Expérience autour du gestionnaire de jobs OAR

Françoise Roch [1]

[1]Observatoire des Sciences de l'Univers de Grenoble

Sommaire

- Le contexte de CIMENT
- Pourquoi un nouveau gestionnaire ?
- Le processus de développement
- Les caractéristiques du logiciels
- Les fonctionnalités
- Les outils associés
- Perspectives
- Exemple d’infrastructure

Contexte

- **OSUG** : 600 personnes de 6 laboratoires + 4 équipes dans des domaines scientifiques relevant des Sciences de l'Univers : Astrophysique, Sismologie, Planétologie, Glaciologie, Hydrologie, …
  - ⇨ Un large spectre d'applications
  - ⇨ Des besoins importants en ressources de calcul, et hétérogènes en terme de modèle de programmation et d'architecture

  SCCI : Service Commun de Calcul Intensif de l'OSUG

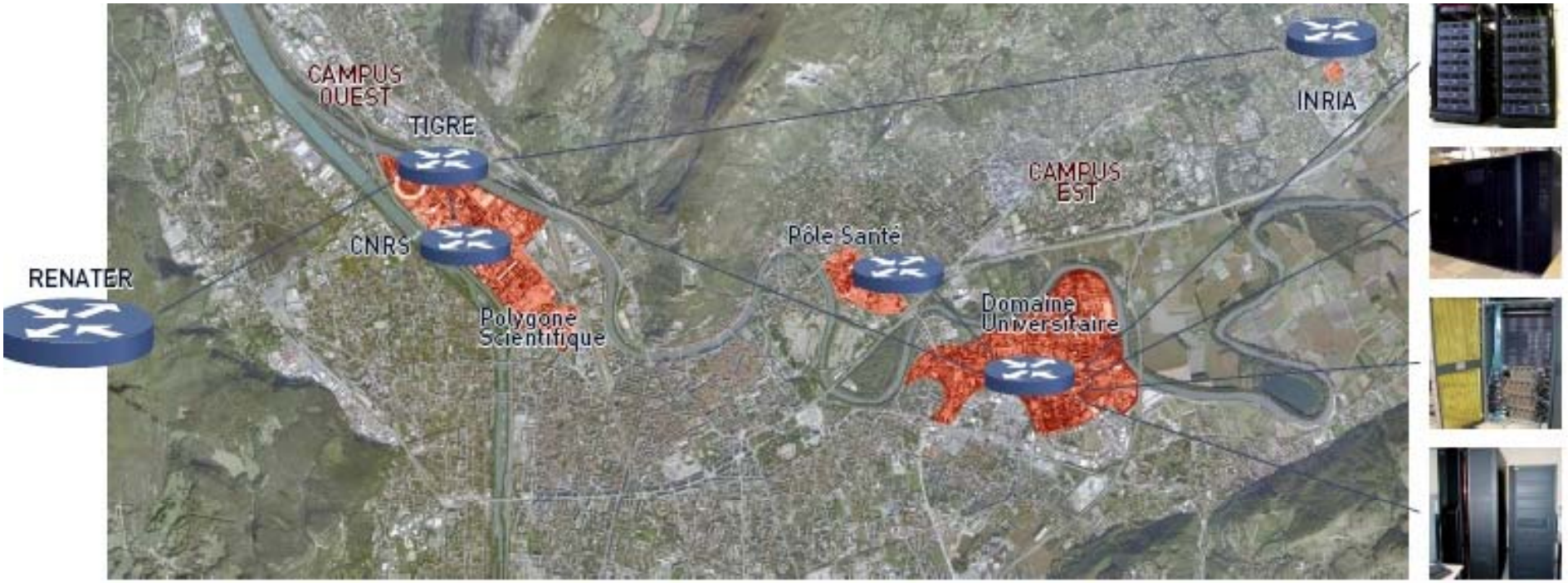
- **Projet CIMENT** : fédération des projets de calcul intensif sur les Universités Grenobloises :

  Plusieurs clusters, sur différents sites

  Un lien très fort entre recherche et production

  Développement de Collaborations avec la communauté des informaticiens

CIMENT



6 pôles sur grenoble

Grille de calcul CIMENT : environ 2200 cœurs, 18 Tflops

Environ 250 utilisateurs

Le gestionnaire de jobs

Gère l'attribution des ressources de calcul sur des Supercalculateurs, des grappes de serveurs de calcul

- En assurant la répartition de la charge
- Et en effectuant le suivi des jobs
- Et la surveillance de l'état des ressources

Un élément central pour le cluster

- doit être fiable, robuste, performant, souple

OAR : développé au sein d'équipes INRIA du labo LIG (Bruno Bzeznik, Nicolas Capit, Olivier Richard, … large expertise sur les outils pour cluster et grille, et sur l'ordonnancement) http://oar.imag.fr

Pourquoi un nouvel outil ?

De multiples gestionnaires de jobs propriétaires ou libres :

PBS Pro, LSF, Loadleveler, Moab, …

Torque/MAUI,SGE, Slurm, …

Des codes monolithiques d'extension problématique

Beaucoup de fonctionnalités mais 80 % utilisées par la majorité des utilisateurs

Objectifs des concepteurs d'OAR : passage à l'échelle, modularité, extensibilité, robustesse

Intérêt pour les utilisateurs :

Un code « moderne », qui peut évoluer avec l'évolution des technologies

OAR

- Développé au départ (2003) pour être à la fois une plateforme de production et une plateforme de recherche (notamment en vue du développement d'un intergiciel de Grille)

Les points importants :

- Gestion des erreurs
- Nettoyage des nœuds
- fiabilité
- Faibles temps de latence à la soumission
- Bonne tenue en charge
- Mise en place de politiques d'ordonnancement
- extensibilité

Le processus de développement
Les outils utilisés

- **Prise en compte des besoins utilisateurs** :

  -Lien étroit avec les sites de production, réunions régulières au démarrage du projet

  -Expérimentation sur cluster INRIA mais également dans les laboratoires et sur les plateformes du projet CIMENT

  -Au départ, se concentrer sur les besoins classiques des sites de production

- **Utilisation d'outils standard et robustes :**

  -Base de données Mysql /PostgreSQL

  -maintient l'ensemble de l 'état du système

  -Utilisée comme mode unique d'échange d'informations , chaque module interagit via des requêtes SQL

  -Langage de script Perl , Ruby

  des structures de données de haut niveau , un cycle de développement court

Architecture hiérarchique des ressources

S’adapter à la topologie des machines (architectures multi-cœurs, NUMA)

L’utilisateur à la possibilité d’exprimer une notion de hiérarchie à la soumission du job :

- Ex : /cluster/switch/node/cpu/core

  oarsub –I –l node=2/cpu=1/core=2

Utilisation des cpuset

- Principe :

  sur un nœud de calcul, les tâches sont confinées au sein d'un groupe de cores, associé à un espace mémoire physique

- Avantages :
  - Isolement des process, pas d'impact sur les autres jobs qui s'exécutent sur le même nœud
  - L'accès aux ressources est réservé aux propriétaires des ressources
  - Permet le nettoyage complet d'un job sans gêne pour les autres jobs, même pour les jobs //

- Inconvénients :
  - Pas très intuitif pour les utilisateurs
  - Configuration un peu complexe

Le cycle d’un job

- Soumission sur une machine frontale
- Le job passe par une règle d’admission
- La requête est enregistrée dans la base, sur le serveur OAR
- Un identificateur de job est renvoyé ; il permettra à l’utilisateur de suivre son job
- Le serveur «planifie » la tâche
- La tâche est exécutée sur les ressources demandées

Règles d'admission

- Permettent de cadrer la requête
- De personnaliser le fonctionnement
- De définir des limites par défaut (nb cores, walltime, …)

Ex: 5 nœuds ont 32 Go de mémoire et une partie du cluster est en infiniband

L'admin:

```
oarproperty –a –c memnode
oarnodesetting –p " memnode=32768"  -h node60
oarnodesetting –p " ib=yes"  -h node60
```

L'utilisateur :

```
oarsub –p "memnode=32768 and  ib=yes"   ./monjob
```

Ordonnancement

- Configuration de la base de données OAR :
  Entrée des hostnames, attribution de propriétés

- **Appariemment des ressources :** chaque requête peut demander des ressources avec des propriétés spécifiques (similaire à PBS)

- Des files d'attentes avec des priorités différentes
- Chaque file a son propre ordonnanceur
  Implémentation de différentes politiques d'ordonnancement :
  FIFO (First In First Out), backfilling (First Fit), fairsharing (partage équitable)
  besteffort (occupation des ressources inutilisées, tâches non interruptibles)

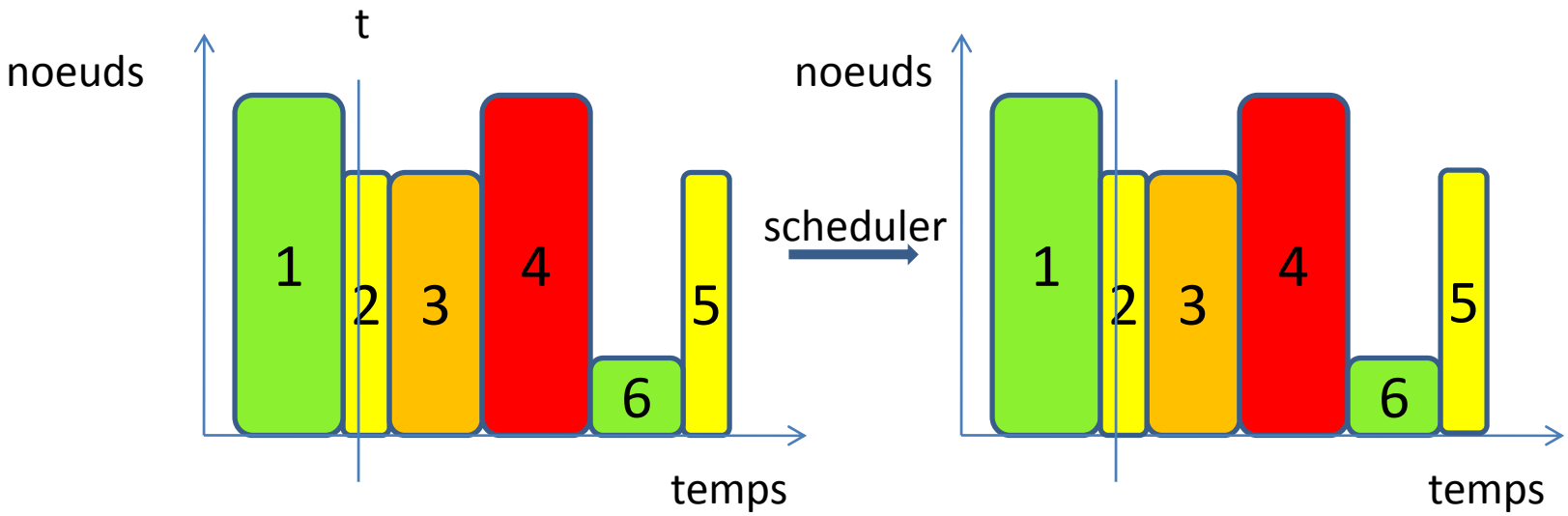
Remarque:

D'autres gestionnaires ont une politique d'ordonnancement où toutes les tâches sont ordonnancées en même temps, ici on perd la vision globale des tâches

Ordonnancement moins agressif, mais plus simple à gérer

Ordonnancement

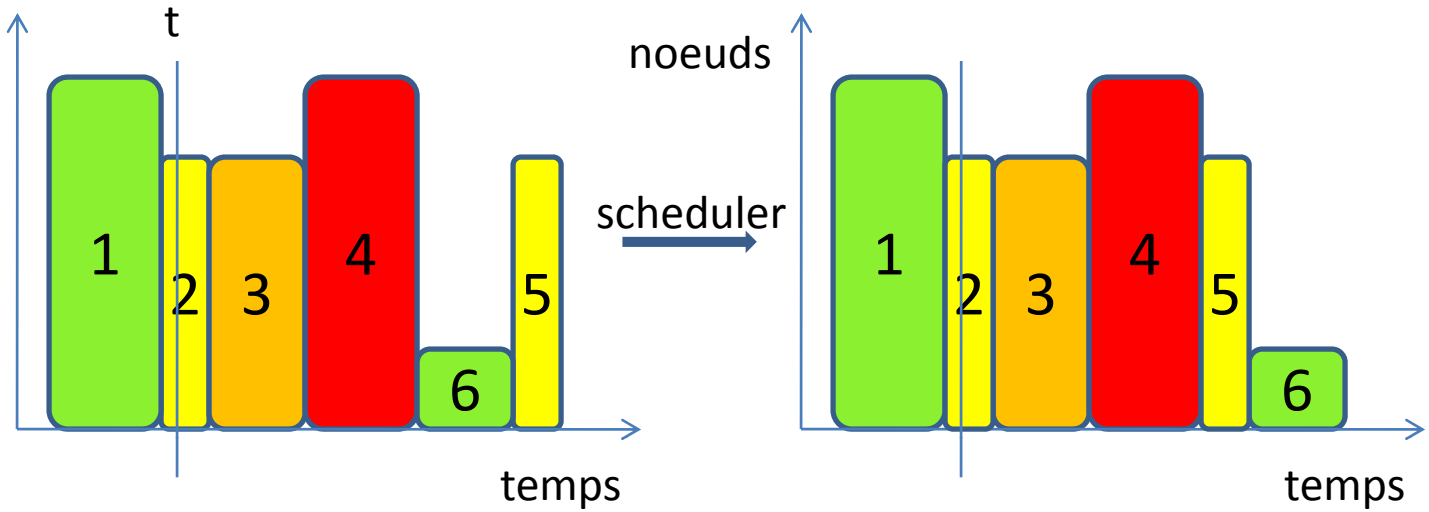
FIFO



Ordonnancement

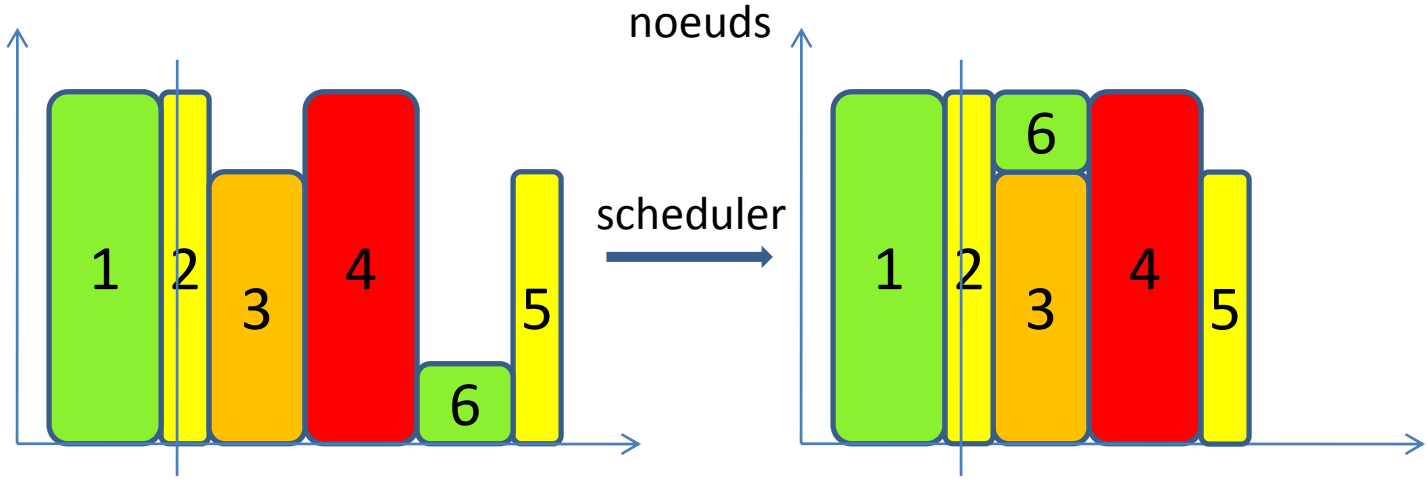
Fairsharing

Partage équitable des ressources en tenant compte de la consommation sur une fenêtre de temps donnée par utilisateur et par projet



Backfilling

Remplissage des trous sans modifier l'ordre des tâches précédentes



Installation du logiciel

- Un serveur
- Un nœud de login à partir duquel s'effectuent les réservations
- Des nœuds de calcul

Packages prérequis

- sur tous les noeuds : Perl, Perl-base, openssh
- sur le serveur: Perl-Mysql, Perl-DBI, Mysql, libmysql

Sur le **serveur** : un démon "Almighty" lancé par /etc/init.d/oar-server start

Sur les **nœuds de calcul** : /etc/init.d/oar-client start

Remarque : contrairement aux autres gestionnaires, **pas de démon sur les nœuds** (Par ex : LSF – au moins 5 démons sur les noeuds, , SGE – 1 démon + 1 par job, PBS Pro 1 démon, Loadleveler –au moins 3)

Différents types de tâches

- Type « besteffort »
- Type « timesharing »
- Type « deploy »
- Type « moldable » (flexible)
- Type « container »

option "-t " de oarsub

Un exemple de tâche flexible

- Mon cluster a des nœuds bi-pro, proc. 4 cores
- Les nœuds ont 8 Go, 5 noeuds en ont 32

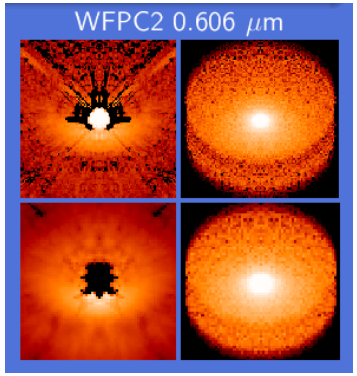
- Mon programme est mixte OpenMP-MPI
- Chaque tâche MPI utilise 6 Go, 12 tâches MPI

Ex de soumission :

```
oarsub –l " {memnode=’32’}/node=3, walltime=2:00:00"
          -l "node=12/core=6,walltime=3:00:00"  ./monprog
```

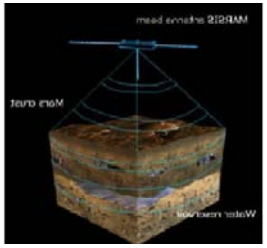
OAR choisit la solution qui se termine le plus tôt.

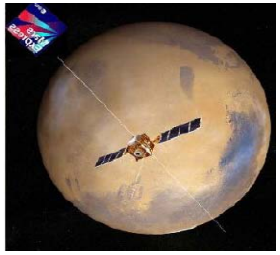
Des exemples d'utilisation du mode besteffort



- Ajustement simultané d'observations possibles de disques protoplanétaires.
- Utilisation d'algorithmes génétiques pour l'exploration d'un espace de paramètres.
- 8.6 années de temps CPU sur Ciment

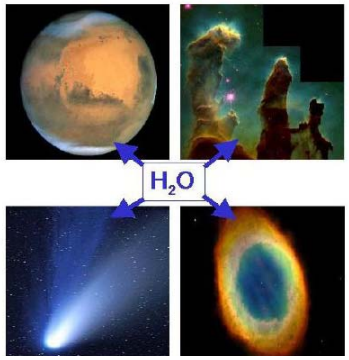
C. Pinte, F. Menard





- Simulation d'echo radar : recherche de l'eau sur Mars (environ 800 jours de calcul).
Simulation pour optimiser la fréquence utilisée et sélectionner des orbites pertinentes.

J-F. Nouvel, W. Kofman, A. Herique



- Application à la chimie quantique : calcul de trajectoires moléculaires, calcul des taux de désexcitation vibrationnelle du pliage de l'eau (environ 8 ans de temps CPU).

A. Faure, P. Valiron, M. Wernli, L. Wiesenfeld

Premières expérimentations en 2004

Objectif : exploiter les ressources de calcul inutilisées au sein du laboratoire;

les stations de travail sont utilisées comme serveurs de calcul.

Le gestionnaire utilisé est OAR.

Aggregation des ressources via Oucapo, prototype du logiciel Computemode (labo ID, Société Icatis).

Certains PCs ne sont utilisés que la nuit

(mécanisme de Wake-On-Lan, PXE, systèmes « diskless »)

Une interface d'administration : intégration des ressources, définition de plages horaires d'utilisation, …

⇨ Construction d'un cluster de calcul virtuel formé de Stations de travail diskless bootées via le réseau.

Quelques fonctionalités intéressantes

La réservation

- **Réservation** de ressources :

```
# oarsub –r "2010-06-26" –l nodes=10/core=2
… OAR_JOB_ID=2679
```

Pour s'y connecter :

```
# oarsub –C 2679
```

Connexion au 1er nœud et chargement de l'environnement OAR

Ex. d'utilisation :

planifier des campagnes de simulation

réserver un nombre important de ressources.

Quelques fonctionalités intéressantes

Le container

- **Container** : exécution de plusieurs jobs ensembles

```
# oarsub –I –t container  -lnodes=10,walltime=2:00:00
   .. OAR_JOB_ID=2679
# oarsub –I –t inner=2679 -l nodes=2
# oarsub –I –t inner=2679 -l nodes=4
# oarsub –I –t inner=2679 -l nodes=4
```

Ex d'utilisation : réservation de ressources pour une formation pour le 26/06:

```
# oarsub  –t container  -r "2010-06-26 08:00:00" -lnodes=10,walltime=4:00:00
        .. OAR_JOB_ID=2680
```

Quelques fonctionalités intéressantes

- Possibilité de scripts Epilogue/Prologue
- Soumission d'un script avec directives OAR :

  # oarsub –S monscript
- Dépendance de jobs, soumission d'un job après terminaison d'un autre (antériorité) :

  # oarsub -a jobid …
- Possibilité d'intégration d'outils de déploiement d'OS
- Tableaux de jobs
- Des ressources de différents types : par ex. support des licences logicielles
- Gestion de l'énergie

Accounting

- Une table Accounting contient un bilan de la consommation par utilisateur, sur une fenêtre de temps de taille spécifiée dans le fichier de config d'OAR

Optimise la recherche du temps de consommation/utilisateur et l'obtention de statistiques

Ex : oarstat --accounting " 2010-01-01, 2010-01-06"

User	First window starts	Last window ends	Asked (seconds)	Used seconds)
roch	2010-01-11 01:00:00	2010-05-22 01:59:59	4356792	2145256
stage9	2010-01-15 01:00:00	2010-03-06 00:59:59	1180800	246121
dupont	2010-01-05 01:00:00	2010-05-13 01:59:59	836049600	145792185
stage1	2010-01-15 01:00:00	2010-01-30 00:59:59	115560	43168
durand	2010-01-15 01:00:00	2010-05-16 01:59:59	10234800	4336529
toto	2010-01-05 01:00:00	2010-03-02 00:59:59	223869600	122091285
tutu	2010-01-07 01:00:00	2010-05-30 01:59:59	16891200	4472600

Le monitoring des jobs

- Monika : visualisation de l'état instantané de toutes les ressources. Ecrit en Perl.

Fostino/R2D2 nodes

default summary

	Free	Busy	Total
host	5	85	123
core	116	604	980

Reservations:

f11	985836	985836	985836	985836	985836	985836	985836	985836
f12	867017	867017	867017	867017	867017	867017	867017	867017
f13	StandBy	StandBy	StandBy	StandBy	StandBy	StandBy	StandBy	StandBy
f14	StandBy	StandBy	StandBy	StandBy	StandBy	StandBy	StandBy	StandBy
f15	985834	985834	985834	985834	985834	985834	985834	985834
f16	985839	985839	985839	985839	985839	985839	985839	985839

Monitoring

- Monika

Selection d'une propriété

Le monitoring des jobs

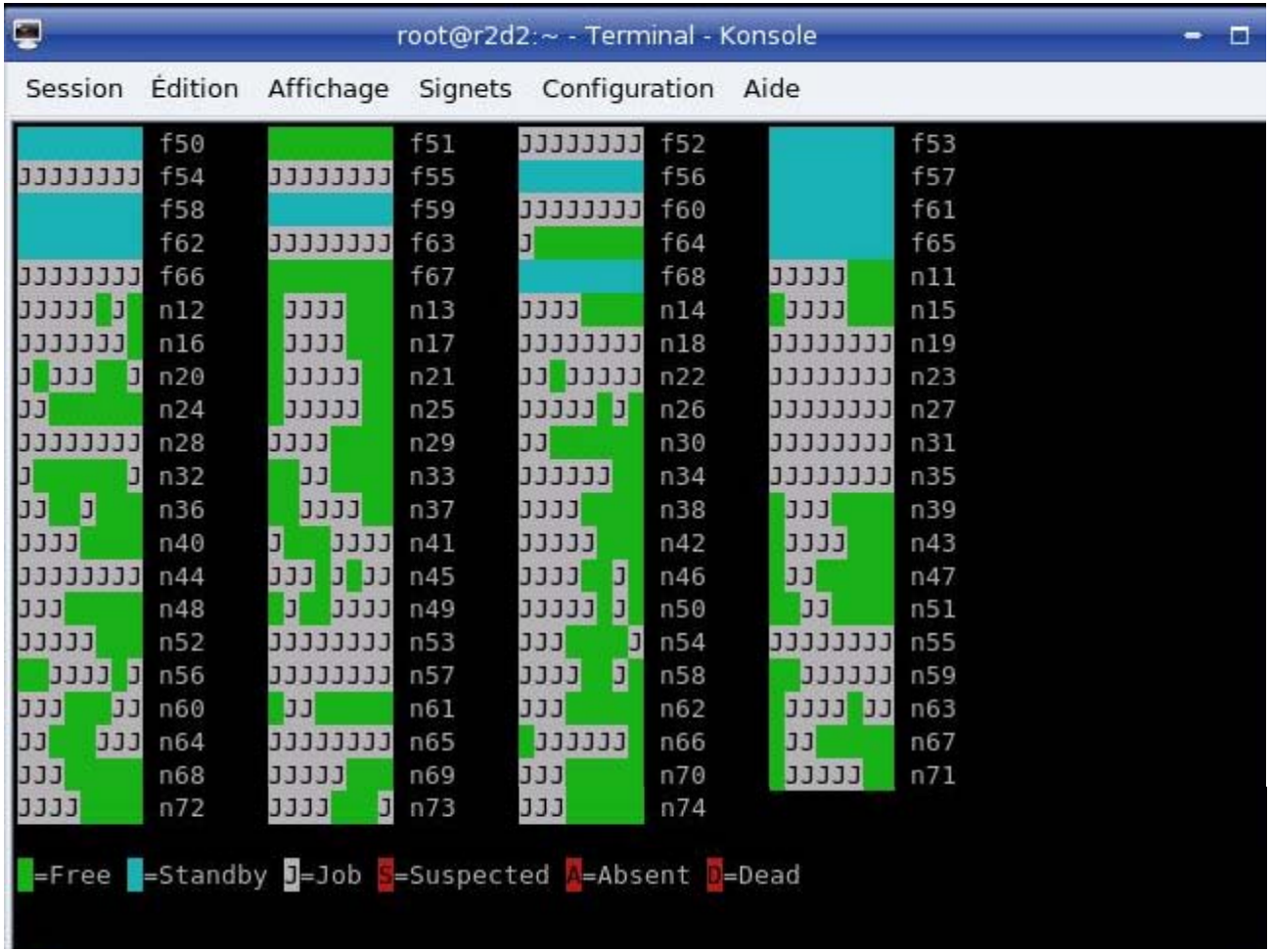
- Monika

Liste des jobs

983685	mjay	Running	r2d2	-l "{type = 'default'}/core=1,walltime=200:0:0"	PASSIVE	((desktop_computing = 'NO') AND cluster = 'r2d2') AND post = 'NO'	None	200:0:0
983687	dupuyb	Waiting	r2d2	-l "{type = 'default'}/host=10 /core=8,walltime=13:0:0"	PASSIVE	((desktop_computing = 'NO') AND cluster = 'r2d2') AND post = 'NO'	None	13:0:0
983688	dupuyb	Waiting	r2d2	-l "{type = 'default'}/host=10 /core=8,walltime=13:0:0"	PASSIVE	((desktop_computing = 'NO') AND cluster = 'r2d2') AND post = 'NO'	None	13:0:0
983740	durand	Running	r2d2	-l "{type = 'default'}/core=1,walltime=100:0:0"	PASSIVE	((desktop_computing = 'NO') AND cluster = 'r2d2') AND post = 'NO'	None	100:0:0
984652	roquesa	Running	r2d2	-l "{type = 'default'}/host=8 /core=1,walltime=40:0:0"	PASSIVE	((desktop_computing = 'NO') AND cluster = 'r2d2') AND post = 'NO'	None	40:0:0
985054	navasg	Running	r2d2	-l "{type = 'default'}/host=1 /cpu=1,walltime=3000:0:0"	PASSIVE	((desktop_computing = 'NO') AND cluster = 'r2d2') AND post = 'NO'	None	3000:0:0

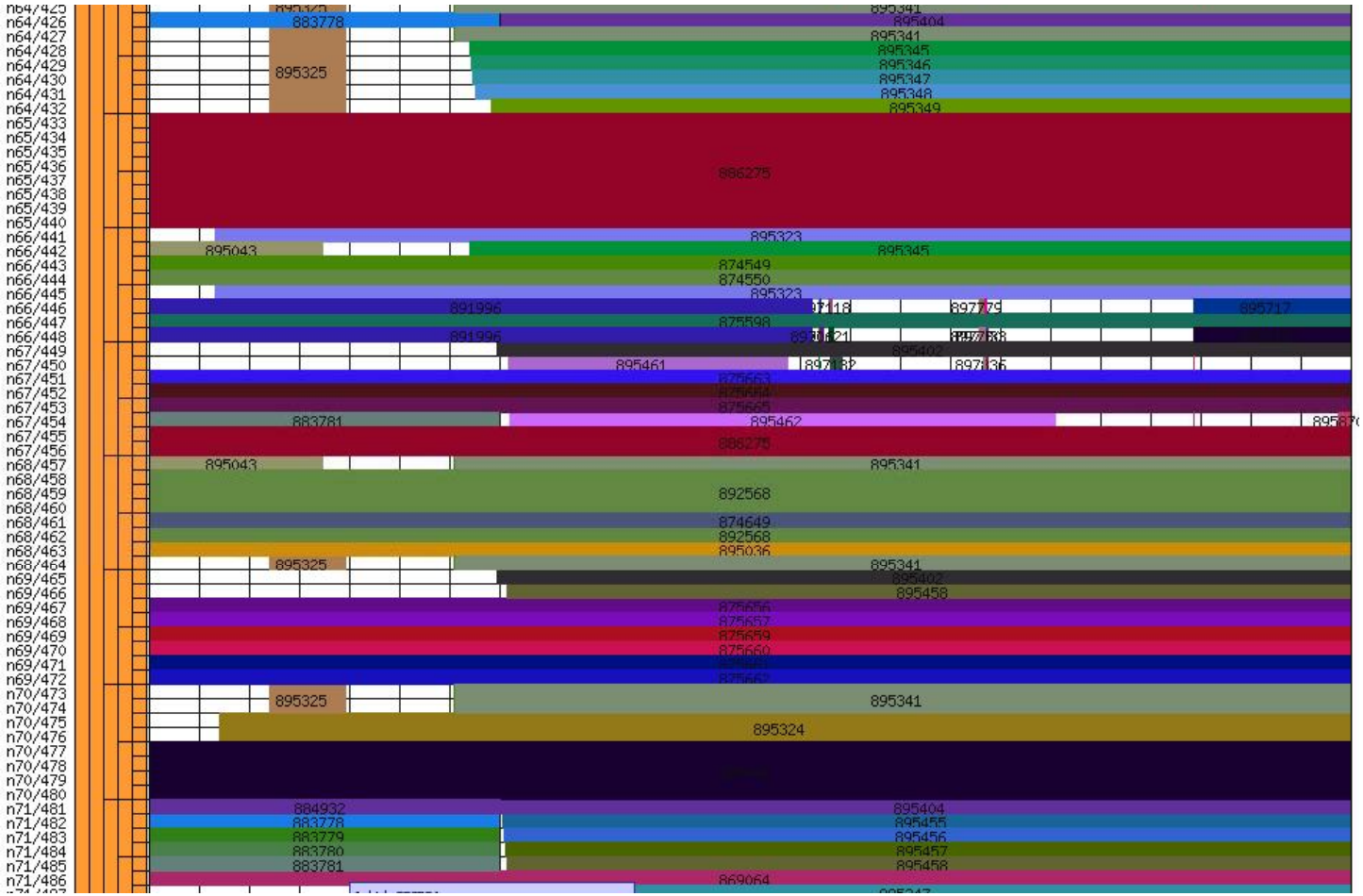
Le monitoring des jobs

- chandler



Le monitoring des jobs

- DrawGrantt : visualisation de l'occupation des ressources dans le temps. Ecrit en Ruby.



Les limites

- Pour l'instant pas d'interface web pour la soumission
- Très faiblement couplé à l 'environnement d'exécution :
  - Modèle de script différent selon le modèle //
  - Les process « spawnés » échappent au contrôle d 'OAR
  - Problèmes éventuels avec OpenMPI
- Forte potentialité mais réglage fin complexe :
  - hiérarchie de ressources
  - mécanisme des règles d'admission
  - paramétrage

examples de script OAR et Loadleveler

```
cat openmp.oar
# Fichier de sortie standard
#OAR –O output.$(OAR_JOB_ID)
#OAR –E error .$(OAR_JOB_ID)
# Nombre de threads et temps elapsed
#OAR –l nodes=1/core=4,walltime=1:30:00
export OMP_NUM_THREADS=4
export XLSMPOPTS=stack=65536000

cd $OAR_WORKDIR
# Execution
./a.out
```

oarsub –S ./openmp.oar

```
cat openmp.ll
# Fichier de sortie standard
# @ output=output.$(jobid)
# Fichier de sortie d'erreur
# @ error =error.$(jobid)
# Type de travail
# @ job_type = serial
# Nombre de cores demandes
# @ resources = consumableCpus(4)
# Temps Elapsed
# @ wall_clock_limit = 1:30:00
export XLSMPOPTS=stack=65536000

cd $LOADL_STEP_INITDIR
# Execution
 ./a.out
```

llsubmit openmp.ll

examples de script OAR et Loadleveler

```
cat mpi.oar
# Fichier de sortie standard
#OAR –O output.$(OAR_JOB_ID)
#OAR –E error .$(OAR_JOB_ID)
# nodes,core et temps elapsed
#OAR –l nodes=8/core=1,walltime=1:30:00

cd $OAR_WORKDIR
# Execution
mpirun –np 8 ./a.out
```

oarsub –S ./mpi.oar

```
cat mpi.ll
# Fichier de sortie standard
# @ output=output.$(jobid)
# Fichier de sortie d'erreur
# @ error =error.$(jobid)
# Type de travail
# @ job_type = parallel
# Nombre de processus demandes (ici 8)
# @ total_tasks = 8
# Temps Elapsed
# @ wall_clock_limit = 1:30:00

cd $LOADL_STEP_INITDIR
# Execution
 ./a.out
```

llsubmit mpi.ll

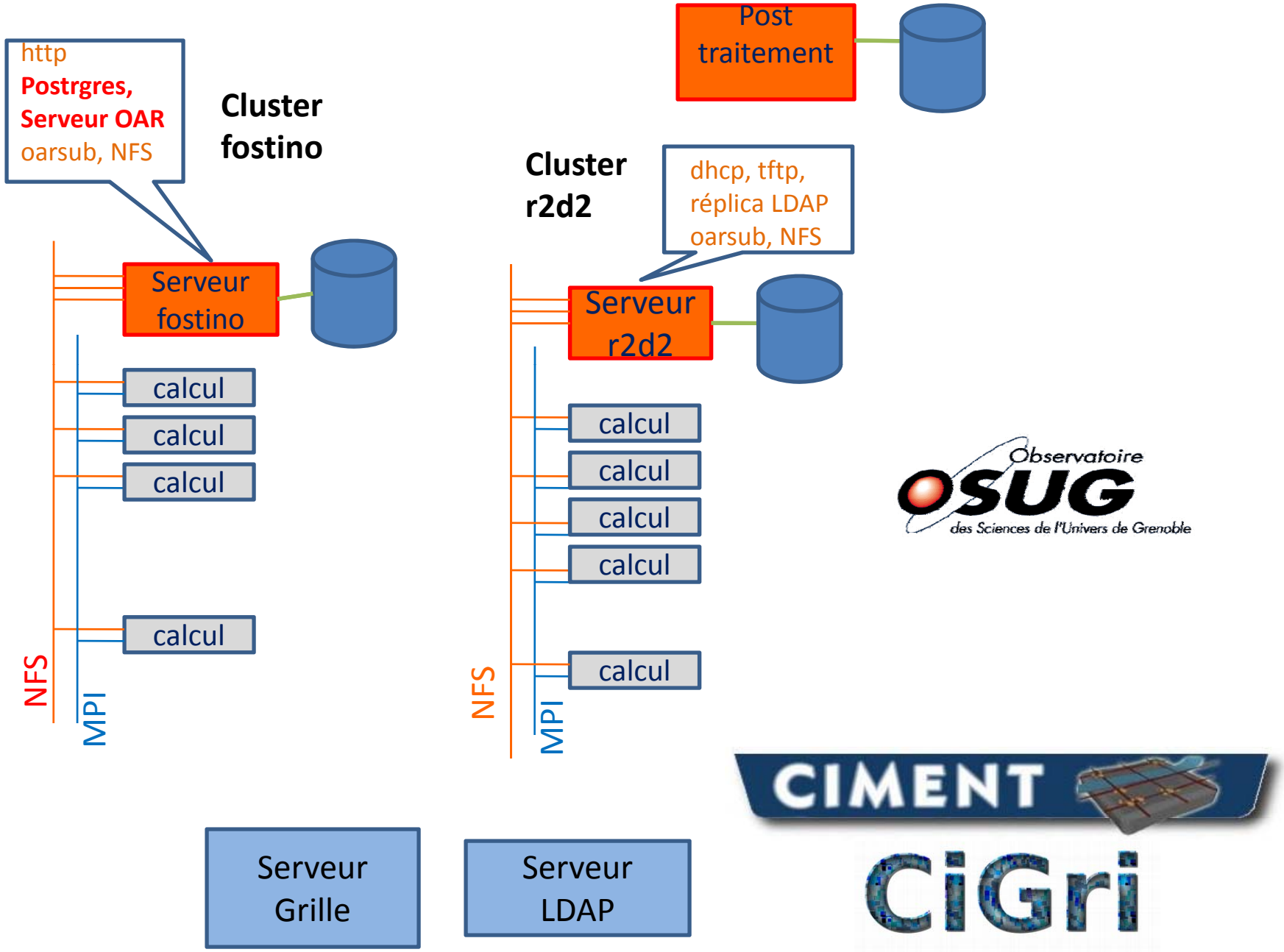
Perspectives

Portail WEB pour la soumission

- doit fournir les mêmes options que la ligne de commande

Monitoring de l’énergie, rapport utilisateurs, projets …

Un exemple d'Infrastructure de production



Un exemple d'infrastructure de production

Niveau CIMENT :

- Gestionnaire de grille « légère » CiGri
- Un serveur LDAP, gestion centralisée des comptes (chaque administrateur a les droits d'accès)
- Demande de création de compte via un formulaire web :

  un groupe primaire donne l'appartenance à un labo

  une liste de groupes secondaires indiquent les clusters auxquels l'utilisateur a accès
- Une application web « rapport » associée à la base LDAP : les utilisateurs doivent mettre à jour un rapport d 'activités
- /applis/CIMENT : versions des logiciels compilés pour être compatibles avec toutes les plateformes de la grille

Niveau cluster local :

- gestionnaire OAR
- Duplica de la base LDAP
- /applis/site : versions des logiciels compilés, optimisés pour la plateforme

Conclusion

- Un logiciel libre moderne, offrant simplicité et extensibilité
- Possibilité de personnalisation :
  - Paramétrage fin grâce aux règles d'admission
  - Des modules complémentaires peuvent être écrits dans n'importe quel langage ayant une librairie d'interfaçage avec les bases de données
- Gestion d'énergie
- Des mises à jour avec peu d'impact sur l'exploitation