Problemi di schedulazione distribuita su Grid

• Ivan Porro
• Università degli Studi di Genova, DIST, Laboratorio BioLab
  • pivan@unige.it
  • 010-3532789
• Si ringrazia per parte del materiale il Dr. Andrea Clematis dell’ IMATI –CNR di Genova
Unit

1. Grid scheduling con Service Level Agreement
2. SNAP e SLA
  • /* References */
Grid scheduling con
Service Level Agreement
Traditional scenario

• scheduling mechanisms used for the enactment of jobs on parallel supercomputer resources are queue-based

• only one level of service
  – run the job when it gets to the head of the queue

• in some cases advanced method such as backfilling:
  – jobs that are behind in the priority queue may be assigned to unutilized processors
  – ensure that there are no idle processors
Grid scenario

• The emergence of Grid Computing has created new opportunities to support compute and/or data intensive scientific applications

• However, at the same time, Grid computing is built upon the notion of virtual organizations
  – a group of individuals and/or institutions who collaborate towards the solution of a particular problem through a set of resource sharing rules
  – collaboration of the members may span across different administrative domains
Grid issue(s)

• scheduling independent jobs of a large applications in Grid requires
  – some coordination
  – across the different schedulers
  – of the individual resources employed

• lack of coordination may annul all the benefits from parallelism that one might expect when independent jobs are running, in parallel, onto different resources
Scenario

- A large application that can be represented by a Directed Acyclic Graph (DAG):
  - Several jobs
  - nodes of the graph represent individual jobs with individual computational requirements
  - edges represent communication of data

- In the context of Grid computing, DAGs can be used to represent many applications that belong to the important family of applications collectively known as scientific workflows
Scenario /2

- We want to exploit the task parallelism available in the DAG, by executing independent nodes in parallel
- Assume that two different nodes are assigned to different resources
- Each resource
  - belonging to a different administrative domain
  - using its own scheduler with its own queuing system
Problem!

- Each job will start execution when it reaches the head of the local queue.
- It is possible that one job may finish execution before the other job has even started.
- Different behavior of the queue-based schedulers of the local resources, they act:
  - without coordination
  - contrary to user expectations
- From a global point of view result is:
  - this is equivalent to executing the two jobs sequentially
  - there are no performance benefits due to the exploitation of parallelism
A solution: advance reservation

- Has been suggested as a means to guarantee that tasks will run onto a resource when the user expects them to run
- Essentially, specifies a precise time that jobs may start running
- This allows the user to:
  - request resources from systems with different schedulers
  - for a specific time interval (e.g., start time, finish time)
  - obtaining a sufficient number of resources for the time s(he) may need
- Has already received significant attention
- An important requirement for future Grid resource management systems
- There has been already significant progress on supporting it by several projects and schedulers (LSF, Maui, COSY, EASY)
Problems

• It is an extreme level of service since it specifies a precise time when jobs can be made to run
• There is some skepticism in the community (to the degree to which advance reservations contribute to improving the overall performance of a scheduler)
• Several problems to the resource owner
  – when an advance reservation is made, the scheduler must place jobs around this fixed job
  – this is done using backfilling, which increases utilisation by searching the work queues for small jobs to plug the gaps
Problems /2

• In practice, this rarely works perfectly, and so the scheduler must either
  – leave the reserved processing elements empty for a time
  – suspend or checkpoint active jobs near to the time of the reservation

• These processes are not instantaneous:
  – Checkpointing a 64 processor Unified Weather Model job on an O3800 takes about 12 minutes and a small total memory footprint of 3Gb
  – Checkpointing a 256 processor jobs can exceed one hour
Problems /3

- Suspension is faster, but can adversely affect the performance of the incoming job, due to the cost of swapping out memory used by the suspended job when it is required by the incoming job.
- There are gaps in the schedule, i.e., CPU time which is not processing users' work.
There is a tendency to offset the cost of the unused time by charging for advance reservation jobs at a considerably higher tariff. While it is possible to set tariffs high enough to compensate:
- this brute-force solution is inefficient in terms of resources, and undesirable
- for both users, who pay higher prices
- for resource owners, who must charge uncompetitive prices and watch utilization fall
Current trend: SLA based scheduling

- Explore the space between the two aforementioned extreme levels of service
  - run this job whenever it gets to the head of the queue
  - run this job at this precise time

- Provide different levels of service by forging agreements between the different parties
  - User
  - resource owner
What is a SLA?

- Agreements
  - Agreed on the basis of different constraints expressed by the user and/or the resource owner
  - essentially specify a desired (agreed!!) level of service
- The use of Service Level Agreements (SLAs) gives rise to a fundamentally new approach for job scheduling on the Grid.
- Negotiated between
  - a client (user, superscheduler, or resource broker)
  - a provider (owner of a resource with its own scheduler)
- May contain information such as acceptable job start and end times
- May be re-negotiated during runtime
Argomenti

- background
- key features of an architecture for job scheduling using SLAs on the Grid
- key issues and challenges
- approaches to address some of these issues
  - the description of the terms in an SLA
  - the design of heuristics for SLA scheduling
Definition of a SLA

• An agreement (legally binding contract) between the parties involved
• It relates to a transaction for the provision of a service (€€€!!!)
• As a result
  – the parties of an SLA can be distinguished between providers and consumers of a service
  – the terms of the SLA describe the expected level of service within which the service will be provided
  – these have been agreed between service providers and service consumers
History

- Forms of SLAs were in operation since the 1960s, when they were used as a method for buying minutes of computer machine time.
- In recent years, SLAs became more widespread as a means to make agreements when outsourcing IT functions:
  - On-line backup
  - Outsourcing of ERP/CRM
  - Hosting of web sites and web applications
- SLAs are traditionally paper-based and drawn after some form of negotiation between appropriate persons.
SLAs for Grid

• In the context of Grid and Web services, the current understanding of the community is that such an SLA
  – is essentially an electronic contract
  – it is expected to be negotiated fully automatically (i.e., without any human intervention) by different processes

• Must be machine readable and understandable
SLA research activities

- Incorporation into grid architectures
- Specification of the SLAs (language, tools)
- Usage of SLAs for resource management on the Grid
- Negotiation of SLAs:
  - early work on the Service Negotiation and Acquisition Protocol (SNAP)
  - further work has argued for the need to take into account the principles of contract law
- Target: negotiating SLAs in order to form legally binding agreements, even considering how the consequences from the use of SLAs may need to be taken into account by guidelines on electronic contracts
SLA research activities /2

- Trust and security
- Business oriented case studies and analyses
- Agents
- Model negotiation
  - heuristics
  - game theoretic
  - argumentation-based approaches
- Economic aspects associated with the usage of SLAs for service provision
  - charges for successful service provision
  - penalties for failure
Standards

- Grid Resource Allocation Agreement Protocol (GRAAP) Working Group of the Open Grid Forum
- Development of WS-Agreement (WS-A), a specification for a simple generic language and protocol to establish agreements between two parties
- Each of the two parties can be either an initiator of or a responder to the agreement
- The agreement structure is composed of several distinct parts, namely
  - Name
  - Context
  - Terms of Agreement
    - service description terms
    - guarantee terms
Example (WS-Agreement)

- `<wsag:Agreement AgreementId="xs:string">`
- `<wsag:Name>`
- `xs:string`
- `</wsag:Name>`
- `<wsag:AgreementContext>`
- `wsag:AgreementContextType`
- `</wsag:AgreementContext>`
- `<wsag:Terms>`
- `wsag:TermCompositorType`
- `</wsag:Terms>`
- `</wsag:Agreement>`

- **Web Services Agreement Specification (WS-Agreement)**
• **Service descriptions** terms mainly describe the functionality to be delivered under the agreement.

• The **guarantee** terms define the assurance on service quality for each item.

• Items are mentioned in the service description terms section of the WS-A.

• In the specific context of job submission, such assurances may be defined as a parameter (constant) or bounds (min/max) on the availability of part or the whole of the resource.
• In WS-A, such assurances are referred to as service level objectives (SLOs)
• in a domain specific to computation services provision, they are usually expressed as values
  – e.g., SLO: CPUcount = 8
• Each SLO may refer to one or more business values, called a business value list (BVL)
• This list expresses different value aspects of a specific SLO
• The other two types of guarantee terms are Qualifying Conditions and Importance, which have a similar function to SLO and BVL, respectively.
A SLA-enabled architecture

• There are three key “players” underpinning this materialization
  – Users negotiate and agree an SLA with a resource broker (or superscheduler, or coordinator)
  – Brokers negotiate and agree an SLA with users; these SLAs may be mapped to one or more SLAs, which are negotiated and agreed with local resources and their schedulers
  – Finally, local schedulers need to schedule the work that is associated with an SLA which they agreed to

• the constraints associated with such an SLA, agreed by a resource, may be stored locally in the resource, in some kind of a resource record
Grid Scheduling and SLA
Feature

- A single SLA agreed between a user and a broker may "translate" to multiple SLAs between the broker and different local resources to serve the user's request.
- This could be the case when the SLA between a user and a broker refers to a workflow application with several tasks that are executed on different resources.
- The user may want to set constraints for the workflow as a whole and the broker may have to translate it to specific SLAs for individual tasks.
• to indicate the possible differences between these two types of SLA, the terms meta-SLA and sub-SLA are used

• Furthermore, as indicated in the figure, this SLA-based view for job submission, may still allow the submission of jobs that are not associated with an SLA

• however, no guarantees about their completion time would be offered in this case.
Issues: vocabulary

- The vision of SLA based scheduling assumes that the SLAs themselves are machine readable and understandable.
- This implies that any agreements, between the parties concerned, for a particular level of service need to be expressed in a commonly understood (and legally binding) language.
- There has been early work on generic languages for SLA description, but none related to the particular problem of the requirements associated with job submission and execution (on possibly high performance computing resources).
Issues: negotiation

- SLAs may be negotiated between machines and users or only between machines. In this negotiation some commonly agreed protocol needs to be followed. This protocol needs to take into account both the nature of the distributed systems and networks which are used for the negotiation
  - what if an offer from one party is not received by the other party due to a network failure?
  - should the receipt of every proposal for an agreement be acknowledged or not?
- In addition, during negotiation, machines should be able to reason about whether an offer is acceptable and possibly they should be able to make counter-offers
- The relevant challenges on negotiation are more of a generic nature rather than specific to the problem of using SLAs for job scheduling.
Issues: scheduling

• Scheduling of jobs on high-performance compute resources is mostly based on priority queues (with the possible addition of backfilling techniques).
• The use of SLAs would require the development of a new set of algorithms for efficient scheduling, which would be based on satisfying the terms agreed in the SLA.
• Efficient scheduling algorithms are researched because they would help to estimate capacity and reason on the possible acceptance (by a local resource) of a new request to make an SLA.
Issues: constitutional aspects

- In the context of any SLA based provision, sooner or later, the need for dispute resolution may arise.
- Users may also be interested in the reliability of specific brokers:
  - for example, how likely (or unlikely) is that a broker will honour an SLA (even if breaking the SLA would require the broker to pay a penalty)?
- This issue of modelling reputation may also be related to the approaches followed for pricing and/or penalties when agreeing SLAs:
  - is there a flat charge for the usage of resources?
  - do the fees vary depending on particular circumstances?
- Since relevant work is at early stage, only simple assumptions can be made:
  - without neglecting the challenges that might arise later!
It is not easy

- Incorporate SLAs in distributed grid computing is not as easy as defining a dictionary of terms which are then placed in some kind of an XML derived structure.
- Several debates about what an SLA should contain and/or how it should look like.
- Example:
  - one of the difficulties that had to be addressed related to how issues related to the SLA document description would be decoupled from issues related to the negotiation protocols.
• An efficient SLA related environment, with guarantees for a certain level of fault tolerance, may often include **renegotiation** of the whole or part of an SLA

• At high resource load, a significant proportion of the agreed SLAs may have to be renegotiated in order to avoid failure of an SLA

• Such renegotiation may have an overhead and may require the user's participation at one or more stages

• This is not possible in Grids: one would expect some abilities of **self-management**
Expressiveness + & -

• In most cases of SLA-based resource management applications, the set of guarantee terms is rigidly defined.
• Within such an SLA, it is known in advance, for example, what the exact financial gain will be when an SLA is audited.
• This arrangement is not flexible and carries too little information for the provider bound by this SLA to perform its tasks successfully without the need for renegotiation.
  – No one would take the risk of an deeply defined SLA at reasonable costs.
• The expressive capacity of an agreement can be improved by
  – adding more terms, but there is a practical limit on the SLA size
  – describe terms analytically.
Analytical functions

- The reservation of resources for a computational job where using a different number of parallel processors to run the job would still result in a successful SLA from the user point of view.
- The running time would vary, depending on the parallelism assigned to it by the scheduler (# of cpu assigned).
- The use of analytical functions means that the scheduler can now schedule and repeatedly reschedule this job in order to meet the objectives of its associated SLA without the need for renegotiation.
Analytical functions example

- The SLA terms relating to the size of the job (with respect to execution time needed and how this may relate to CPUs used) may look like:

  - Maximum number of CPU Nodes that can be used, $N_{CPU}^{max}$
  - CPU Nodes reserved, $N_{CPU} = \{2, 3, 4, ..., N_{CPU}^{max}\}$
  - Reserved time for job execution, $t_D = \frac{t_D^1}{N_{CPU}}$

where $t_D^1$ is the projected time for the job to complete if it runs on a single CPU Node; and $N_{CPU}^{max}$ is limited by the capacity of the resource. It can be seen that the reserved time for job execution is described as a function of the number of CPU nodes reserved. If a function was not used, then the SLA would have to include a single value for each of $N_{CPU}$ and $t_D$; any attempt to deviate from these values would trigger the renegotiation of the SLA.
Scheduling heuristics

- Local schedulers must now take into account
  - functional properties of the submitted job (such as parallelism and execution time)
  - non-functional requirements expressed as terms and constraints in SLA
- the existences of various constraints that need to be satisfied may point to a constraint satisfaction problem
- standard methods based on various forms of (exhaustive) searching would not be practical in a grid environment where a large number of SLAs may have to be negotiated and scheduled quickly
- Simple scheduling heuristics could provide efficient scheduling solutions with negligible time overheads for a large set of SLAs
- The concept is based on how SLAs would be prioritised
  - picking jobs (in order of priority) from the prioritised list
  - fitting them onto the resource, without rescheduling previous jobs already allocated
Priority (H) calculation:

\[ H = \min (h_1 + w \cdot h_2) \]  \hspace{1cm} (1)

\[ H = \max (h_1 + w \cdot h_2) \]  \hspace{1cm} (2)

where \( h_1, h_2 \) is one of:

- Earliest job start time, \( T_s \)
- Latest job finish time, \( T_f \)
- Reserved time for job execution, \( t_D \)
- Number of CPU Nodes required, \( N_{CPU} \)
- Job size, measured in CPU-hours, \( A = N_{CPU} \times t_D \)
- Job laxity, defined as \( t_L = T_f - (T_s + t_D) \)

and \( w \) is a weighting coefficient that can be both positive and negative. Obviously, for each heuristic, \( h_1 \) and \( h_2 \) are different, so the total number of heuristics that can be created combining all the terms above with each other is 15. By sweeping across values of \( w \) in equations 1 and 2 the best effort configuration can be found for each heuristic.
Flat rate: € prop. satisfiability

<table>
<thead>
<tr>
<th>heuristic</th>
<th>SLA%</th>
<th>CPU%</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min(T_F + w N_{CPU})$</td>
<td>97.0%</td>
<td>84.8%</td>
<td>0.48</td>
</tr>
<tr>
<td>$\min(T_F + w A)$</td>
<td>96.8%</td>
<td>82.8%</td>
<td>0.06</td>
</tr>
<tr>
<td>$\min(T_S + w A)$</td>
<td>96.1%</td>
<td>83.0%</td>
<td>0.28</td>
</tr>
<tr>
<td>$\min(T_F + w t_D)$</td>
<td>95.4%</td>
<td>88.0%</td>
<td>-0.025</td>
</tr>
<tr>
<td>$\min(T_S + w t_D)$</td>
<td>95.4%</td>
<td>87.0%</td>
<td>10.0</td>
</tr>
<tr>
<td>$\min(T_F + w t_L)$</td>
<td>95.3%</td>
<td>86.3%</td>
<td>0.015</td>
</tr>
<tr>
<td>$\min(T_S + w t_L)$</td>
<td>95.3%</td>
<td>86.0%</td>
<td>1.15</td>
</tr>
<tr>
<td>$\min(T_F + w T_S)$</td>
<td>95.3%</td>
<td>85.5%</td>
<td>-0.05</td>
</tr>
<tr>
<td>$\min(T_S + w N_{CPU})$</td>
<td>92.7%</td>
<td>77.5%</td>
<td>4.5</td>
</tr>
</tbody>
</table>
## Pay as you go: € prop. CPU utilization

<table>
<thead>
<tr>
<th>heuristic</th>
<th>SLA%</th>
<th>CPU%</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min (T_F + wt_D)$</td>
<td>92.0%</td>
<td>94.0%</td>
<td>-6.63</td>
</tr>
<tr>
<td>$\min (T_S + wt_L)$</td>
<td>92.2%</td>
<td>93.9%</td>
<td>0.4</td>
</tr>
<tr>
<td>$\min (T_F + wT_S)$</td>
<td>91.9%</td>
<td>93.9%</td>
<td>2.0</td>
</tr>
<tr>
<td>$\min (T_F + wt_L)$</td>
<td>92.1%</td>
<td>93.7%</td>
<td>-0.7</td>
</tr>
<tr>
<td>$\min (T_S + wt_D)$</td>
<td>91.7%</td>
<td>93.7%</td>
<td>3.2</td>
</tr>
<tr>
<td>$\min (T_S + wN_{CPU})$</td>
<td>85.0%</td>
<td>93.3%</td>
<td>-0.63</td>
</tr>
<tr>
<td>$\min (T_F + wA)$</td>
<td>89.7%</td>
<td>93.2%</td>
<td>-0.3</td>
</tr>
<tr>
<td>$\min (T_S + wA)$</td>
<td>89.0%</td>
<td>92.8%</td>
<td>0.0</td>
</tr>
<tr>
<td>$\min (T_F + wN_{CPU})$</td>
<td>90.5%</td>
<td>89.3%</td>
<td>-3.0</td>
</tr>
</tbody>
</table>
SNAP e SLA
Grid Scheduling con Service Level Agreement

- Nell’ambito del Service Negotiation and Access Protocol vengono definiti i Service Level Agreement (SNAP/SLA)
- In questo contesto i meccanismi di scheduling locale, distribuito e di gestione di workflow possono essere visti come parti di un sistema “coesivo” ma “aperto” in cui le nuove strategie di scheduling possono evolvere senza sconvolgere l’infrastruttura.
- Si tratta di un’ architettura che rimane neutra, e cerca di mediare tra le politiche potenzialmente in conflitto di risorse, e comunità di utenti
SLA - Generale

- Coordinamento dell’accesso alle risorse in ambito Grid è complicato dal fatto che esistono conflitti tra consumatori e detentori
- Applicazione → assicurazione sul tipo ed il livello del servizio; comprensione e condizionamento del comportamento delle risorse senza doverne conoscere i dettagli
- Gestore risorse → controllo locale e possibilità di uso delle risorse
- Soluzione: stabilire una procedura attraverso la quale le parti negoziano un “service level agreement” (SLA) che esprime un contratto di un service provider con un cliente per fornire qualche “measurable capability” o eseguire uno specifico task
SLA generale

- Globus Resource Manager Architecture, ed in particolare GRAM – 2 è basata sull’idea di negoziazione di uno SLA tra client e provider
- Resource provider può essere collegato ad una risorsa, oppure virtualizzare più risorse (broker, superscheduler)
- SLA permette di costruire una gerarchia di entità di negoziazione che agiscono come provider e client.
Service Negotiation and Access Protocol - SNAP

- Un’ architettura astratta che definisce le operazioni per stabilire e manipolare 3 tipi distinti di SLA:

- Resource Service Level Agreement (RSLA) rappresenta un impegno a fornire una risorsa quando richiesta in un successivo SLA

- Task Service level Agreement (TSLA) rappresenta un impegno ad eseguire un’attività o task con le sue richieste di risorse

- Binding Service Level Agreement (BSLA) rappresenta un impegno ad applicare una risorsa ad un task esistente (ora o in futuro), cioè ad estendere i “requirements” di un task dopo la sua sottomissione o durante l’esecuzione
Scenari di applicazione

- Il modello SNAP/SLA è adatto a diversi tipi di applicazione che sono gestite attraverso l’aggregazione di SLA di base

- Modello interattivo di gestione delle risorse

- Community Scheduler (meta scheduler)

- File transfer (co allocatore)

- Job staging con transfer service e requisiti di sicurezza
Incremental Negotiation

- RSLA: reserve resources for future use
  - TSLA: submit task to scheduler
  - BSLA: bind reservation to task
- Resources change state due to SLAs and scheduler decisions
Community Scheduler

• Un community scheduler è un’entità che agisce come un intermediario “policy enforcing” tra un insieme di risorse e una comunità di utenti

• Lo scheduler quindi lavora per schedulare le attività sulle risorse in modo da realizzare le politiche della comunità riguardo l’uso delle risorse: ad esempio ottimizzare response time oppure il throughput

• Two tiers SLA negotiation process: dal cliente allo scheduler e dallo scheduler al cliente

• L’aderenza al modello multi-tiers per la negoziazione delle SLA condiziona la predicibilità e l’efficienza del processo di scheduling
Community Scheduling Example

- Individual users
  - Require service
  - Have application goals
- Community schedulers
  - Broker service
  - Aggregate scheduling
- Individual resources
  - Provide service
  - Have policy autonomy
  - Serve above clients
Community Scheduler

- Più risorse (R1-R6) ciascuna con un interfaccia RSLA e TSLA
- Lo scheduler negozia inizialmente “capacity guarantees” (via RSLA) con le sue risorse
- Con queste “capacity guarantees” disponibili può negoziare RSLA o TSLA con i suoi clienti per mappare le attività utente alle capacity negoziate
- Il processo di negoziazione prosegue e si adatta alle richieste della comunità ed al workload; in ogni caso la capacità di negoziare accordi con le risorse maschera gli effetti di altri community scheduler o di workload non Grid. *Questo nell’ipotesi che il resource manager forzi il rispetto degli SLA da parte delle risorse locali*
Community Scheduler

- I Community Scheduler (S1, S2) hanno un’interfaccia TSLA verso l’utente
- Gli utenti interagiscono con il community o con il resource level scheduler in modo adeguato rispetto ai loro obiettivi ed ai loro privilegi
- Il cliente “privilegiato” con il job j7 non ha bisogno di una RSLA né dell’aiuto del community scheduler perché le sue esigenze sono espresse direttamente nel TSLA con la risorsa R6
- Il cliente con il job j1 acquisisce una RSLA da R2 come anticipazione della sua richiesta ed utilizza questa prenotazione in un TSLA
Community Scheduler

- I job da j2 a j6 sono sottomessi ai Community Scheduler S1 ed S2 che possono usare privilegi speciali e/o conoscenze domain specific per implementare in modo efficiente i loro jobs sulle risorse condivise.

- Notare che R2 sta eseguendo J1 mentre garantisce una disponibilità futura ad S1, che a sua volta garantisce a J2 uno slot per l’esecuzione basato su questa disponibilità.

- Allo stesso modo R4 esegue J4 e garantisce uno slot futuro attraverso S2.

- Lo scheduler S1 ha un RSLA speculativo con R1 per servire più rapidamente un high priority job in futuro.
File transfer in ambiente distribuito
File Transfer

- Trasferimento di un file da uno storage system ad un altro con deadline. Richiede più risorse:
  - Spazio di memoria sul sito di destinazione
  - Network bandwidth durante il trasferimento
  - I/O bandwidth sul nodo di destinazione durante il trasferimento
  ⇒ Gestione di più risorse con co-scheduling

- Il file transfer scheduler S1 ha un’interfaccia TSLA, lo storage system ha interfacce TSLA/RSLA ed il network manager R2 ha interfacce RSLA/BSLA

- L’utente sottomette un “transfer job” J1 allo scheduler e negozia un TSLA che include una deadline
File transfer

- Il file transfer scheduler lavora per soddisfare la deadline:
- Ottiene uno storage reservation sulla destinazione R3 per assicurare la disponibilità dello “spazio disco”
- Ottiene “bandwidth reservation” dalla rete e dallo storage device per poter garantire il rispetto della deadline
- Sottomette i job J2 e J3 che implementano il trasferimento dati tra sorgente e destinazione usando la banda di I/O e lo spazio disco prenotati
- Stabilisce una BSLA con R2 per utilizzare la banda di rete prenotata per i socket creati tra J2 e J3
Utilizzo di SLA collegate per casi più complessi

- Dependent SLAs nest intrinsically
  - BSLA2 defined in terms of RSLA2 and TSLA4
- Chained SLAs simplify negotiation
  - Optionally link destruction/reclamation
Job staging con file transfer

- Situazione di co-allocazione più complessa, basata su tre attività:
  - Trasferimento dati da uno storage system ad una locazione intermedia
  - Calcolo sulla locazione intermedia usando i dati
  - Trasferimento risultati ad una locazione finale
- Caratteristiche: risorse distribuite e comunità di utenti
- Richieste specifiche: sicurezza dei dati → utilizzo di un account temporaneo protetto sulla locazione intermedia
- SLA → facilitano la scomposizione di attività necessarie e la gestione dinamica dell’account
Job Staging

- TSLA1 è il risultato della negoziazione con la risorsa per stabilire un account temporaneo per il cliente.
- Tutti i job eseguiti e le SLA utilizzate dal cliente sulla risorsa sono collegati (linked) a questa TSLA di lunga durata
- Tutte le SLA collegate devono essere distrutte ed i job terminati per poter chiudere in modo “safe” l’account
- RSLA1 promette al cliente 50 GB di memoria in un particolare filesystem della risorsa
- BSLA1 lega 30 GB della memoria promessa ad un particolare insieme di file
- TSLA2 esegue un job complesso che genera (spawn) due sotto job per la gestione dell’I/O
Job Staging

- TSLA3 è il TSLA negoziata per il primo task di file transfer che gestisce l’input. Non è richiesta una garanzia addizionale di qualità del servizio.
- TSLA4 è il TSLA negoziato per il secondo di file transfer che gestisce l’output con una deadline, prima che lo spazio sul file system locale vada perso.
- RSLA2 e BSLA2 sono usate dal file transfer (output) per ottenere la banda necessaria a completare il trasferimento prima della deadline.
Job Staging

- La “logica” (il codice) per avviare i job di staging e negoziare TSLA3 e TSLA4 può essere contenuta nel job per il quale è negoziato TSLA2, oppure nell’entità che negozia TSLA2 a nome del cliente. In Figura il nesting delle SLA è mirato ad evidenziare come il tempo di vita di queste astrazioni di gestione sia collegato nella pratica.
- Il collegamento può essere forzato attraverso una dipendenza tra i soggetti delle SLA (ad esempio BSLA2 è privo di significato fuori dal tempo di vita di TSLA4 e RSLA2)
- In alternativa può essere inserita come operazione di gestione (ad esempio attivando la distruzione ricorsiva di tutti gli SLA dalla radice in modo da velocizzare il recupero delle risorse collegate all’applicazione
Virtualizzazione delle risorse ed intermediari

- Il community scheduler considerato virtualizza le risorse a beneficio dei partecipanti alla comunità → questo tipo di virtualizzazione è utile per aumentare il livello di fiducia necessaria in un’applicazione Grid.

- Gli utenti della comunità hanno fiducia nel loro scheduler per la definizione degli accordi per ottenere risorse ed eseguire i task; a sua volta lo scheduler ha un suo “trust model” per determinare quali risorse sono un obiettivo ragionevole per il workload generato dalla comunità.

- La virtualizzazione permette di avere un approccio “application oriented” evitando i dettagli di gestione della risorsa

- La presenza di intermediari porta ad astrazioni ed anche al collegamento di politiche di utilizzo attraverso domini diversi
Virtualizzazione delle risorse ed intermediari

*Figure 8.4.* SLA negotiation with intermediaries. A negotiation pipeline between a user, community scheduler, and resource manager permits policies to be introduced at each stage which affect the outcome and are illustrated using color mixing. User policy affects what requests are initiated, community policy affects how user requests are mapped to resource-level requests, and resource policy affects how resources may be utilized. Thus policy from each source mixes into the stream of requests going to the right, and into the streams of advertisements and request-responses going to the left.
Comprensione degli SLA

• **L’automazione** degli SLA richiede di essere in grado di esprimere in modo trattabile da un programma la negoziazione ed in particolare che cosa è richiesto e cosa è offerto.

• Questo pone diversi problemi:
  – Tutte e tre gli SLA riguardano “che cosa deve essere fatto dalle risorse” → come le distingo tra loro in modo più preciso?
  – Come le rappresento?
  – Cosa vuol dire per uno scheduler stabilire un accordo o una promessa di fare qualcosa?
Comprensione degli SLA

• Diversi tipi di accordo - nel quadro considerato ci sono tre tipi di promesse che uno scheduler può fare sulla disponibilità futura di risorse e definire attraverso gli SLA:
  – Una risorsa sarà consumata in futuro;
  – Una risorsa sarà consumata in un certo modo;
  – Un task sarà eseguito con le risorse di cui ha bisogno

• RSLA
  – promette la disponibilità di una risorsa in futuro senza associarla ad un piano di utilizzazione
  – Corrisponde ad una (advanced o immediate) reservation quindi potrà essere usata nella negoziazione di future SLA
  – non ha effetto fino a quando non è richiesta in una successiva BSLA o TSLA
  – è particolarmente utile per la co-allocazione o sincronizzazione
Comprensione degli SLA

- BSLA: collega una risorsa ad un task, indipendentemente dalla creazione del task, fornisce risorse al task. Non implica l’attivazione del task. Il task può essere già attivo o sarà attivato in futuro.

- Questo disaccoppiamento tra creazione del task e fornitura delle risorse può essere usato per la fornitura di risorse di risorse a task creati al di fuori del meccanismo di negoziazione SLA, ad esempio un socket, o un processo Unix creati in modo interattivo.

- Un altro esempio è un task che richiede un insieme di risorse fisse (negoziate attraverso un TSLA di durata pari a quella del task) e un insieme di risorse variabili, negoziate attraverso BSLA di durata più breve.
Comprensione degli SLA

• TSLA: rappresenta l’impegno da parte di un resource manager ad eseguire un task con il livello di risorse richiesto

La figura riassume il tipo di promessa degli SLA. Notare che tutti e tre promettono la disponibilità delle risorse
Lo SNAP Agreement Protocol

- Il nucleo dell’architettura di gestione delle risorse è rappresentato da un’interazione tipo client-server utilizzata per negoziare gli SLA. Ogni operazione può essere rappresentata da un messaggio dal cliente al servizio o viceversa.

- Tutte queste operazioni seguono un pattern tipo RPC: la richiesta di un servizio è la chiamata della RPC mentre il messaggio di ritorno restituisce il valore della chiamata (Il sottostante livello di trasporto mette in correlazione chiamate e messaggi di ritorno)

- Questa interpretazione è consistente con la modalità di deployment del protocollo in unambiente Web services, usando WSDL per modellare i messaggi RPC
Transizioni di stato e accordi

- Considerando le dipendenze di un BSLA da un RSLA (ed eventualmente da un TSLA) esistono 4 stati attraverso i quali può avanzare il processo di planning per una risorsa:

  - S0: Non esistono SLA sulla risorsa (non sono state create o sono state rilasciate)

  - S1: è stato raggiunto un accordo su un RSLA e un TSLA ma non è stato fatto un collegamento tra i due

  - S2: TSLA è collegato al RSLA e questo collegamento è un BSLA per “risolvere” il task

  - S3: Le risorse sono utilizzate per il task e possono essere ancora controllate e/o modificate
Lo SNAP Agreement Protocol

- Il nucleo dell’architettura di gestione delle risorse è rappresentato da un’interazione tipo client-server utilizzata per negoziare gli SLA. Ogni operazione può essere rappresentata da un messaggio dal cliente al servizio o viceversa.

- Tutte queste operazioni seguono un pattern tipo RPC: la richiesta di un servizio è la chiamata della RPC mentre il messaggio di ritorno restituisce il valore della chiamata (Il sottostante livello di trasporto mette in correlazione chiamate e messaggi di ritorno).

- Questa interpretazione è consistente con la modalità di deployment del protocollo in un ambiente Web services, usando WSDL per modellare i messaggi RPC.
Rappresentazione di SLA

- Integrazione con Web e Grid service porta all’utilizzo di XML → si tratta di una rappresentazione abbastanza pesante che di solito deve sottostare a delle regole stabilite da uno standard body, per permettere un’efficace interoperabilità.

- Vediamo qual è il contenuto di SLA trattabili da processi standard; gli elementi di base sono costituiti da:

  1. SLA references che permettono ad una SLA appena negoziata di essere associata con SLA pre esistenti;
  2. Resource description: che è il soggetto principale di una RSLA e può apparire in una TSLA o BSLA (potenzialmente accompagnata da RSLA references)
  3. Resource metadata che qualificano le capability con info quali tempo di disponibilità, affidabilità etc…
  4. Task description. Che è il soggetto principale di una TSLA e può apparire in una BSLA
  5. SLA metadata: che qualificano il contenuto della SLA con il livello di commitment, le politiche di revoca, il tempo di vita…
Rappresentazione di SLA

- Il contenuto di una RSLA include resource description e metadata, e gli SLA metadata.
- Il contenuto di una BSLA comprende, riferimenti a TSLA o descrizione di task (chi consuma le risorse), descrizione di risorse (quelle che saranno fornite al task), e riferimenti a RSLA (identificano risorse promesse e da utilizzare) e metadati su SLA.
- Il contenuto di una TSLA comprende un task description (quello da eseguire), descrizione di risorse e opzionale un RSLA ref. (le risorse da usare requisiti del task) e SLA metadata.
SLA nello SNAP protocol

• SNAP protocol mantiene un insieme di SLA (lato manager) usando messaggi inviati dal client

• Tutte le SLA contengono un identificatore \( l \) di SLA, un cliente \( c \) con il quale la SLA è fatta, una data di scadenza \( T_{\text{dead}} \), una descrizione specifica \( a \) di RSLA, BSLA, TSLA:

\[
\langle l, c, T_{\text{dead}}, a \rangle
\]

• Ogni tipo di SLA ha il suo tipo di descrizione del contenuto (ad esempio resource requirements oppure task description)

• Esiste un linguaggio estensibile J per descrivere task (job), un linguaggio sottoinsieme R contenuto in J in grado di esprimere i resource requirements in J, o res. Req. stand alone.

• Intendiamo la relazione \( a' \) contenuto-uguale \( a \) come il fatto che \( a' \) describe gli stessi termini dell’accordo \( a \), eventualmente con qualche restrizione in più. Quindi \( a' \) soddisfatto => \( a \) soddisfatto
SLA e SNAP

- **RSLA:**
  \[<l, c, t\text{-}dead, <r>R>\]
  dove \(r\) è la descrizione della risorsa nel linguaggio \(R\) sottoinsieme di \(J\);

- **TSLA:**
  \[<l, c, t\text{-}dead, <j>T>\]
  \(j\) è il task a cui garantire le risorse, la descrizione di \(j\) include la descrizione delle risorse \(r=j|_R\)

- **BSLA:**
  \[<l, c, t\text{-}dead, <j>B>\]
  anche in questo caso \(j\) è il task e la descrizione di \(j\) comprende le risorse richieste
Resource e task meta language

Figure 7. Hypothetical resource description. A parallel computer with 128 dedicated dual-processor nodes, each providing at least 256 MB of memory and 1 GB disk with disk performance of 30 MB/s, connected by Myrinet-enabled MPI. A parse tree is provided to help illustrate the nested expression.
Examples
Automatic verification of SLA for Firewall Configuration in Grid Environments
Motivation

*Unofficial:* probably the easiest case of SLA 😊

*Official:* Facilitate the integration of new resources into a Grid:

1. Definition of security profiles
2. Certification of firewall setup
3. Monitoring firewall configuration as part of the Service Level Agreements
Summary

1. Firewall configuration issues
2. Classification of middleware components
3. SLA extension
4. Tool for automatic verification of firewall configuration
Integration of new partners

- Installation of Grid middleware(s)
- Creation of local user accounts
- Registration to the information services
- ...
- ...
- **Configuration of firewall rules**
  - If too restrictive → prevent legitimate communications
  - If too loose → allow unauthorized communications
Classification of middleware components

Four categories of middleware components:

1. Computing frontends
2. Data frontends
3. Interactive nodes
4. Worker nodes

Different security profiles!
Communication paths

- Identification of network ports used by each component for incoming connections

<table>
<thead>
<tr>
<th>Component</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT 4.0 GRAM</td>
<td>2811, 8443</td>
</tr>
<tr>
<td></td>
<td>20000-25000</td>
</tr>
<tr>
<td>OGSA-DAI</td>
<td>8443</td>
</tr>
<tr>
<td>dCache SE</td>
<td>2135, 2811, 8443, 20000-25000</td>
</tr>
</tbody>
</table>
SLA extension

• Each site declares which security profile will be implemented
• Provide guarantee that communications to/from certain Grid services is allowed, i.e. firewall is correctly configured
• Verification:
  • before accepting a site in production
  • periodically for all the duration of the collaboration
Verification of firewall configuration

Central service performing periodic verifications:
- requested ports are accessible
- all other ports are blocked

In a further evolution
- allow peer-to-peer verification of selected sites
- triggered on-demand
## Firewall Verification Tool

Last check performed on Thursday 09.10.2008

<table>
<thead>
<tr>
<th>Site</th>
<th>Host</th>
<th>Type</th>
<th>Port</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leibniz Universität Hannover</td>
<td>gramd1.d-grid.uni-hannover.de</td>
<td>Globus GRAM</td>
<td>2811, 8443, 20000-25000</td>
<td>OK, OK, OK</td>
</tr>
<tr>
<td></td>
<td>logced1.d-grid.uni-hannover.de</td>
<td>LCG/gLite CE</td>
<td>2119, 2170, 2811, 20000-25000</td>
<td>OK, OK, OK</td>
</tr>
<tr>
<td></td>
<td>unicore1.d-grid.uni-hannover.de</td>
<td>UNICORE NJS</td>
<td>1128</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>dcached1.d-grid.uni-hannover.de</td>
<td>SRMldCache</td>
<td>2135, 2811, 8443, 20000-25000</td>
<td>Filtered</td>
</tr>
<tr>
<td>Technische Universität Dortmund</td>
<td>udo-gt01.grid.uni-dortmund.de</td>
<td>Globus GRAM</td>
<td>2811, 8443, 20000-25000</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>udo-ce01.grid.uni-dortmund.de</td>
<td>LCG/gLite CE</td>
<td>2119, 2170, 2811, 20000-25000</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>udo-njs01.grid.uni-dortmund.de</td>
<td>UNICORE NJS</td>
<td>1128</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>udo-dcache01.grid.uni-dortmund.de</td>
<td>SRMldCache</td>
<td>2135, 2811, 8443, 20000-25000</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>udo-ogsa01.grid.uni-dortmund.de</td>
<td>OGSA-DAI</td>
<td>8443</td>
<td>OK</td>
</tr>
</tbody>
</table>
Grid vs Cloud

Figure 2: Grid Protocol Architecture

Figure 3: Cloud Architecture
What about the future?

What does the future hold? We will hazard a few predictions, based on our beliefs that the economics of computing will look more and more like those of energy. Neither the energy nor the computing grids of tomorrow will look like yesterday’s electric power grid. Both will move towards a mix of micro-production and large utilities, with increasing numbers of small-scale producers (wind, solar, biomass, etc., for energy; for computing, local clusters and embedded processors—in shoes and walls) co-existing with large-scale regional producers, and load being distributed among them dynamically. Yes, computing isn’t really like electricity, but we do believe that we will nevertheless see parallel evolution, driven by similar forces.
<table>
<thead>
<tr>
<th>Features</th>
<th>Amazon</th>
<th>Flexiscale</th>
<th>Elastic Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computing</strong></td>
<td>(1.75GB RAM)</td>
<td>(2 GB RAM)</td>
<td>(2 GB RAM)</td>
</tr>
<tr>
<td></td>
<td>$0.11 per hour (Linux) $0.135 per hour (Windows)</td>
<td>£0.20 per hour (Linux) £0.23 per hour (Windows)</td>
<td>£0.065/hour (Linux)</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>$0.18 per GB-month</td>
<td>£0.30/GB-month</td>
<td>£0.06/GB-month</td>
</tr>
<tr>
<td><strong>Uploads</strong></td>
<td>$0.10/GB</td>
<td>£0.07/GB</td>
<td>£0.10/GB committed</td>
</tr>
<tr>
<td></td>
<td>Free bandwidth between EC2 and S3</td>
<td></td>
<td>£0.20/GB burst above committed</td>
</tr>
<tr>
<td><strong>Downloads</strong></td>
<td>$0.17/GB</td>
<td>£0.10 &lt; 1TB/month</td>
<td>£0.10/GB committed</td>
</tr>
<tr>
<td></td>
<td>Free bandwidth between EC2 and S3</td>
<td></td>
<td>£0.20/GB burst above committed</td>
</tr>
<tr>
<td><strong>Firewalling</strong></td>
<td>-------------------------------------------</td>
<td>£0.01 / hour per IP</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td><strong>Load Balancing</strong></td>
<td>-------------------------------------------</td>
<td>£0.02 / hour per IP</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td><strong>Extra Features</strong></td>
<td>SOAP and ReST and other API support, SQS service</td>
<td>Control Panel, Monitoring &amp; Recovery, API support</td>
<td>----------------------------------------</td>
</tr>
</tbody>
</table>
Effective Date: October 1, 2007
This Amazon S3 Service Level Agreement ("SLA") is a policy governing the use of the Amazon Simple Storage Service ("Amazon S3") under the terms of the Amazon Web Services Customer Agreement (the "AWS Agreement") between Amazon Web Services, LLC ("AWS", "us" or "we") and users of AWS’ services ("you"). This SLA applies separately to each account using Amazon S3. Unless otherwise provided herein, this SLA is subject to the terms of the AWS Agreement and capitalized terms will have the meaning specified in the AWS Agreement. We reserve the right to change the terms of this SLA in accordance with the AWS Agreement.

Service Commitment
AWS will use commercially reasonable efforts to make Amazon S3 available with a Monthly Uptime Percentage (defined below) of at least 99.9% during any monthly billing cycle (the "Service Commitment"). In the event Amazon S3 does not meet the Service Commitment, you will be eligible to receive a Service Credit as described below.
Definitions

“Error Rate” means: (i) the total number of internal server errors returned by Amazon S3 as error status “InternalError” or “ServiceUnavailable” divided by (ii) the total number of requests during that five minute period. We will calculate the Error Rate for each Amazon S3 account as a percentage for each five minute period in the monthly billing cycle. The calculation of the number of internal server errors will not include errors that arise directly or indirectly as a result of any of the Amazon S3 SLA Exclusions (as defined below).

- “Monthly Uptime Percentage” is calculated by subtracting from 100% the average of the Error Rates from each five minute period in the monthly billing cycle.

- A “Service Credit” is a dollar credit, calculated as set forth below, that we may credit back to an eligible Amazon S3 account
• **Service Credits**

Service Credits are calculated as a percentage of the total charges paid by you for Amazon S3 for the billing cycle in which the error occurred in accordance with the schedule below.

<table>
<thead>
<tr>
<th>Monthly Uptime Percentage</th>
<th>Service Credit Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal to or greater than 99% but less than 99.9%</td>
<td>10%</td>
</tr>
<tr>
<td>less than 99%</td>
<td>25%</td>
</tr>
</tbody>
</table>
• **Effective Date: October 23, 2008**
This Amazon EC2 Service Level Agreement ("SLA") is a policy governing the use of the Amazon Elastic Compute Cloud ("Amazon EC2") under the terms of the Amazon Web Services Customer Agreement (the "AWS Agreement") between Amazon Web Services, LLC … bla bla bla.

• **Service Commitment**
AWS will use commercially reasonable efforts to make Amazon EC2 available with an Annual Uptime Percentage (defined below) of at least **99.95%** during the Service Year. In the event Amazon EC2 does not meet the Annual Uptime Percentage commitment, you will be eligible to receive a Service Credit as described below.

• **Downtime: 4,38 minuti in un anno**
• Definitions
  – “Service Year” is the preceding 365 days from the date of an SLA claim.
  – “Annual Uptime Percentage” is calculated by subtracting from 100% the percentage of 5 minute periods during the Service Year in which Amazon EC2 was in the state of “Region Unavailable.” If you have been using Amazon EC2 for less than 365 days, your Service Year is still the preceding 365 days but any days prior to your use of the service will be deemed to have had 100% Region Availability. Any downtime occurring prior to a successful Service Credit claim cannot be used for future claims. Annual Uptime Percentage measurements exclude downtime resulting directly or indirectly from any Amazon EC2 SLA Exclusion (defined below).
  – “Region Unavailable” and “Region Unavailability” means that more than one Availability Zone in which you are running an instance, within the same Region, is “Unavailable” to you.
  – “Unavailable” means that all of your running instances have no external connectivity during a five minute period and you are unable to launch replacement instances.
  – The “Eligible Credit Period” is a single month, and refers to the monthly billing cycle in which the most recent Region Unavailable event included in the SLA claim occurred.
  – A “Service Credit” is a dollar credit, calculated as set forth below, that we may credit back to an eligible Amazon EC2 account.
Google Apps SLA

• During the Term of the applicable Google Apps Agreement, the Google Apps Covered Services web interface will be operational and available to Customer at least 99.9% of the time in any calendar month (the "Google Apps SLA"). If Google does not meet the Google Apps SLA, and if Customer meets its obligations under this Google Apps SLA, Customer will be eligible to receive the Service Credits described below. This Google Apps SLA states Customer's sole and exclusive remedy for any failure by Google to provide the Service.

• Definitions. The following definitions shall apply to the Google Apps SLA.
  – "Downtime" means, for a domain, if there is more than a five percent user error rate. Downtime is measured based on server side error rate.
  – "Downtime Period" means, for a domain, a period of ten consecutive minutes of Downtime. Intermittent Downtime for a period of less than ten minutes will not be counted towards any Downtime Periods.
  – "Google Apps Covered Services" means the GMail, Google Calendar, Google Talk, Google Docs, and Google Sites components of the Service. This does not include the GMail Labs functionality or Gmail Voice and Video Chat components of the Service.
  – "Monthly Uptime Percentage" means total number of minutes in a calendar month minus the number of minutes of Downtime suffered from all Downtime Periods in a calendar month, divided by the total number of minutes in a calendar month.
  – "Scheduled Downtime" means those times where Google notifies Customer of periods of Downtime at least five days prior to the commencement of such Downtime. There will be no more than twelve hours of Scheduled Downtime per calendar year. Scheduled Downtime is not considered Downtime for purposes of this Google Apps SLA, and will not be counted towards any Downtime Periods.
  – "Service" means the service provided by Google to Customer under the applicable Google Apps Agreement.
Credits when SLA fails

• "Service Credit" means the following:

<table>
<thead>
<tr>
<th>Monthly uptime %</th>
<th>Days of Service added to the end of the Service term, at no charge to Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 99.9 ≥ 99.0</td>
<td>3</td>
</tr>
<tr>
<td>&lt; 99.0 ≥ 95.0</td>
<td>7</td>
</tr>
<tr>
<td>&lt; 95.0</td>
<td>15</td>
</tr>
</tbody>
</table>
References


- 21 experts define Cloud: [http://weblogic.sys-con.com/node/612375](http://weblogic.sys-con.com/node/612375)