

JSDL EXTENSIONS

UPGRADING JSDL TO SUPPORT NEW CONCEPTS FROM XTREEMOS

Matej Artac (XLAB)

Massimo Coppola (CNR)

Toni Cortes (BSC)

Yvon Jegou (INRIA)

John Mehnert-Spahn (UDUS)

Ramón Nou (BSC)

Tabla de contenido

| | |
|-----------------------------------|----------|
| Index | 2 |
| Security | 3 |
| SSO/delegation (Yvon)..... | 3 |
| Resource selection..... | 3 |
| Tolerance (Massimo) | 3 |
| Scheduling Hints | 5 |
| File usage (Toni and Ramón) | 5 |
| Job execution | 7 |
| Job time (Matej) | 7 |
| Firewalls (MaTej) | 10 |
| Interactive jobs (Yvon) | 11 |
| Checkpointing (JOHN, MaTej) | 12 |

SECURITY

SSO/DELEGATION (YVON)

MOTIVATION

When a user submits a request to the Grid from his desktop, the Grid access point locates the user credentials/certificates, and checks that they are valid and that the user is allowed to use them through some cryptographic means (keyword challenge). In order to apply this procedure from a node running a user application on the Grid, the user certificate as well as the corresponding private key must be present on this node. Moreover, if this private key is protected by a password, some means must be provided to implement the private key challenge process.

Our proposition for single-sign-on is based on an xtreemos service, the user session management service, which forwards grid requests from the user to XtreamOS services. In order to provide delegation and to allow users to run grid requests from jobs, such services must be started on the resource(s) allocated to the job. For this reason, we plan to extend the jsdl. The execution manager is then responsible for starting the service if it is requested in the jsdl.

USECASE EXAMPLE

A user submits a job which needs to interact with the grid: requesting more resources, mounting a user volume...

SYNTAX

Extend JobDescription with:

```
<delegation>true/false</delegation>
```

IMPACT OUT OF XTREEMOS

This option does not depend on the specific implementation of delegation (XtreamOS user session management service, Globus certificate proxy) and might be of interest outside of XtreamOS as it allows delegating credentials only when necessary.

RESOURCE SELECTION

TOLERANCE (MASSIMO)

MOTIVATION

We are using a JSDL extension (which should be compatible and ignored by standard JSDL parsers) in order to specify the tolerance allowed when matching desired values for dynamic attributes with respect to their static value.

USECASE EXAMPLE

When searching for a set of machines to execute a job, XtreamOS make use of information about dynamic attributes of machine.

For instance consider a resource query with an Exact query constraint of 4GB ram and tolerance 0.4. A machine with at least 4GB and at least 1.6GB ($1.6\text{GB} = 4\text{GB} * (0.4)$) free memory is accepted while a machine with 4GB but only 1.5GB free is rejected (as $1.5\text{GB} < 1.6\text{GB}$).

The exploitation of dynamic values inside SRDS is done with two modifications to the standard JSDL.

The first one regards the already existing tag Resource that is extended with a new attribute to indicate the tolerance. The Resource elements affected are

1. IndividualPhysicalMemory
2. IndividualVirtualMemory
3. IndividualDiskSpace
4. IndividualNetworkBandwidth

The **tolerance** attribute is applied to the already existing tags Lower-Bound, UpperBound, Exact, LowerBoundedRange and UpperBoundedRange.

The threshold value for a query about a dynamic value is computed multiplying the corresponding static bound, listed previously, by the tolerance in order to get the dynamic value range.

A node is discarded when the dynamic value for a resource is not in the threshold range computed using the tolerance. In case of exact values a node is discarded if its dynamic value stays beyond the threshold.

Currently this solution is implemented on the latest version of XtreamOS (2.0) with a different syntax (dynamic-epsilon in place of tolerance) and with a semantic limited to the LowerBound elements of a Range constraint.

The second extension to the JSDL aim to add the tags:

1. Uptime
2. IdlePercentage

to the JSDL. They refers to inherently dynamic values, thus missing in the JSDL standard, and they are relevant in the XtreamOS settings, where we want to be able to deploy processes on machine only partially loaded.

Both of these attributes are of Resources type. For the sake of uniform syntax, we keep the tolerance attribute also in the IdlePercentage and Uptime, even if they miss a static counterpart.

SYNTAX

A dynamic request with tolerance constraint on the lower and the upper bounds of a Resource.

```

<jSDL-srds:IndividualDiskSpace>
< jSDL-srds:Range>
    <jSDL-srds:LowerBound tolerance = " 0.5 " > 30000
    </jSDL-srds:LowerBound >
    <jSDL-srds:UpperBound tolerance = " 0.2 " > 600000
    </jSDL-srds:UpperBound >
</jSDL-srds:Range>
</jSDL-srds:IndividualDiskSpace>

```

A dynamic request with tolerance constraint on the exact value of a Resource.

```

<jSDL-srds:IndividualDiskSpace>
    <jSDL-srds:Exact tolerance="0.5"> 30000
    </jSDL-srds:Exact>
</jSDL-srds:IndividualDiskSpace>

```

IMPACT OUT OF XTREEMOS

This is clearly usable with any Grid system.

SCHEDULING HINTS

FILE USAGE (TONI AND RAMÓN)

MOTIVATION

It is clear that the performance of some applications may depend on the performance of the access to the files they use. As this is an important issue in the scheduling of XtremOS, we need to find a way to specify the important files used by an application, and the JSDL seems to be the best option.

USECASE EXAMPLE

In the scheduling process, the JobManager will ask for resources close to the important files and thus the application will have a faster access to its important files.

SYNTAX

Namespace : jSDL-sdl

```

<SchedulingHint>
  <FileSystem type="XtremFS">
    <Volume id="Default">
      <File>out.txt</File>
      <File>tmpdir/process.txt</File>
    </Volume>
  </FileSystem>

```

</SchedulingHint>

Volume id values = Default|UUID

This is the XSD, with all SchedulingHint proposals. Scheduling Hints is not an enumeration for now.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema                                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.ggf.org/jsdl/2009/11/jsdl-sdl"                                xmlns:jsdl-
sdl="http://schemas.ggf.org/jsdl/2009/11/jsdl-sdl"
targetNamespace="http://schemas.ggf.org/jsdl/2009/11/jsdl-sdl"
elementFormDefault="qualified">
```

```
<!--=====-->
```

```
<xsd:complexType name="Volume_Type">
```

```
<xsd:sequence>
```

```
<xsd:element name="File" type="xsd:string" minOccurs="1"/>
```

```
</xsd:sequence>
```

```
<xsd:attribute name="id" type="xsd:string" use="required" />
```

```
</xsd:complexType>
```

```
<!--=====-->
```

```
<xsd:complexType name="FileSystem_Type">
```

```
<xsd:sequence>
```

```
<xsd:element name="Volume" type="Volume_Type" minOccurs="1"/>
```

```
</xsd:sequence>
```

```
<xsd:attribute name="type" type="xsd:string" use="required" />
```

```
</xsd:complexType>
```

```
<!--=====-->
```

```
<xsd:element name="SchedulingHint" type="SchedulingHint_Type" />
```

```
<xsd:complexType name="SchedulingHint_Type">
```

```
<xsd:sequence>
```

```
<xsd:element name="FileSystem" type="FileSystem_Type"
minOccurs="0"/>
```

```
<xsd:element name="Scheduler" type="xsd:string" minOccurs="0"/>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<!--=====-->
```

</xsd:schema>

The path should be relative to the root directory of the volume.

IMPACT OUT OF XTREEMOS

Although this is not currently taken into account by current systems, XtremOS may lead the path and others may follow. Thus it makes plenty of sense to extend the JSDL to include this information, even if it is just used as a hint that many systems ignore.

Please not that this not the same as stage in/out as both mechanisms might be in use in a system.

JOB EXECUTION

JOB TIME (MATEJ)

MOTIVATION

Every job's execution has a start time and duration. While the job itself can mostly control its duration, the start time is fully a matter of the infrastructure. The user, however, will want to have some level of control about the temporal aspects of the job's lifetime.

The JSDL specifications [1] assume that the Job Lifetime Management Language would be used as one of the specifications to supplement the JSDL, but according to our knowledge, to date no such specification has been published.

When defining the job's lifetime, we would like to be able to define the following metrics:

- The time at or after which we want the job to start.
- The execution time. This is the duration of the job's execution.

Advanced jobs may have a need to express further constraints and requirements regarding their execution time:

- Days of week when the job can be executed.
- Times of day when the job can be executed.

Outside these constrains the job should not be given any execution time. It should either be stopped when the permitted period is about to end, and restarted at the next interval. The restarting can occur from the scratch, or from a check-pointed state.

Our proposal is partly based on the one found in a PhD thesis by Omar Subhi Aldabbas [2].

USECASE EXAMPLE

We see a twofold benefit of adding the job lifetime requirements to the job specification:

- A hint for the scheduler, helping it with restrictions that limit the possible space for solutions. It also enables an early decision on whether the job will run on a resource given an existing set of reservations.
- Provides the users with a better control about when the job will start and when to expect it to be finished. With additional constraints, it also provides a way for interesting jobs that need to perform computation tied to periodic schedules.

Additional constraints require that the job is handled in a more advanced way, but offers that the jobs can do periodic tasks. We can imagine a job that collects the data sets, created by other jobs, daily for the previous day. These jobs can be scheduled to run after midnight every day for a short period of time. Some users may also find it better to run their jobs during the night times or at the weekends, when the resources are more likely to be idle.

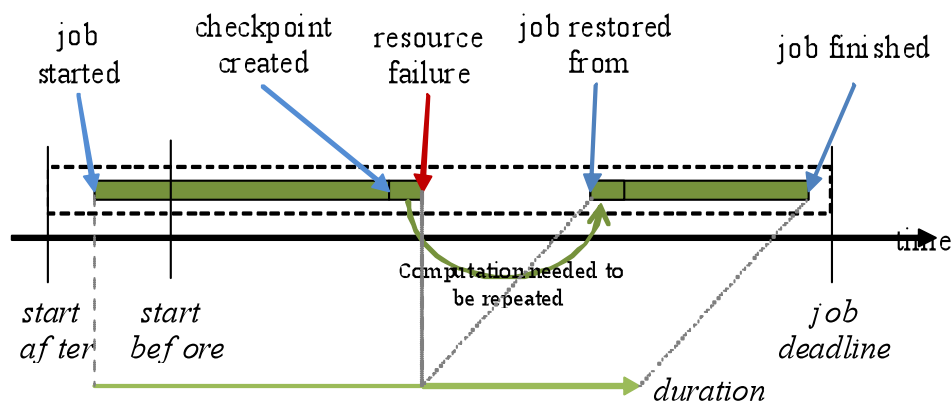


Figure 1: An illustration of the job lifetime variables shown on a job that needed to be restored from an earlier checkpoint. ¡Error!No se encuentra el origen de la referencia.

SYNTAX

```
<LifeTime>
  <StartTime> datetime </StartTime>
  <ExecutionTime> time </ExecutionTime>
  <Constraints>
    <Constraint>
      <DayOfWeek> dayofweek </DayOfWeek>
      <TimeInterval>
        <Start> time </Start>
        <End> time </End>
      </TimeInterval>
    </Constraint>
  </Constraints>
</LifeTime>
```

- **LifeTime** – a complex structure expressing the life time requirements and constraints. Zero or one can occur in a document. If it is missing, the system can select any values.

- **StartTime** – expresses the time at or after which the job should start. Exactly one should occur in the tag.
- **ExecutionTime** – the time in seconds expressing the expected execution time.
- **Constraints** – a complex tag expressing additional temporal constraints. Zero or one occurrence of the tag is possible.
 - **Constraint** – a complex tag expressing additional constraints. They are composed of the days of week when the job is allowed to run, and the time intervals within those days when the job is allowed to run. There should be one or more occurrences of the tag.
 - **DayOfWeek** – a numerical representation (0-6, 0 representing a Monday, and 6 representing a Sunday) of the day of the week when the job is allowed to run. Zero or more occurrences of the tag are expected. Zero occurrences mean the job can run on *any* day of the week. Multiple occurrences mean that the job can run on *each* day of week listed in the tag.
 - **TimeInterval** – composed of **Start** tag and an **End** tag, which represent the start and the end time of the interval, respectively. Within this interval, the job is allowed to run, but only on the days of week specified by the **DayOfWeek** tag on the same level as the **TimeInterval**. Zero or more occurrences of the tag are expected. If zero tags occur, the system assumes that the job can run on the whole day. If multiple tags occur, the job can run on all of the provided intervals.

To clarify the usage of the **Constraints** tag, the following example will create a time-table as shown on the Figure 2

```
<LifeTime>
  <StartTime> 25.09.2009 16:00 </StartTime>
  <ExecutionTime> 200000 </ExecutionTime>
  <Constraints>
    <Constraint>
      <DayOfWeek> 2 </DayOfWeek>
      <DayOfWeek> 3 </DayOfWeek>
      <DayOfWeek> 5 </DayOfWeek>
      <TimeInterval>
        <Start> 4:00:00 </Start>
        <End> 12:00:00 </End>
      </TimeInterval>
    </Constraint>
    <Constraint>
      <DayOfWeek> 4 </DayOfWeek>
      <DayOfWeek> 5 </DayOfWeek>
    </Constraint>
  </Constraints>
  <TimeInterval>
    <Start> 20:00:00 </Start>
    <End> 24:00:00 </End>
  </TimeInterval>
</LifeTime>
```

```

    </Constraints>
  </LifeTime>

```

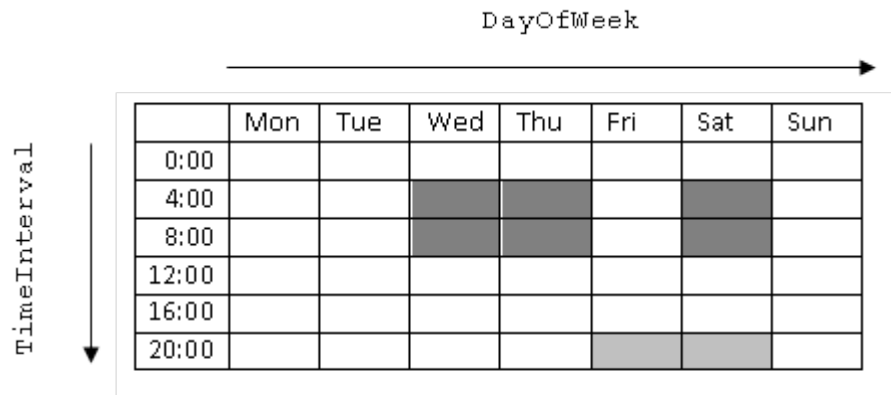


Figure 2: An illustration of the constraints shown in the example. The job can run only on Wednesdays, Thursdays and Saturdays from 4 am to 12pm, and on Fridays and Saturdays from 8pm to 12am.

If the **StartTime** tag expresses the time not within the **Constraints** time specifications, then the job starts at the time representing the start of the next permissible period. Since the job can only run within the constraints, the **ExecutionTime** will be accounted only within the effective duration. For example, a job which starts on Monday which is constrained to run only for 1 hour each day of the week will end on Sunday, if the **ExecutionTime** is set to 7 hours.

Note on existing POSIX extension standard: an existing POSIX application assumes a **POSIXApplication / WallTimeLimit** tag, which is a soft limit on the duration of the application's execution, in seconds. An application will need to use this value, if present, instead of the proposed **ExecutionTime**.

IMPACT OUT OF XTREEMOS

Job lifetime parameters are important for any system capable of creating reservations and scheduling tasks and jobs to run at provided times in the future.

FIREWALLS (MATEJ)

MOTIVATION

An increasing number of programs exploit the network connectivity or rely on the information available from remote servers and nodes. Further, on a distributed system, a malicious job that starts up unauthorized network servers or generates a high load of traffic would represent a huge liability. This means that, generally, we can expect that the worker nodes on an XtreamOS grid will employ firewall rules permitting a limited traffic, or possibly no outgoing traffic for the jobs at all. Additionally, we can view the network connectivity a resource which could represent a cost to the owner of the nodes.

We therefore need a way to express in a job description whether the job needs any network connectivity, and if, which specific transports. The purpose of this description can be twofold: during the resource selection, the nodes can evaluate the job request and decide whether they are willing to (or capable of) providing the required connectivity to the job. Additionally, if the system supports a dynamic per-job firewall rule set manipulation, it can instruct the application firewall which rules to apply, thus also protecting the job from any external exploits or attacks.

USECASE EXAMPLE

We can think of a job that has network connectivity as an important part of the overall execution, for instance, it needs an external web service's responses for its normal execution. When submitting the job, we specify which outgoing port it needs available and open. In the grid, we have most nodes protected with a firewall, and a few of them which permit outgoing traffic. The resource selection therefore returns only the nodes where the network traffic will not be blocked.

SYNTAX

```
<jsd1:Resources>
<network:Network
xmlns:net="http://xtreemos.org/schemas/jsdl/net/200805">
<net:Netmask>201.123.123.0/24</net:Netmask>
<net:Ports>1000-2000,60000,65000</net:Ports>

<net:Proto>TCP</net:Proto>
</network:Network>
</jsdl:Resources>
```

IMPACT OUT OF XTREEMOS

As this is common problem in Grids, it is not especially related to XtreamOS and can be used in any other Grid system.

INTERACTIVE JOBS (YVON)

MOTIVATION

Allowing the execution of interactive applications on the Grid extends the usability of the Grid to new domains: running matlab or SciLab on the grid, visualizing simulation results from a thin client, ...

Our proposition for interactive jobs is based on a job daemon. This job daemon is started with the job. This daemon listens on a UNIX socket for requests coming from the user. This job daemon also controls the execution of the job executable. The following main scenarios are possible:

- a. The executable is immediately started and the job daemon accepts new interactive commands from the user. This scenario allows users to interactively debug their jobs (for instance). In this case, the executable is not interactive.
- b. The executable is started only when requested by the user. In this case, the executable can be interactive as the user can provide a tty and/or a X11 display.
- c. Other variations around the main scenarios are possible.

In order to implement it, we propose to extend the JSDL. When requested from the jsdl, the execution manager starts the job daemon. The job daemon must receive the executable and its arguments by some means. The jsdl must also provide means to specify various scenarios.

The current implementation (demonstrated during the review+SAC meetings) is based on scheme b and does not require any extension to the jsdl. The main advantage of this solution is that it is completely independent from other XtreamOS services. The main advantage of solution is that the extensions to the jsdl can be exploited by other services (during resource negotiation for instance).

SYNTAX

We need to provide two means to interact with grid applications in XtreamOS: interacting with running jobs and running interactive applications.

1- Interacting with running jobs: extend JobDescription element with:

```
<opensession>true/false</opensession>
```

2- Running interactive applications: extend Application element with:

```
<inout>none,raw,TTY</inout>
```

```
<x11>true/false</x11>
```

IMPACT OUT OF XTREEMOS

The proposed extension could be used outside of XtreamOS as they do not depend on any implementation technique.

CHECKPOINTING (JOHN, MATEJ)

MOTIVATION

Flexibility, efficiency and safe execution of job fault tolerance and the operating system itself can be increased by taking checkpoint/restart-specific properties into account during job submission. The latter include:

- Checkpoint initiator: the operating system, the xos and/or the application.
- Favourite Checkpointer type: BLCR, SSI, OpenVZ, Linux-mainline, MTCP, etc.
- Checkpointer version.
- Favourite Container type: cgroup, OpenVZ container, BSD jails, Solaris zones, None.
- Container parameter: e.g. subsystems.
- Application software resources: match checkpointer capabilities against application software resources to identify best-suited checkpointer before job submission and restart.
- Checkpointing protocol: coordinated, uncoordinated, optimistic/pessimistic/causal message logging, uncoordinated and message logging, communication-induced.
- Checkpoint protocol parameter: full or incremental or concurrent checkpoint. checkpoint interval.

USECASE EXAMPLE

Job A is a single-threaded, single-process application with no further resources such as IPC-objects or open files. Since just a subset of checkpointers are able to handle this application, a suitable checkpointer must be identified with respect to the application software resources. Due to the existence

of merely one process, process synchronization overhead is minimal, coordinated checkpointing is to be applied exclusively.

Job B is a distributed application. The associated jsdl contains a list of the application resource types multiple threads, multiple processes, file descriptors, and sockets. The later allows to use BLCR and OpenVZ as suitable checkpointers. Since the application is distributed and has many processes the favourite checkpointing protocol is independent checkpointing with or without message logging. OpenVZ container features can be used to realise VO resource (memory, CPU) control The latter implies to use OpenVZ as checkpointer.

SYNTAX

```
<JobFaultTolerance>
  <Initiator>
    <User> yes </User>
    <OperatingSystem> yes </OperatingSystem>
  <Application>no</Application>
</Initiator>

<Checkpointer> BLCR </Checkpointer>

<CheckpointerVersion>0.8.2 </CheckpointerVersion>

<ContainerType>cgroups</ContainerType>

<ContainerParameter>

  <NetworkNS>yes</NetworkNS>

  <PidNS>yes</PidNS>

  <IpcNS>yes</IpcNC>

  <Uts>yes</Uts>
</ContainerParameter>

  <ApplicationSoftwareResources>

    <Singleproc>yes</Singleproc>
    <Singlethread>yes</Singlethread>
    <Sysipcshm>yes</Sysipcshm>
    <Sysipcmsgq>yes<Sysipcmsgq>

    <Sysipcsem>yes<Sysipcsem>

    <Files>yes<Files>

  </ApplicationSoftwareResources>

  <CheckpointProtocol>Coordinated</CheckpointProtocol>
  <CheckpointProtocolParameter>
    <Periodinseconds>5</Periodinseconds>
  </CheckpointProtocolParameter>
```

</JobFaultTolerance>

IMPACT OUT OF XTREEMOS

Resource node retrieval must be matched against the properties given here. Containers must be created at job submission and job restart time and configured towards the given container parameters. The grid checkpointing mechanisms must read in checkpoint protocol and strategy data to enforce checkpointing in the manner as listed.

References

- [1] Ali Anjomshoaa *et. al.* *Job Submission Description Language (JSDL) Specification, Version 1.0*. November 2005. <http://forge.gridforum.org/projects/jsdl-wg>
- [2] Omar Subhi Aldabbas. *A Framework for Mobility and Temporal Dimensions of Grid Systems*. PhD Thesis. 2008.