing probes at the network, system and software levels of the Grid deployment infrastructure. These probes could range from SNMP agents for network components and machines, a set of perf tools (e.g. perfview) for system level data collection, and special mechanisms for software like the web server (NSAPI/ISAPI filters, log files), application servers (ARM, JMX) and data bases (APIs). Once the probes are installed the data collected has to be modeled in the SLM engine. The CIM [12] based model mimics the topology and dependency of the resources.

Measurement exchange protocol

In cases where an SLA cannot be evaluated solely based on local measurements a measurement exchange protocol is executed for transferring measurements from one site to the site that evaluates the SLAs. Such a protocol is designed with the following objectives in mind: (a) minimize the amount of data that is transmitted between the two sides, and (b) transfer the data in time for the evaluation of SLA to take place when triggered.

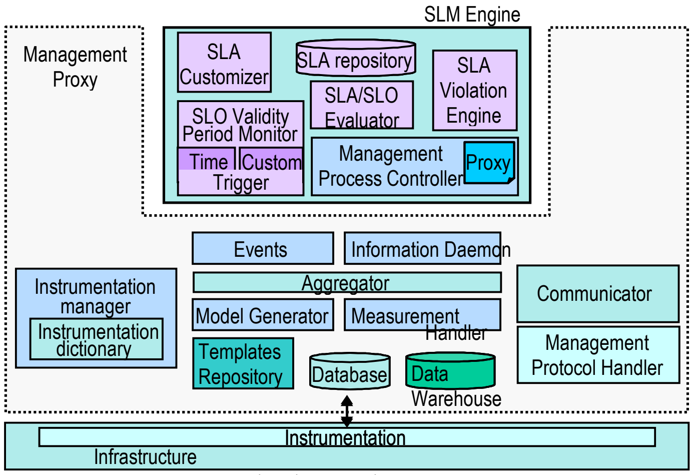

Figure 8: SLA Monitoring engine.

The following needs to be agreed upon (a) what measurements need to be transferred and at what level of aggregation, and (b) how frequently they should be transferred. The type and level of aggregation of the measurements depends on both evalFunc and measuredAt. To specify the level of aggregation, we use typical sampling functions such as count (t), totaled, averaged, movingAvg(lastN), minN, maxN, threshold. In the case when the sampling function cannot be determined from the evalFunc, all measurements are transferred. The reporting frequency depends on evalWhen. The measurement protocol handles both the agreement on level of aggregation and frequency, as well as the transfer of agreed measurements. There are in essence 5 different types of messages that form the protocol

- *Init:* sent by the site that measures to the site that evaluates the SLA. The init message carries possible choices of sampling function, interval, duration and reporting interval details that the site supports.
- *Request:* The evaluator site decides the exact measurement specification (sampling function, sampling params and reporting params) that it chooses and specifies it in its request message
- *Agreement:* The measurement site sends this message if it agrees to the request

- *Start:* message from the evaluator site to commence the reporting
- *Report:* actual measurement report messages
- *Close:* Message to terminate reporting.

A management proxy prototype was implemented in Java. The management proxy uses an Oracle9i warehouse for data archiving and retrieval purposes. A metric model derived from CIM was used to model the collected data. The management proxy was implemented as web service using Tomcat 3.2.3. It uses Apache SOAP router to communicate with other proxies. The measurement exchange protocol was also implemented.

6 Summary and Conclusion

Applying the Grid model to commercial environment requires specification, monitoring and assurance of guarantees expressed in terms of service level agreements.

In this article we described a language for unambiguous and precise specification of service-level agreements. We also explained a monitoring engine for evaluating these SLAs and a measurement exchange protocol for obtaining and aggregating measurements from multiple sites.

References

[1] Foster, I., Kesselman, C., Nick, J.M., Tuecke, S., *The Physiology of the Grid – An Open Grid Services Architecture for Distributed Systems Integration*, http://www.globus.org/research/papers/ogsa.pdf, May 2002.

[2] HP, *Utility Data Center*, http://www.hp.com/go/hpudc, http://www.hp.com/go/always-on, November 2001.

[3] Sahai A, Durante A, Machiraju V. Towards Automated SLA Management for Web Services. HPL-2001-130.

[4] Sahai A, et al. Automated SLA Monitoring for Web Services. IEEE/IFIP DSOM 2002. Montreal Canada 2002.

[5] The Global Grid Forum, http://www.gridforum.org/..

[6] Sahai A, Graupner S, Wooyoung K. The Unforlding of Web Services Paradigm. Internet Encyclopedia by John Wiley. Also HPL-2002-130.

[7] Sturm R, Morris W, Jander M. Foundation of Service Level Management. SAMS publication.

[8] Krauter, K., Buyya, R., Maheswaran, M.:*A Taxonomy and Survey of Grid resource Management Systems,* Software-Practice and Experience, 32(2):135-164, 2002.

[9] Graupner, S., Kotov, V., Trinks, H.: *Resource-Sharing and Service Deployment in Virtual Data Centers*, IEEE Workshop on Resource Sharing in Massively Distributed Systems (ICDCS-2002), July 2, 2002, Vienna, Austria.

[10] Graupner, S., Reimann, C.: *Globus Grid and Firewalls: Issues and Solutions in a Utility Data Center Environment*, HP Labs Technical Report, HPL-2002-278, September 2002.

[11] Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: *SNAP: A Protocol for Negotiation of Service Level Agreements and Coordinated Resource Management in Distributed Systems*, submission to Job Scheduling Strategies for Parallel Processing Conference (JSSPP), April 30, 2002.

[12] Common Information Model CIM from DMTF. http://www.dmtf.org/standards/standard_cim.php

[13] Web Service Description Language (WSDL) http://www.w3.org/TR/wsdl