

Connecting Client Objectives with Resource Capabilities: An Essential Component for Grid Service Management Infrastructures

Asit Dan
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
asit@us.ibm.com

Catalin Dumitrescu
Computer Science Department,
University of Chicago
1100 E 58th St., Chicago, IL 60637
cldumitr@cs.uchicago.edu

Matei Ripeanu
Computer Science Department,
University of Chicago
1100 E 58th St., Chicago, IL 60637
matei@cs.uchicago.edu

ABSTRACT

In large-scale, distributed systems such as Grids, an agreement between a client and a service provider specifies service level objectives both as expressions of client requirements and as provider assurances. Ideally, these objectives are expressed in a high-level service or application specific manner rather than requiring clients to detail the necessary resources. Resource providers on the other hand, expect low-level, resource specific performance criteria that are uniform across applications and can easily be interpreted and provisioned.

This paper presents a framework for grid service management that addresses this gap between high-level specification of client performance objectives and existing resource management infrastructure. It identifies three levels of abstraction for resource requirements that a service provider needs to manage, namely: detailed specification of raw resources, virtualization of heterogeneous resources as abstract resources, and performance objectives at an application level. The paper also identifies three key functions for managing service level agreements, namely: *translation* of resource requirements across abstraction layers, *arbitration* in allocating resources to client requests, and *aggregation and allocation* of resources from multiple lower level resource managers. One or more of these key functions may be present at each layer of a service level manager. Thus, the composition of these functions across resource abstraction layers enables modeling of a wide array of management scenarios. We present a framework that supports these functions: it uses the service metadata and/or service models to map client requirements to resource capabilities, it uses business values associated with objectives in allocation decisions to arbitrate between competing requests, and it allocates resources based on previously negotiated agreements.

Keywords

Management of service level agreements, resource managements, service oriented computing.

1. INTRODUCTION

In large-scale, service-oriented systems like today's Grids [1], a client may establish an agreement [2] with a service provider prior to the actual invocation of the service. Such an agreement provides assurance to the client on the availability of the requested resources and/or on the service performance. The agreement may include one or more guarantee terms on service level objectives (SLO) – e.g., average service response time lower than one second. A guarantee term may also include the importance of meeting the objective (e.g., expressed as a penalty for not meeting the objective) and additional qualifying conditions (e.g., duration) [3, 4]. The presence of SLOs enables flexible allocation of resources at the service provider: First, resources can be allocated dynamically to meet the SLO only when a client invokes a service, leading to higher resource utilization through resource sharing across client requests. Second, when a resource conflict arises due to a peak in offered load the provider can make more informed resource allocation decisions.

However, finding the appropriate level of abstraction for SLOs is non-trivial. For ease of use, clients would like to express their objectives in terms that are meaningful to them, such as response time, or even an abstract expression of processing requirements, e.g., required MIPS. With an abstraction at this level a client does not have to specify resource details, such as requirements on cache or processor characteristics. To facilitate this abstraction, however, a provider needs to present its resource capabilities in abstract terms. Furthermore, the provider needs to be able to interpret offered SLOs and to determine the detailed resource configuration necessary to meet them. SLOs on response time of a service invocation or completion time of a job execution, however, are difficult to meet if the application is not well characterized or if there is a high variance on resource requirement, (e.g., execution time is dependent on the size of the input data).

Managing client objectives might also imply aggregating resources from multiple resource providers. To avoid the overhead and the latency of resource acquisition, an intermediary may acquire resources for longer intervals thus

enabling reuse across client requests. Figure 1 illustrates this scenario: the intermediary acts as a resource aggregator on clients' behalf and establishes SLAs with each resource provider.

The challenge is to provide the right level of abstraction: the SLO parameters (e.g., granularity, duration) that can be met by the provider while requiring low resource management overheads at the provider, i.e, help avoiding resource reconfiguration, and promote high resource utilization.

In this paper, we introduce a framework for bridging the gap between the high-level specification of client objectives, and detailed specification of resource capabilities managed by a resource manager. This framework, we refer to as the *Service Level Manager (SLM)*, performs three essential functions. Firstly, it uses detailed service performance modeling embedded in service factories to maps client objectives to resource requirements. Secondly, the SLM acquires and aggregates resources, (possibly from multiple resource managers), and maintains the bindings to these resources. Finally, the SLM keeps track of client established SLAs, and arbitrates dynamic allocation of resources across multiple client objectives.

The paper is organized as follows. Section 2 uses a concrete example to illustrate how client objectives can be expressed at various levels of abstraction. We show that detailed specification of resource requirements is both cumbersome for clients and results in resource underutilization. Raising the abstraction level on the other hand introduces challenges in managing client objectives. In Section 3, we examine the architectural requirements for supporting the three key SLM functions, and the composition of these functions in a layered architecture going from high-level client objectives to detailed resource specification. We summarize in Section 4.

2. CLIENT AND RESOURCE PROVIDER OBJECTIVES

We use a real world scenario to better motivate the need to bridge the gap between client objectives and resource provider capabilities. We refer to this scenario throughout the rest of the paper. This section then describes possible abstraction levels at which clients and resource provider could specify their performance objectives. Our description is not intended to be exhaustive, or prescriptive; we are rather interested in exploring the space of performance objectives and resource requirements that need to be expressed using SLAs.

2.1. Motivating Scenario

Consider the following example scenario in a financial company [5]:

An investment-bank, active on multiple financial markets, owns computational resources in each country where it operates. These resources are primarily used to support traders (i.e., as trading screens). Since traders are active only during the day, the bank decides to run a portfolio risk evaluation, a compute and data intensive application, on these resources at nighttime. One strong

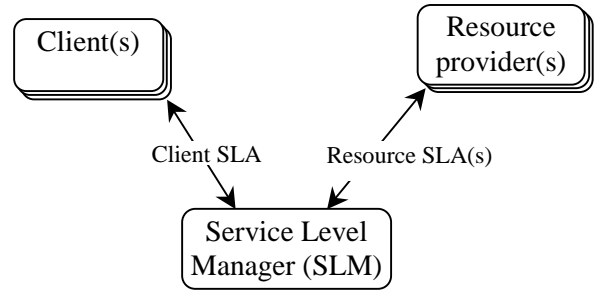


Figure 1: The Service Level Manager (SLM) bridges the gap between client and resource provider abstraction levels.

requirement is that results are available before 7:00am, the start of the trading day, in each market.

The application consists of two stages. First it solves a large system of linear equations. This is a parallel, tightly-coupled, compute and network intensive computation. The result, together with new input, is used in the second stage organized as a master-worker: workers compute the risk profile for a set of trades, send results back to the master, and receive a new set of trades to price. The master assembles all information received into the final result.

In each market, at the end of the trading day, a new computation is started with a 7:00AM next day deadline expressed as an SLA. This application level goal is translated into lower level SLAs for each stage of the application, and either matched with existing services or with new services created on demand. If available resources are scarce these services are instantiated at a utility provider with which the bank has negotiated CPU outsourcing agreements. If SLA violations are reported, a repair mechanism is invoked: it reevaluates the feasibility of producing the final result before the deadline and might acquire additional resources (through the same SLA mechanism). A related scenario is suggested in [6]: there, HTTP requests with a given average response time goal are given priority over a long-running application and peak load is offloaded to resources used for this application.

A number of requirements emerge from this scenario. Client applications need to express performance objectives at various levels: in our example the portfolio evaluation deadline is a performance objective that is in turn translated into deadline and/or throughput requirements for each application stage. Resource providers, on the other hand, express their offers and negotiate contracts in resource specific terms. A separate component is needed to bring clients and resource providers together, to monitor progress, and to take action if progress does not meet expectations. Additionally, the runtime framework needs to match client and resource provider SLAs, to dynamically allocate resources either internally or externally (e.g., from utility providers), and to dispatch requests according to the SLAs that have been created.

2.2. Resource Provider Objectives

Resource providers handle low-level, resource-specific performance objectives that (ideally) are application independent and can be easily interpreted and provisioned for. We envisage three levels, at which resources or groups of resources can be abstracted: firstly, at the most primitive *raw resource level* resources are identified individually, secondly at the *abstract resource level* qualitative and quantitative performance objectives are specified characterizing in fact individual or groups of physical resources, finally, at the more complex *service level* performance objectives are tied to the execution of a specific service. We detail each of these abstraction levels below.

2.2.1. Raw Resource Level Objectives

At the lowest level, service level objectives (SLOs) include detailed resource configuration information: they describe individually the physical resources requested (e.g. “I need 16 processors on cluster X for 4 hours starting as soon as possible”). Note that this implies that the client has prior knowledge about the characteristics of the resource requested.

A slightly more involved step is to describe objectives in terms of predefined, static or dynamic resource attributes. (“16 processors with more than 512MB of memory and faster than 2GHz, interconnected with a 10Gbps switch”).

A number of computing frameworks routinely present resources at this level. Examples are: the Globus Toolkit’s Grid Resource Management Service (GRAM) [7] and its associated Resource Specification Language (RSL) or the Job Submission Definition Language (JSDL) [8].

The main drawbacks for solutions at this level are twofold. First, they imply a burdensome level of detail on the clients that need to fully specify the SLAs and acquire resources from providers. Second, fixed allocation reduces resource management flexibility: it does not allow for resource sharing across client requests and may result in inefficient resource utilization. While this level of detail might be required in certain cases, we are interested in expressing abstract, higher-level objectives that can be translated into lower-level resource requests by the run-time infrastructure.

2.2.2. Abstract Resource Level Objectives

To quickly convey intuition we start with an example: the resource objective in the example above may be restated as “I need aggregate compute power of 3.2 GFLOPS and aggregate memory of 8GB. The communication pattern of my application is a 2D mesh and the communication/computation ratio is 1.2Mb/MFLOP”.

At this level we use higher-level SLOs that a resource provider accepts and translates to raw resource requests. Obviously this provides more flexibility: the resource provider can allocate resources appropriately, and, in effect, it takes over the task of finding appropriate resources from a client that does not have – or does not care about – detailed resource information.

We list below some of the performance related objectives that can be used:

- CPU related metrics: e.g., aggregates of MIPS, MFLOPS, etc.
- Memory related metrics: aggregated memory size and CPU to memory bandwidth.
- Storage related metrics: total space to be used, read/write aggregate throughput to storage.
- Network related metrics: aggregate generated traffic in a time interval; metrics that link the generated traffic volume to CPU related metrics (e.g. traffic generated per MFLOP).

Most of these metrics can be used to express either ‘strong’ deterministic guarantees (e.g. the system will deliver more than N units of the metric) or as statistical guarantees (e.g., averages over some time period). Specifying the time period over which averages are computed and the variability (standard deviation, variance) are possible refinements

In addition to the performance related objectives, *qualitative objectives* turn out to be necessary:

- Data transfer patterns: for some applications, application topology information is crucial to efficiently select and configure resources. One way to cope with the large space of possible application topologies is to predefine the most commonly used topologies – star, mesh, hyper-cube, etc. – and use graphs to model exotic ones.
- Heterogeneity levels. While a large number of parallel applications can efficiently use heterogeneous resources, other classes of applications require homogeneity both in terms of resource types (e.g. same operating system) and resource capabilities (e.g. same CPU rates).
- Hardware configuration. For example some applications might require machines that implement a specific floating point operations standard.
- Reliability guarantees. Long running client applications might require reliability guarantees from resource providers.

2.2.3. Resource Provider - Application Level Objectives

The resource provider might move up in the value chain and provide services (in addition to providing raw resources where clients instantiate their own services). One example are the parallel libraries (e.g.: BLAS [9], SCALAPACK [10]) which are de-facto standards and can form the basis of a set of linear algebra services offered by a provider.

Below, we suggest performance objectives that could be used at this level to set-up SLAs between a client and a resource provider. As above, we only enumerate single objectives; nothing prevents clients to negotiate SLAs that include multiple objectives.

- For simple services, where application behavior is well understood, provider goals might get close to client application objectives such as completion time or throughput (which we describe in detail in the following section).
- For more complex services, resource providers may still express goals in terms of resource consumption (CPU,

memory or storage related metrics). The significant advantage at this level is that, for a large class of applications, the service topology and generated network traffic are abstracted away. In fact, the performance of the service can be expressed using only one of the metrics described earlier.

- New, service specific, qualitative metrics might come into play at this level. For example, the service that solves a linear system of equations will advertise the specific solving algorithm used or the guaranteed solution accuracy.

Note that, from a resource provider perspective, accepting commitments at this level is risky unless the application/service behavior is well understood and faithfully modeled. There are multiple challenges; firstly, applications with nonlinear performance or resource consumptions are not uncommon. Secondly, for some classes of services it is impossible to estimate beforehand the work required to serve the request.

2.3. Client Level Objectives

Clients (or client applications) need a high-level, application specific, and resource independent way to define their performance objectives. These objectives might imply aggregates over multiple resource, resource types, and/or resource providers, and a tight control of provider's resources. Below we attempt to classify client application objectives:

Time related client objectives cover a large part of the usage space: they specify time bounds for serving a certain request (or request type). In addition to best-effort service application-level objectives can include:

- Per request service time objectives. These can either specify a guaranteed service time upper bound or averages over longer periods.
- Throughput objectives. A specified number of requests are served during a time interval of specified duration. Note that meeting a throughput objective does not imply bounds on per-request service time.

Objectives might be conditioned on request argument values (e.g. "If arg1 of the request is in a specified interval then the service should be able to serve X requests per second").

Clients can add application specific, *quality of solution objectives* (e.g., the solution to an optimization problem approximates the best solution with certain accuracy) or *reliability objectives* (e.g., guaranteed service uptime, or data availability.) Client objectives may also include assurances on resource availability at a future time, i.e., advance reservation. Providers may express confidence level on the availability of resources at a future time.

3. KEY SERVICE LEVEL MANAGER FUNCTIONS: CONNECTING CLIENT AND RESOURCE PROVIDER OBJECTIVES

3.1. Resource Management and the Service Level Manager

We argue that the Service Level Manager (SLM) needs to fulfill three main functions:

- *Resource and environment identification.* Here high-level client performance objectives and service abstraction are translated into detailed resource requirements and identification of environment. To achieve this goal the SLM needs two types of information: accurate modeling of service/application performance and timely information on available resources. The SLM needs to translate high-level performance objectives into raw resource requirements that can be allocated by resource managers at the resource provider.
- *Arbitration and matching.* The SLM matches the available resources with detailed resource requirements, and arbitrates across simultaneous requests that compete for same set of resources for a resource allocation that maximizes the 'utility' delivered by client applications.
- *Aggregation and allocation of resources.* Actual physical resources are allocated and/or environment is configured prior to application execution. When multiple resources are involved in serving a single request or in instantiating one service, the SLM is also the coordinator in acquiring and managing these resources. Resources involved might belong to a single or to multiple administrative domains and/or might be of different types (e.g., the transfer of large data-sets between two organizations where computational capacities and storage need to be reserved at each end as well as a network light-path).

To avoid developing new custom code for each new service, these SLM functions are developed in a general manner (referred to as "middleware"), and applicable across a wide range of services.

3.1.1. Resource and Environment Identification

There are multiple possible; approaches to map a high level objective to a detailed resource configuration. The choice of a specific approach depends on the application objectives being modeled. Prior to creation of a SLA, a provider specifies SLA templates that describe the parameters that can be customized by clients [2]. Hence, the most straightforward approach is to use a table for mapping a combination of customizable SLA parameters to a set of resource configuration parameters. Such tables can be built using analysis, heuristics or application specific benchmarking for specific resource configurations. Clearly, this approach can work only if a small number of parameters are to be changed across SLAs, and/or a small set of services are to be modeled by the reusable SLM code. To handle a larger variability in SLAs, and hence, derived resource configurations, mapping requires use of customized application models for each application and SLA templates. Building such a model,

however, is not straightforward, and requires domain knowledge. Extrapolation from a known set of configurations is advocated in [11] while other projects focus on automatically extracting application performance models [12]. In Section 3.4 we describe our experience with a service-based prototype addressing these functions for a high performance scientific computing environment.

3.1.2. Matching and Arbitration

Once the resource requirements for a specific SLA are identified, the next step is matching [13, 14]: discovering sets of candidate resource that meet these requirements. Information on available resources characteristics, on their status, and on other existing SLAs, needs to be taken into account. The result of matching is a set of pre-allocated resources so as to meet SLA objectives. When multiple concurrent requests are present, if insufficient resources to meet the SLAs are available, then allocations need to be prioritized based on the business value or utility associated with meeting a the service objective [2]. This process is referred to as arbitration.

Traditionally, resource management systems have focused on optimizing system-centric performance metrics such as: mean computation time, or average system utilization [7, 15, 16]. These resource management systems use combinations of priorities and fair sharing to express resource allocation policies [17]. These mechanisms however do not express the value clients attach to fulfilled service requests or the value resource providers associate with their assets. The SLA based management therefore introduce two value related metrics: (1) the utility value associated by the client to a served request, and (2) a penalty function for not meeting the specified performance objectives. The penalty function can be as simple as saying that no value is delivered if performance objectives are not met, to more sophisticated functions: (e.g. a decreasing value function over time for time related objectives [4, 18, 19]). In [20], an workload management system is described that uses adaptive algorithms for managing multiple client objectives. The algorithm learns dynamically resource requirements for meeting an objective, and adjusts various types of systems resources allocated to an application (e.g., MIPS, buffer size, etc.). In [21] a SLA based web services management uses request prioritization to manage average response time goal.

3.1.3. Allocation and Binding

Once a resource pre-allocation decision has been taken, making this decision effective may be as simple as marking a specific node as allocated, and starting a job or application in the allocated node. However, this may also mean provisioning or reconfiguring a server for use by the selected application. In a shared resource environment, this physical allocation process is continual: dynamic adjustments of resource allocation are necessary, either changing policies, execution priorities, or reconfiguring the execution environment by starting, stopping or migrating the application to new server node. If the resources used by an application are managed by multiple independent managers (e.g., multiple independent clusters, decomposed into multiple application tiers, decomposed into storage, database and application server, etc.), then the allocation or

reconfiguration decision is conveyed to all the resource managers. Here, the application level SLM may have lower level SLAs with the resource managers.

3.2. Composition of SLM functions

One can view the SLM as a vertically integrated component providing the three functionalities we have just presented. However, we believe that a layered view that matches the virtualization levels described in Section 2 is a better start for architecting the SLM. Thus, we loosely separate SLM in three functional layers (Figure 2):

- Layer 1: *Service-level resource management*. At this layer application/service specific performance and structural models are used to translate high-level performance objectives to abstract resource requirements and to infer the categories of resources needed.
- Layer 2: *Abstract resource management*. This layer uses information published by resource providers (and obtained through monitoring mechanisms) to translate abstract performance objectives to requests for concrete resources.
- Layer 3: *Raw resource management*. We include this layer here only for completeness as, in fact, this is the layer where most existing resource managers (e.g., cluster schedulers [11, 16, 17] or network reservation mechanisms [22]) already operate to perform raw resource acquisition.

3.2.1. SLM Layer 1: Service-Level Resource Management

This SLM layer translates application-level performance objectives resource requirements at the abstract resource level. Additionally, this layer interfaces with service factories

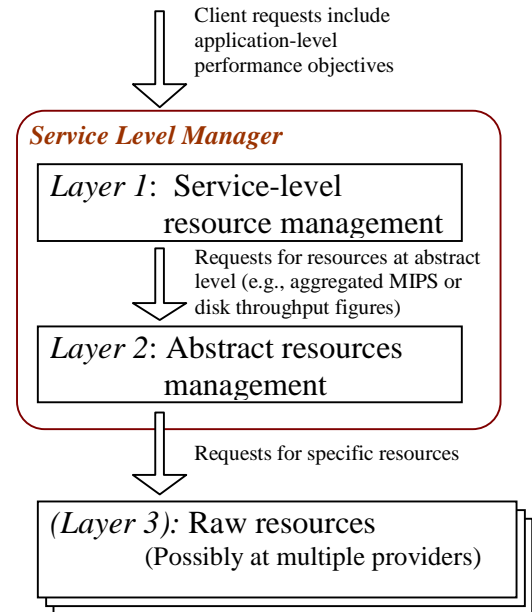


Figure 2: Service Level Manager (SLM) layer decomposition

that encapsulate all service/application specific details: at a minimum, factories incorporate a performance model, information needed to map the service components to resources, and information needed to start and configure the service.

A service instantiation request includes service performance objectives and notification conditions. Before accepting the request, this layer collaborates with the service factory to instantiate the service. The handle of the newly created service instance is returned to the client to submit actual request(s) or to subscribe to notification events. All subsequent invocations of the service are transparent with regard to actual low-level resource management or resource aggregation (co-scheduling and co-reservation) [23, 24].

3.2.2. SLM Layer 2: Abstract Resource Management

This layer provides the ability to translate abstract performance objectives to raw resources requests, and to start execution and monitoring services on specified resources. However, in order to support these functionalities, the SLM needs to:

- Query resource providers for static resource characteristics and monitor dynamic characteristics of these resources.
- Acquire and manage resources (potentially across multiple providers. This SLM layer maintains aggregate views of resources and keeps track of current allocations in the system.
- Submit and control job executions at these providers.

Heterogeneity is one significant practical obstacle at this layer: each resource manager (RM) at each provider may have its own protocols and APIs for the tasks enumerated above. If these RM that are not Grid-enabled, this layer will gather information in a different way from each RM, translate this information to an uniform representation, aggregate it, and keep track of current allocations. Grid-enabled RMs provide uniform higher-level functionality (like aggregate views over managed resources and advanced monitoring mechanisms) that simplify this SLM layer.

We note that, while the translation, arbitration, and aggregation functions presented above are implemented by the SLM as a whole, each of the SLM layers implement some, if not all, of these functions themselves. Thus, the behavior of the whole SLM is obtained by composing the functionalities of its layers. For example, performance objective translation is realized at both layer 1 and 2. Layer 1 uses an application specific performance model to translate application level performance objectives into abstract resource objectives, while layer 2 uses resource status and configuration information to further translate these objectives into request for specific resources. Similarly, an aggregation function may be present at both these layers although with different goals: the higher level might aggregate across resource providers (belonging to different administrative domains) while the lower level might simply aggregate resources from within the same administrative domain.

Similarly, an arbitration function can also be present at all three layers. Multiple client requests for the same service may be arbitrated at the layer 1, with layer 2 merely translating request objectives. Arbitration can also be pushed down to layer 2, if no aggregation is required at layer 1. A similar scenario is possible between layer 2 and layer 3.

We envisage an architecture where these layers are loosely coupled and use SLAs to communicate performance objectives, utilities associated with requests, and monitoring and fault information from one layer to the next. We believe that, as these ideas gather acceptance and as standardization work matures, these mechanisms will be incorporated in the default resource management schemes deployed by the resource providers themselves in the same way schedulers with sophisticated functionality are adopted in today cluster resource managers.

In the rest of this section we first reexamine our application scenario from Section 2.1 and show how it maps on this three-layer SLM view. We then describe in the functionality of and the information maintained by each SLM layer; and conclude by presenting our prototyping effort.

3.3. Revisiting the Motivating Scenario

We assume that the SLM hosts two service factories that match the two stages of the application: a factory that creates linear equation solver services and a factory for services that evaluate trade risk profiles.

The first application stage solves a large system of linear equations. The client contacts the SLM to instantiate a solver service with specified response time and precision (Step1 in Figure 3, next page). SLM layer 1 contacts the service factory with these application level performance objectives. The factory, which incorporates a performance model for the solver service, translates these objectives into lower level (abstract-level) resource requirements and passes them back to the SLM (steps 2 and 3). The SLM uses this information to negotiate resource reservations through its Resources Manager (RM) component (4 and 5) and passes the reservation handlers back to the service factory (6). The service factory can now instantiate new service on these resources (7) and returns its handler to the SLM. At this moment, the SLM passes back the service handle to the client (8) and steps out from the communication. The client will use this handle to communicate with the service instance to submit requests, to check service status, or to receive notifications (9).

When the first stage completes, the client contacts the SLM to instantiate a new service that estimates risk profiles. The client estimates that it will have about 100,000 trades to estimate before the deadline so it asks for a throughput of 30 requests/sec. Through the same mechanism the SLM collaborates with the service factory to obtain resource specifications, acquires resources, starts the service instance, and returns back a handle. The client starts using the service and, if it realizes that in fact it has more risk profiles to estimate it asks for additional throughput to be added to the service.

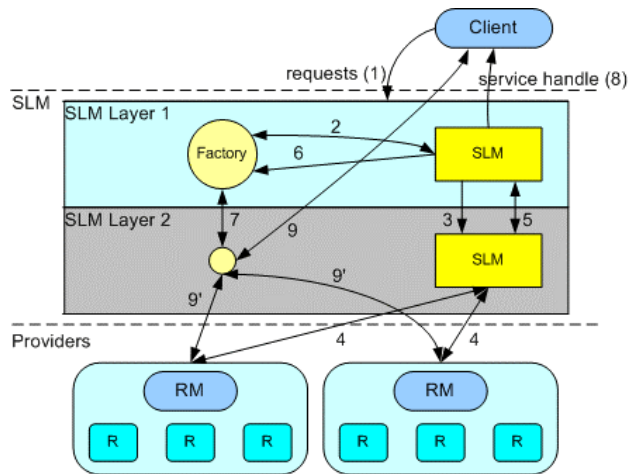


Figure 3: Service Level Manager – Sample usage

Virtualization and aggregation are achieved at multiple levels. Firstly, the SLM, acting on behalf of the client application, transparently instantiates services with specific application-level performance objectives and translate these objectives into lower level resource requirements. Secondly, the SLM is able to aggregate resources from multiple resource providers.

3.4. Prototyping

While we do not have a complete implementation of the framework presented above, we developed two prototypes to investigate (1) the feasibility of integrating service factories that incorporate service performance models into the SLM and, (2) the benefits of managing low-level resources using WS-Agreements. We are describing the first prototype in the rest of this section.

Cactus [25, 26] is a framework for developing scientific and engineering high-performance computations. We have chosen Cactus for our experiments as it has a relatively large use base and as we have extensive experience [27] with and have detailed application performance model [28] for this application.

Cactus is a traditional MPI application; therefore our first enhancement was to modify and expose it as a Grid service [29]. Additionally, we have implemented the corresponding service factory with the two main functionalities we described for in Section 3.1. Firstly, the service factory hides all application specific details of starting configuring the computational framework. Secondly, the factory embeds the application's performance model and is able to translate high-level performance objectives (in this case response time) into lower level resource requirements (in this case number of nodes on a cluster). With this prototype we have deviated slightly from the model proposed and, once this translation to resource provider level objectives is made, the service factory acquires itself resources from the resource provider. The factory then instantiates a new Cactus service and returns its handle for use. Besides the advantages of this architecture advocated earlier in this paper (e.g., clean encapsulation of application specific behavior in the service factory) we obtain additional performance gains turning a

traditional MPI application into a service: startup costs can be significant for a MPI application requiring a large number of CPUs, and our service-oriented architecture amortizes startup costs over multiple requests.

4. SUMMARY

This paper addresses the gap between the high-level specification of client resource requirements to meet specific performance objectives, and detailed specification of resources managed by resource providers. In the absence of a framework to bridge this gap, clients are forced to specify detailed required resource configurations in support of their performance objectives. We identify three levels of abstraction for specifying resource requirements a service provider needs to manage; namely: detailed specification of raw resources, virtualization of heterogeneous resources as abstract resources, and service level requirements at an application level. We also identify three key functions to manage service level agreements, namely: *translation* of resource requirements across virtualization layers, *aggregation* of resources from multiple lower level resource managers, and *arbitration* in allocating resources across client requests in managing service level agreements. We use a real world example from the financial industry, to illustrate the three abstraction layers and the use of these key SLM functions in managing these layers. Finally, we describe a prototype that addresses an important SLM implementation aspect: using service factories to encapsulate service specific information (service performance models, and service instantiation details) and to make it available to the SML in a uniform way.

Acknowledgements: We would like to thank Marcos Novaes for providing details on the example scenario, and Ian Foster for his feedback on the paper.

5. REFERENCES

- [1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*: Morgan-Kaufmann, 2004.
- [2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification (WS-Agreement), Version 1.1, Draft 20," in *Global Grid Forum*, 2004.
- [3] H. Ludwig, A. Keller, A. Dan, and R. King, "A Service Level Agreement Language for Dynamic Electronic Services,," presented at 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS'02), Newport Beach, California, USA, 2002.
- [4] "WSLA Language Specification, Version 1.0," IBM Corporation, 2003.
- [5] M. Novaes, "Personal Communication," 2002.
- [6] A. Leff, J. T. Rayfield, and D. M. Dias, "Service-Level Agreements and Commercial Grids," *IEEE Internet Computing*, vol. 7, pp. 44-50, 2003.
- [7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *4th Workshop on Job Scheduling Strategies for Parallel Processing*: Springer-Verlag, 1998, pp. 62-82.

- [8] A. Anjomshoaa, F. Brisard, R. L. Cook, D. K. Fellows, A. Ly, S. McGough, and D. Pulsipher, "Job Submission Description Language (JSDL) Specification v0.3," Global Grid Forum 2004.
- [9] J. J. Dongarra, J. DuCroz, I. Duff, and S. Hammerling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, vol. 16, pp. 1-17, 1990.
- [10] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, Graham Fagg, K. Roche, and S. Vadhiyar, "Numerical Libraries And The Grid: The GrADS Experiments With ScaLAPACK," *International Journal of High Performance Computing Applications*, vol. 15, 2001.
- [11] "IBM Load Leveler: User's Guide," September 1993.
- [12] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development," *International Journal of High Performance Computing Applications*, vol. 15, pp. 327-344, 2001.
- [13] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," presented at IEEE International Symposium on High Performance Distributed Computing, 1998.
- [14] C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications," presented at 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, 2002.
- [15] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," in *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, pp. 104-111.
- [16] R. Henderson and D. Tweten, "Portable Batch System: External Reference Specification," 1996.
- [17] "Maui Scheduler, <http://www.supercluster.org/maui/>," Center for HPC Cluster Resource Management and Scheduling, 2004.
- [18] D. Irwin, L. Grit, and J. Chase, "Balancing Risk and Reward in Market-based Task Scheduling," presented at HPDC-13, Honolulu, Hawaii, 2004.
- [19] M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Contract-Based Load Management in Federated Distributed Systems," presented at NSDI'04, San Francisco, CA, 2004.
- [20] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive algorithms for managing a distributed data processing workload," *IBM Systems Journal*, vol. 36, 1997.
- [21] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web services on demand: WSLA-driven automated management," *IBM Systems Journal*, vol. 43, pp. 136, 2004.
- [22] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," presented at Proc. International Workshop on Quality of Service, 1999.
- [23] C. H. Crawford and A. Dan, "eModel: Addressing the Need for a Flexible Modeling Framework in Autonomic Computing," presented at MASCOTS, 2002.
- [24] S. B. Calo and D. Verma, "Service Level Driven Provisioning of Outsourced IT Systems," IBM T.J. Watson, Hawthorne, NY RC22501, 06/25/2002. 2002.
- [25] G. Allen, T. Goodale, G. Lanfermann, E. Seidel, W. Benger, H.-C. Hege, A. Merzky, J. Mass'o, T. Radke, and J. Shalf, "Solving Einstein's Equation on Supercomputers," *IEEE Computer*, pp. 52-59, 1999.
- [26] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, and P. Walker, "Numerical Relativity in a Distributed Environment," presented at Proc. 9th SIAM Conference on Parallel Processing for Scientific Computing, 1999.
- [27] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus," presented at SC'2001, Denver Colorado, 2001.
- [28] M. Ripeanu, A. Iamnitchi, and I. Foster, "Cactus Application: Performance Predictions in Grid Environments," presented at European Conference on Parallel Computing (EuroPar), 2001.
- [29] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Standholm, D. Snelling, and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI) version 1.0," presented at Global Grid Forum, 2003.