A Distributed Resource and Network Partitioning Architecture for Service Grids

Bruno Volckaert, Pieter Thysebaert*, Marc De Leenheer,
Filip De Turck+, Bart Dhoedt, Piet Demeester
Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
E-mail: Bruno.Volckaert@intec.UGent.be
*Research Assistant of the Fund of Scientific Research - Flanders (F.W.O.-V., Belgium)
+Postdoctoral Fellow of the Fund of Scientific Research - Flanders (F.W.O.-V., Belgium)

Abstract *In this paper, we propose the use of a distributed service management architecture for state-of-the-art service-enabled Grids. The architecture is capable of performing automated resource and network bandwidth partitioning based on registered Grid resource properties and monitored Grid service demand. A main characteristic is that it enables the use of different service priority schemes and allows for policy-based differentiation between local and foreign service offerings. Resource and network bandwidth partitioning algorithms are introduced and their performance is evaluated on a sample Grid topology using NSGrid, an ns-2 based Grid simulator. Our results show that the use of this Service Management Architecture improves resource efficiency, simplifies schedule making decisions, reduces the overall complexity of managing the Grid system, and at the same time improves Grid service QoS support (with regard to job response times) by automatically making Grid resource and network service reservations prior to scheduling.*

1 Introduction

As more and more application types are ported to Grid environments, we notice an evolution from purely computational and/or data Grid offerings to full-scale service Grids [1]. In this paper, a 'service Grid' denotes a Grid infrastructure capable of supporting a multitude of *application types* with varying QoS levels (i.e. our definition of Service Grid is not limited to web-service enabled Grids). The architectural standards for these Service Grids are provided by the Global Grid Forum's Open Grid Service Architecture (OGSA) [2], and (to a lesser extent) the Web Service Resource Framework, building on concepts of both Grid and Web Service communities.

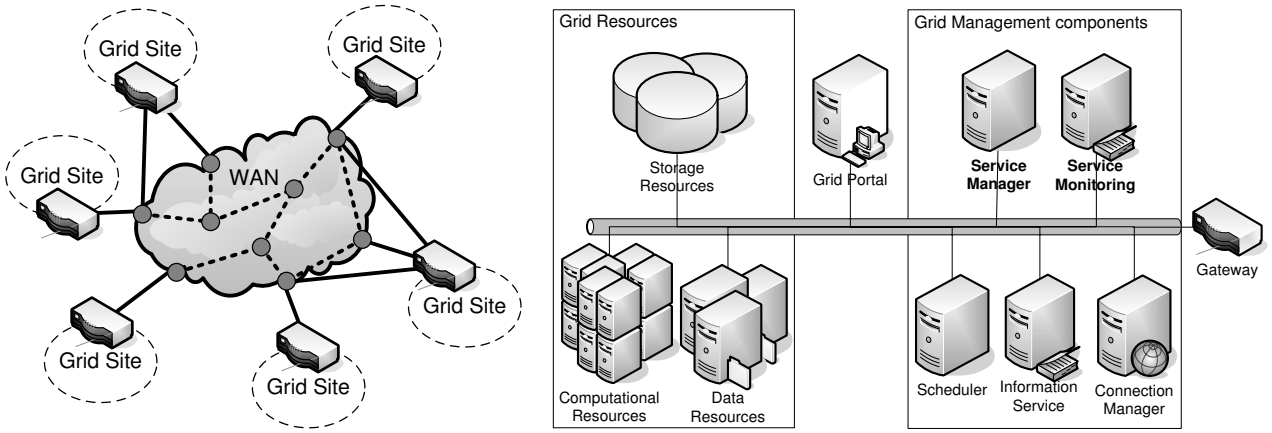
Widespread Grid adoption also increases the need for automated distributed management of Grids, as the number of resources offered on these Grids rises dramatically. Automated self-configuration and self-optimization of Grid resource usage can greatly reduce the cost of managing a large-scale Grid system, and at the same time achieve better resource efficiency and QoS support [3, 4].

The distributed service management architecture proposed in this paper can be described as a distinct implementation of the OGSA 'Service Level Manager' concept. Service Level Managers are, according to the OGSA specification, responsible for setting and adjusting policies, and changing the behavior of managed resources in response to observed conditions.

Our main goal is to automatically and intelligently assign Grid resources (both network, computing and data/storage resources) to a particular service class for exclusive use during a specified time frame. The decision to assign a resource to one particular service will be based on the resources available to the Grid and monitored service class characteristics.

In order to compare the performance of a service managed Grid versus a non-service managed Grid we use NSGrid (reported on in [5]), an ns-2 based Grid simulator capable of accurately modeling different Grid resources, management components and network interconnections. More specifically, we evaluated Grid performance (in terms of average job response time and resource efficiency) when different partitioning strategies are employed, and this both in case network aware as when network unaware scheduling is used.

This paper is structured as follows: section 2 summarizes related work in this area, while section 3 continues with an overview of the service management architecture and its interaction with other Grid components. Section 4 elaborates on the different resource partitioning strategies, while the evaluation of those partitioning strategies in a typical Grid topology is compared to a non-resource partitioned situation for varying job loads in section 5. Finally, section 6 presents some concluding remarks.



(a) Grid (b) Grid Site

Figure 1. Grid Model

2 Related Work

A lot of work has already been done in the area of distributed scheduling for Grids [6]. Grid scheduling taking into account service specific requirements has been dubbed *application-level scheduling*.

In AppLeS [7], service-class scheduling agents interoperable with existing resource management systems have been implemented. Essentially, one separate scheduler needs to be constructed per application type. Our service management architecture differs from this approach in that it operates completely separated from the Grid scheduling components, working in on service-exclusivity properties located at the Information Services (responsible for storing resource properties and answering resource queries from e.g. the different schedulers).

GrADS [8] on the other hand is a project to provide an end-to-end Grid application preparation and execution environment. Application run-time specific resource information comes from Network Weather Service and MDS2. For each application a performance (i.e. computational, memory and communication) model needs to be provided by the user. This differs from our Service Monitor approach, which actively monitors application behavior and deduces service characteristics at run-time (see 3.3).

Optimally assigning resources to services has been the subject of research in [9]. In this study however, resource selection occurs each time a job is submitted to a Grid Portal (i.e. service aware scheduling). This differs from the work proposed in this paper in which resources are pre-assigned to service classes based on service class characteristics (i.e. prior to the job scheduling process).

In contrast to the above mentioned research projects, our contribution focuses on distributed intelligent resource-to-service partitioning in a Grid environment.

3 Service Manager Concept

3.1 Grid/Job Model

We regard a Grid as a collection of *Grid Sites* interconnected by WAN links (see figure 1). Each Grid Site has its own resources (computational, storage and data resources) and a set of management components, all of which are interconnected by means of LAN links. Management components include a *Connection Manager* (capable of offering network QoS by providing bandwidth reservation support, and responsible for monitoring available link bandwidth and delay), an *Information Service* (storing registered resources' properties and monitoring their status) and a *Scheduler*. Every Grid resource in our model is given an associated service class ID property (stored in the Information Service with which the resource is registered). If no Service Management components are instantiated in the Grid, all resources' service class ID equals '0', meaning these resources can be used by *any* job (i.e. belonging to *any* service class).

The basic unit of work in our model is a *job*, which can roughly be characterised by its length (time it takes to execute on a reference processor), the needed input data, the output data size and the service class to which it belongs (when a client submits a job, the exact job properties are generated from pre-configured distributions). Each Grid Site has one or more *Grid Portals* through which clients can submit their jobs. Once submitted, a job gets queued at the local *Scheduler*, which in turn queries the Information Services (IS) (located both at the local site and at foreign sites) for resources adhering to the job's requirements. Once the results of those queries are in, the Scheduler applies one of its scheduling algorithms and (if possible) selects one or more Data Resources (DR) (for job input data),

together with one or more Storage Resources (SR) (for storing job output data) and a Computational Resource (CR) (for job processing), all *not necessarily* located at one Grid Site (note that DRs and SRs can reside on the same network node - modelling one data/storage capable resource).

If the scheduling algorithm is network aware, the Connection Manager (CM) is queried for information about available bandwidth on (shortest route) paths between resources and, once a scheduling decision is made (taking into account the speed at which I/O data can be fetched/stored to/from the processing job and adjusting computational power that gets reserved for this job to match), attempts to make connection reservations between the selected resources; connection reservations provide a guaranteed minimum bandwidth available for that job. Note that reservations are not physically set up by the CM: if the bandwidth requirements of the requested connection reservation are not infringing previously guaranteed connection reservations' minimum bandwidth, the request is granted. If however this is not the case, the connection reservation request is rejected and the job will be put back in the scheduler queue. Once all reservations are succesfull, the job is sent to the selected Computational Resource which fetches the different input datasets and stores the job's output data.

3.2 Resource Partitioning

Our goal is to intelligently and automatically assign service class IDs to each resource so they can be used exclusively for jobs spawned from that service class. This classification of Grid Resources in a per-service resource pool has multiple benefits:

1. resource efficiency and average job response times improve,
2. allows for faster and easier scheduling decisions,
3. service class priorities can be given by assigning more resources to high-priority service classes, and
4. locally offered service classes can be prioritised over foreign Grid Site service classes.

As we will see in section 5, resource efficiency (and average job response times) can be improved by limiting resource availability to service classes that can make efficient use of that particular resource (e.g. taking into account service class' data locality). In addition, the number of job resource query results returned by the Information Services to the Scheduler will be less than when there is one common resource pool, allowing faster scheduling decisions.

Of course, one has to be very careful when automatically assigning resources to service classes, as it creates the risk that certain service classes are (involuntarily) left starving for resources on which to run, while other resources are assigned to a service class for which there are no job submissions at that time (and are thus unnecessarily left idle). One also has to take into account service class necessities when making resource partitioning decisions, in order to avoid excluding a service class from access to a critical resource (e.g. prohibiting a service class access to mandatory data resources).

The same way computational, storage and data resources can be partitioned amongst different service class resource pools, network resources can also be split up by performing per-service bandwidth reservations (e.g. VPN technology). This can prevent data-intensive service classes from monopolising network bandwidth usage and thereby hampering the performance of jobs from other service classes (see figure 2). Instead, each service class should automatically receive a certain bandwidth and be able to use this bandwidth without having to worry about the network usage of other services' jobs.

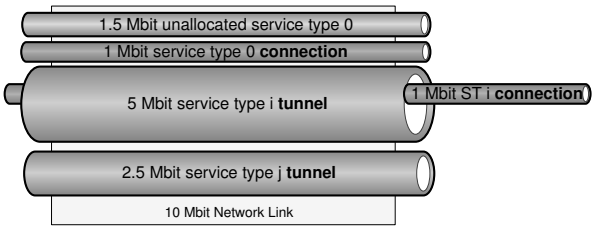


Figure 2. Network Resource Partitioning

With combined network and resource partitioning, a Grid can be modeled as a dynamic collection of overlay Grids or *Virtual Private Grids* (VPG), with one VPG for each service class offered in the Grid. These VPGs are not static structures in that they do not have resources assigned to them in a permanent way, but react to monitored changes in service characteristics (e.g. additional service offerings can lead to the construction of new VPGs and reallocation of resources across existing VPGs). Resource reallocation can stem from important changes in monitored service class characteristics (e.g. higher job submission rates for a service class), a change in service class priorities or, as already mentioned, the addition of new service classes.

3.3 NSGrid implementation

In NSGrid, a distributed service management architecture was implemented in order to evaluate the effectiveness of different resource partitioning strategies. Each Grid Site has a local *Service Manager*, which interacts with the local Information Service (IS), Connection Manager (CM) and *Service Monitor* (all control messages are XML-encoded

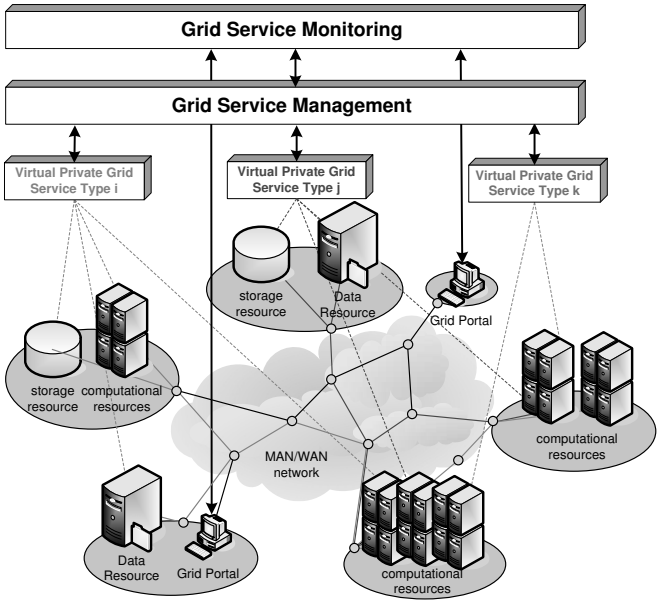


Figure 3. Virtual Private Grid Partitioning

and sent over the underlying ns-2 network). Every locally offered Grid service needs to be registered with the Service Monitor (together with an associated priority): this gives the service class a unique ID and permits users to submit jobs from that service class. The Service Monitor monitors each local service class: it stores job inter arrival times (IATs), ID-based input and output data requirements and processing length (compared to a reference processing speed) of jobs. At specified intervals, the Service Monitor sends the collected service class' characteristics to all known foreign Service Monitors, so they can keep their '*foreign service class* information up-to-date. Each Service Monitor thus holds two types of information: on one hand characteristics pertaining to locally offered service classes and on the other hand foreign service class information (note that the same service class can be offered at more than one Grid Site).

The Service Manager in turn queries the local Service Monitor for information regarding the different service classes. When the monitored service characteristics do not differ (with regard to a certain treshold) from the ones used to partition the Grid resources in a previous partitioning run, no repartitioning will occur. If however the difference between the previous values and currently monitored service characteristics (average job IAT, processing length, I/O bandwidth necessities, etc.) is too big, or if no resource partitioning has yet been done, the Service Manager will query the ISs for all locally offered Grid resources. Once the answer to this query has been received, one of the resource partitioning algorithms (detailed in section 4) is applied to the resource set, and the resulting resource partitioning solution is sent back to the ISs, who in turn change the service class property of their registered resources. If the partitioning algorithm also works in on network resources, the CM will be contacted to make service bandwidth reservations (based on assigned CRs, necessary input datasets and monitored service class' bandwidth requirements).

It is important to note that, while resources are assigned for exclusive use by a particular service, not one job using that resource will be interrupted (preventing jobs from being preempted when the CR it is running on is assigned to a different service class). The service assignment will thus only be effective for *new* jobs or jobs currently in the scheduler queue. At the time of scheduling, queries will be sent to the ISs for resources adhering to the job's requirements, and the ISs will return only those resources that are assigned to that particular job's service class.

4 Partitioning Strategies

Recall that we are trying to partition resources into service class resource pools. A solution in this case is a mapping from resource to a particular service class ID, and this for all resources returned from the Service Manager - IS queries. A resource can also be assigned ID '0', meaning it can be used by every service class. Exhaustively searching for an optimal partitioning (by evaluating the fitness of a solution by means of a cost function) quickly becomes infeasible, as the amount of solutions that needs to be evaluated is $(\#serviceclasses+1)^{\#resources}$ (e.g. if a Grid Site has 24 operational Computational Resources and 2 service classes are offered, one has to check $3^{24} = 282429536481$ possible service assignments). To overcome this limitation, heuristics are needed. Due to the nature of the solution space (finite set of discrete resource - service ID assignments), the use of a genetic algorithm heuristic (GA) is preferred. In algorithm 4.1 ρ_C denotes crossover probability, while ρ_M denotes mutation probability. The fitness of each solution is evaluated by means of a to-be-maximised cost function $f(x)$, detailed in the next sections.

Depending on how much time is available between partitioning runs (which in turn depends on the stability of the different services characteristics), parameters of this GA can be tuned in such a way that feasible search times can be attained (i.e. search time $<<$ time between partitioning runs). It is the duty of the Service Monitor to detect when services show relevant behavioral changes and push this info to the Service Manager. The same way the Service Monitor tracks changes in service characteristics, the Information Services are responsible for signaling changes in Grid resource availability. Sudden resource unavailability has to be reported to the Service Manager, which in turn is responsible for assessing the impact of this change on the affected service classes. The Service Manager can act by first releasing some or all service-assigned resources for

use by new services and second, once service characteristics have been normalised, can repartition the available resources in Virtual Private Grids.

Algorithm 4.1: GENETIC ALGORITHM($resources$)

$population_{initial} \leftarrow (b_{(1,0)}, ..., b(m,0)),\ t \leftarrow 0$
while $stopcondition$ **false**
do
- **comment:** proportional selection
- **for** $i \leftarrow 1$ **to** m
  - **do**
    - $x \leftarrow rand[0,1],\ k \leftarrow 1$
    - **while** $k < m$ **and** $x < \sum_{j=1}^{k} \frac{f(b_{j,t})}{\sum_{j=1}^{m} f(b_{j,t})}$
      - **do** $k \leftarrow k + 1$
    - $b_{i,t+1} \leftarrow b_{k,t}$
- **comment:** two-point crossover
- **for** $i \leftarrow 1$ **to** $m - 1$ **step** $i + 2$
  - **do**
    - **if** $rand[0,1] \leq \rho_C$
      - **then**
        - $pos1 \leftarrow rand[1,m]$
        - $pos2 \leftarrow rand[1,m]$
        - **if** $pos1 > pos2$
          - **then** $switch(pos1, pos2)$
        - **for** $k \leftarrow pos1$ **to** $pos2$
          - **do** $switch(b_{i,t+1}[k], b_{i+1,t+1}[k])$
- **comment:** mutation
- **for** $i \leftarrow 1$ **to** m
  - **do**
    - **for** $k \leftarrow 1$ **to** m
      - **do**
        - **if** $rand[0,1] < \rho_M$
          - **then** $b_{i,t+1}[k] \leftarrow rand[0, \#ST]$
- $t \leftarrow t + 1$

In the next sections we provide details on some implemented partitioning strategies (and accompanying cost functions). We assume that the Service Manager has received both up-to-date local and foreign service characteristics from the Service Monitor and resource properties from the Information Service.

4.1 Local Service CR Partitioning

The first (and simplest) partitioning strategy only takes into account the computational processing needs and priority of the different *local* service classes. The Service Manager queries the IS for all local CRs and calculates average service class' requested processing power[1]:

$$\forall ST \cdot ppower_{req_{ST}} = sites_{ST} \times \frac{ptime_{ref_{ST}}}{IAT_{ST}}$$

The relative processing power assigned to a service class (sum of processing power of CRs assigned to that ST) can be described by[2]:

$$\forall ST \cdot ppower_{asg_{ST}} = \sum_{\forall CR \in ST} \frac{speed_{CR}}{speed_{CR_{ref}}} \times ptime_{ref_{ST}}$$

Once CR query answers have been received, GA 4.1 will be started with following cost function $f(x)$:

Algorithm 4.2: $f_{CRpart_{local}}(x)$

$result \leftarrow \frac{ppower_{asg_0}}{2}$
$maxAlloc_{over} \leftarrow 0,\ maxAlloc_{under} \leftarrow 0$
for $i \in ST_{local}$
do
- $aux \leftarrow ppower_{req_i} - ppower_{asg_i}$
- **if** $aux < 0$
  - **then**
    - **if** $-aux > maxAlloc_{over}$
      - **then** $maxAlloc_{over} \leftarrow -aux$
    - $aux \leftarrow ppower_{asg_i}$
  - **else**
    - **if** $\frac{aux}{ppower_{req_i}} > maxAlloc_{under}$
      - **then** $maxAlloc_{under} \leftarrow \frac{aux}{ppower_{req_i}}$
    - $aux \leftarrow ppower_{asg_i} - aux$
- $result \overset{+}{\leftarrow} \frac{priority_i}{(\sum_{j \in ST_{local}} priority_j)} \times aux$

$result \overset{-}{\leftarrow} maxAlloc_{over} + maxAlloc_{under}$

In this cost function (which is to be maximised), the objective is to donate to each *local* service class the same amount of processing power *relative* to their requested processing power (giving a higher cost function impact factor to service classes that have a high priority). The $maxAlloc_{over}$ and $maxAlloc_{under}$ parameters assure an even spread of processing power to services (both in case insufficient processing power is available as when sufficient processing power is available).

4.2 Global Service CR Partitioning

The second partitioning strategy adds support for services offered at foreign grid sites. The importance of assigning resources to foreign service classes can be adjusted by the local Service Manager by tweaking the foreign service policy $\rho_{ST_{foreign}}$. Support for foreign service classes can range from no impact at all on the cost function ($\rho_{ST_{foreign}} = 0$) to an impact equal to that of local service classes ($\rho_{ST_{foreign}} = 1$) or any value in between. The resulting cost function is stated in algorithm 4.3:

[1] $ptime_{ref_{ST}}$ = average processing time of service class ST job on reference CR, $sites_{ST}$ = amount of Grid portals launching service class ST's jobs, IAT_{ST} = average service class ST's job interarrival time

[2] $speed_{CR}$ = processing speed of CR, $speed_{CR_{ref}}$ = processing speed of reference CR

Algorithm 4.3: $f_{CRpart_{global}}(x)$

$result \leftarrow \frac{ppower_{asg_0}}{2}$
$maxAlloc_{over} \leftarrow 0,\ maxAlloc_{under} \leftarrow 0$
for $i \in ST_{local} \cup ST_{foreign}$
do
- $aux \leftarrow ppower_{req_i} - ppower_{asg_i}$
- **if** $aux < 0$
  - **then**
    - **if** $-aux > maxAlloc_{over}$ **then** $maxAlloc_{over} \leftarrow -aux$
    - $aux \leftarrow ppower_{asg_i}$
  - **else**
    - **if** $\frac{aux}{ppower_{req_i}} > maxAlloc_{under}$ **then** $maxAlloc_{under} \leftarrow \frac{aux}{ppower_{req_i}}$
    - $aux \leftarrow ppower_{asg_i} - aux$
- **if** $i \in ST_{foreign}$ **then** $aux \leftarrow aux \times \rho_{ST_{foreign}}$
- $result \stackrel{+}{\leftarrow} \frac{priority_i}{(\sum_{j \in ST} priority_j)} \times aux$

$result \stackrel{-}{\leftarrow} maxAlloc_{over} + maxAlloc_{under}$

4.3 Input Data Locality Penalisation

Resource partitioning based solely on the processing needs of the different services can lead to bad performance. In case of data-intensive services in particular, one wants these services to be processed on computational resources located near input data that is generally requested by those service classes. In order to provide this functionality, the Service Manager queries the Information Services for both Computational and Data Resources and constructs a list of which CRs have local access (i.e. accessible from the local Grid Site) to which input sets. We adjust the cost function to include this notion and penalise (relative to the data-intensiveness of the service class) assigning a Computational Resource that has no local access to an input dataset much-needed by the assigned service[3]

$$cost_{CR \in ST_i} = \frac{\frac{InputReq_i}{IAT_i}}{\sum_{\forall j \in ST} \frac{InputReq_j}{IAT_j}} \times \frac{\rho_{cost}}{\#CR_{assigned_i}}$$

An additional (yet larger) penalty is given when, amongst all Computational Resources assigned to a particular service, *not one of them* has access to a needed dataset, as it can be considered best practice that at at least one Computational Resource can access a needed input set locally. This cost is only charged once for each service class.

$$cost = \frac{\frac{InputReq_i}{IAT_i}}{\sum_{\forall j \in ST} \frac{InputReq_j}{IAT_j}} \times \rho_{cost}$$

[3] $InputReq$ = avg. service class's input size requirement, $OutputReq$ = avg. service class's output size requirement, $\#CR_{assigned}$ = amount of CRs assigned to service class, ρ_{cost} = data non-locality penalty factor

Both costs can be used as a penalty for the cost function in algorithm 4.2 and 4.3.

4.4 Network Partitioning

Since the Service Monitor keeps track of I/O data characteristics of each service, data intensiveness relative to the other services can be calculated. This in turn can be used to perform per-service network bandwidth reservations. We have implemented a proof-of-concept network partitioning strategy, in which the Service Manager calculates average data requirement percentages for each service class i [4]

$$bw_{req_i} = \frac{\frac{bw_{input_i} + bw_{output_i}}{IAT_i}}{\sum_{\forall j \in ST} \frac{bw_{input_j} + bw_{output_j}}{IAT_j}}$$

and passes this information to the Connection Manager, who in turn will make service class bandwidth reservations on all network links for which it is responsible. Network partitioning can be applied to all previously mentioned partitioning algorithms.

5 Performance Evaluation

5.1 Resource setup

A fixed Grid topology was used for all simulations (run on an AMD Athlon XP1700+ processor with 1 GB RAM, and Debian Woody). First, a WAN topology (containing 8 core routers with an average out-degree of 3) was instantiated using the *GridG* tool [10]. Amongst the edge LANs of this topology, we have chosen 12 of them to represent a Grid site. Each site has its own resources, management components and Grid portal interconnected through 1Gbps LAN links, with Grid site interconnections consisting of dedicated 10Mbps WAN links. A Service Manager was instantiated, and was given access to the different Grid Sites' Information Services.

We have assigned 3 CRs to each Grid Site (for a total of 36 CRs). To reflect the use of different tiers in existing operational Grids, not all CRs are equivalent: the least powerful CR has two processors (which operate at the reference speed). A second class of CRs has four processors, and each processor operates at twice the reference speed. The third - and last - CR type contains 6 processors, each of which operates at three times the reference speed. Conversely, the least powerful type of CR is three times as common as the most powerful CR, and twice as common as the middle one.

[4] bw_{input} = avg. service class's input bandwidth need: $\frac{speed_{CR}}{speed_{CR_{ref}}} \times \frac{InputReq}{ptime_{ref}}$, bw_{output} = avg. service class's output bandwidth need: $\frac{speed_{CR}}{speed_{CR_{ref}}} \times \frac{OutputReq}{ptime_{ref}}$

It is assumed that all processors can be time-shared between different jobs.

We have assumed that SRs offer "unlimited" disk space, but are limited in terms of access/write speed by the bandwidth of the link connecting the SR to the Grid Site. Each site has at its disposal exactly one such SR. Each site's DR contains 6 out of 12 possible data sets. These data sets are distributed in such a way that 50% of the jobs submitted to a site can have local access to their needed data set.

5.2 Job parameters

We have used two different, equal-priority service classes (each accounting for half of the total job load) in our simulations; one is more data-intensive (i.e. higher data sizes involved), while the other is more cpu-intensive. At *each* Grid Site, two "clients" have been instantiated, one for each job type. Each client submits mutually independent jobs to its Grid Portal. All jobs need a single IR and a single SR. The ranges between which the relevant job parameters vary have been summarized in table 1. In each simulation, the job load consisted of 1308 jobs. For each scheduling algorithm, we chose to use a fixed interval of $20s$ between consecutive scheduling rounds.

	CPU-Job	**Data-Job**
Input(GB)	0.01-0.02	1-2
Output(GB)	0.01-0.02	1-2
IAT(s)	30-40	30-40
Ref. run time(s)	100-200	40-60

Table 1. Relevant service class properties

5.3 GA partitioning

Since an exhaustive search for an optimal resource-to-service partition is infeasible (see section 4), we evaluated the performance of the implemented GA resource partitioning heuristics by letting a central Service Manager partition the 36 CRs in our topology. We used Grefenstette's GA settings [11], with an initial population of 30, $\rho_C = 0.9$ and $\rho_M = 0.01$, and a stop condition of 100 runs. Figure 4 shows the trend of the best cost function optimum for different GA generations. The cost function used is the one discussed in section 4.1 (Local Service CR partitioning) with Input Data Locality penalisation. Average time needed to calculate the final resource-to-service partition was $850s$ ($8.5s$ per generation; note that this time is not exclusive for GA solution calculation, but is also spent on all other simulation tasks during partitioning), but from figure 4 we can see that a 30 to 35% improvement can be attained by using a more intelligent stop condition (i.e. stop when only a small improvement is made over a certain number of runs). During partitioning, Grid operation continues as normal, as the Service Management components do not block any other management components. In case faster partitioning times need to be attained, one can deploy a Service Monitor/Service Management at every Grid Site, who are then responsible for communicating with the foreign site's Service Monitor components and partitioning the resources at their assigned site (as described in section 3.3).

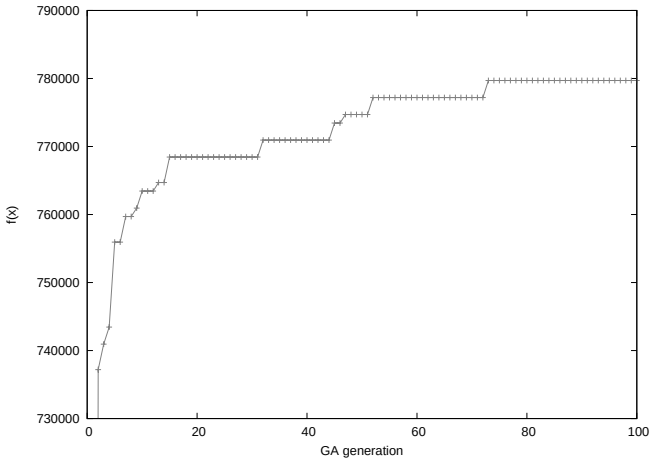


Figure 4. Genetic Algorithm optimum trend

5.4 Job response time

We define the *response time* of a job as the difference between its end time and the time it is submitted to the scheduler. In figure 5 we present this average job response time for different scenarios, comparing both the Service Managed versus the non-Service Managed case, while at the same time evaluating the different partitioning strategies discussed in previous sections for both network aware and non network aware scheduling algorithms (detailed in [12]). The results show that average job response times can be improved significantly when employing a resource partitioning algorithm prior to scheduling. This behavior can be explained because resources are reserved for exclusive use by a service class. It is this service-exclusivity that forces the scheduler to not assign jobs to less-optimal resources (e.g. non-local access to needed input data, low processing power available,...), but to keep the job in the scheduling queue until a service-assigned resource becomes available.

5.5 Resource Efficiency

Using the same job load, total network usage dropped by 8% when network unaware scheduling was employed, and by 4% when a network aware scheduling heuristic was used

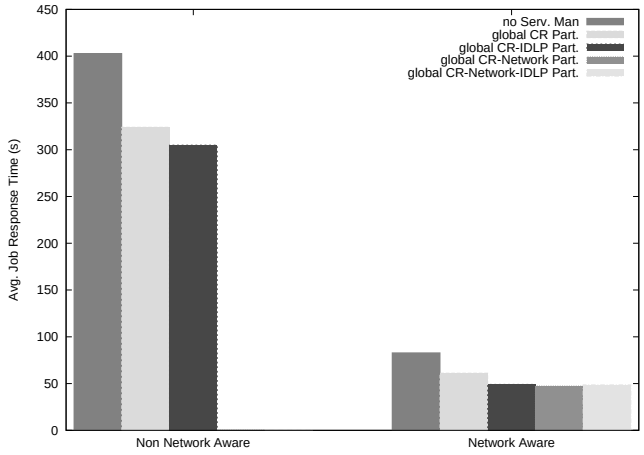


Figure 5. GA-GSCR performance

(compared to the non-service managed case), due to the fact that input/output data was located at resources closer to the job's service class' assigned CRs.

Furthermore, we calculated the average Computational Resource utilization:

$$\frac{\sum_{j \in Jobs_{CR}} Load_j}{Makespan \times speed_{CR}}$$

The improvement obtained by employing resource-to-service partitioning when using network unaware scheduling equals 20.5%, whereas in the case where network aware scheduling is used, it is 17%.

5.6 Scheduling

We measured the time it takes to calculate a scheduling decision and noticed a decrease in scheduling time of 28% when comparing the service managed Grid to the non-service managed Grid in case network aware scheduling is used (i.e. from an average $12.71s$ in the non service managed case to 9.13s in the service managed Grid). This behavior can be explained by the fact that a scheduler queries the Information Services for resources adhering to a job's requirements *and* assigned to either the job's service class or the standard service class 0. When resources are partitioned amongst services, less results will be returned to the scheduler, allowing for easier schedule making decisions.

6 Conclusions

In this paper we proposed the use of a distributed service management architecture, following the OGSA 'service level manager' concept, capable of monitoring service characteristics at run-time and partitioning Grid resources amongst different priority service classes. A variety of partitioning algorithms was discussed and their performance was evaluated using NSGrid. Our results show that the proposed service management architecture can ease the process of making schedule decisions, improves Grid resource efficiency and job turnaround times, while at the same time remaining unobtrusive. Furthermore, we believe that the Virtual Private Grid concept offers an exciting avenue for future research.

References

[1] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke, "*Grid services for distributed system integration*", IEEE Computer, Vol. 35-6, pp. 37-46, 2002

[2] I. Foster et al., "*The Open Grid Services Architecture, Version 1.0*", draft-ggf-OGSA-spec-019, `http://forge.gridforum.org/projects/ogsa-wg`

[3] J.O. Kephart, D.M. Chess, "*The vision of autonomic computing*", IEEE Computer, Vol. 36-1, pp. 41-50, 2003

[4] A.G. Ganek, T.A. Corbi, "*The dawning of the autonomic computing era*", IBM Systems Journal, Vol. 42-1, pp. 5-18, 2003

[5] B. Volckaert, P. Thysebaert, F. De Turck, P. Demeester, B. Dhoedt, "*Evaluation of Grid Scheduling Strategies through a Network-aware Grid Simulator*", Proc. of PDPTA 2003, Vol. 1, pp. 31-35, 2003

[6] K. Ranganathan, I. Foster, "*Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*", Journal of Grid Computing, Vol. 1-1, pp. 53-62, 2003

[7] F. Berman et al., "*Adaptive Computing on the Grid Using AppLeS*", IEEE Transactions on Parallel and Distributed Systems, Vol. 14-4, pp. 369-382, 2003

[8] H. Dail, F. Berman, H. Casanova, "*A Decoupled Scheduling Aproach for Grid Application Development Environments*", Journal of Parallel and Distributed Computing, Vol. 63-5, pp. 505-524, 2003

[9] H.L. Lee et al., "*A Resource Manager for Optimal Resource Selection and Fault Tolerance Service In Grids*", CCGrid 2004, pp. 572-579, 2004

[10] D. Lu, P. Dinda, "*GridG: Generating Realistic Computational Grids*", Performance Evaluation Review, Vol. 30-4, 2003

[11] J.J. Grefenstette, "*Optimization of control parameters for genetic algorithms*", IEEE Trans. Systems, Man, and Cybernetics, Vol. 16-1, pp. 122-128, 1986

[12] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, "*Network Aspects of Grid Scheduling Algorithms*", Proc. of PDCS 11, pp. , 2004