Utilisation de grilles de calcul pour la génomique comparative

C. Blanchet,P. Calvat,Y. Cardenas, C. Gauthey,J-Y Nief, S. Penel, B.Spataro

Comparaison de deux grilles de calcul: la plateforme GRISBI & la plateforme TIDRA

GRIlle, Support pour la BIoinformatique
Traitement de données et Informatique Distribuée en Rhône-Alpes

PLAN

- Contexte biologique
  - La phylogénomique
  - La base de séquences de protéines non redondantes BGENR
  - Calcul de similarité sur BGENR
- Calcul sur la 1ère release de BGENR :
  - Calcul sur TIDRA
  - Stratégie, problèmes recontrés avec TIDRA
- Calcul sur la 2ème release de BGENR :
  - Nouveaux outils mis en place sur TIDRA
  - Stratégie, problèmes rencontrés avec TIDRA
  - Calcul sur GRISBI
  - Stratégie, problèmes rencontrés sur GRISBI
- Conclusion et perspective

Contexte biologique
la phylogénomique : phylogénie + génomique
du grec *phylon* « tribu »

génomes (tout l'ADN) ➔ gènes (ADN codant) ➔ protéines

Contexte biologique

la phylogénomique : phylogénie + génomique

- Bases de données dédiées à la phylogénomique
  - HOVERGEN : vertébrés
  - HOGENOM : génomes complets (plus de 1000 génomes)
- Principe : pour chaque base de données :
  - Calcul de similarité « tout contre tout » sur les protéines (BLAST 2.2)
  - Clustering en familles (SiLiX, clustering ultra-rapide)
  - Calcul d'alignement et d'arbre phylogénétique pour chaque famille (MUSCLE, PHYML)
- Généralisation : banque de séquence « BGENR »
  - Création d'une **base unique non redondante** contenant toutes les séquences de protéines
  - Servira de point de départ pour la construction de HOVERGEN et HOGENOM

GRILLE : Calculs de similarité sur des grands nombres de séquences de protéine
(de l'ordre de 8,000,000 x 8,000,000)
Logiciel BLAST

Contexte biologique

la phylogénomique : phylogénie + génomique

Données:

séquences des proteines, format FASTA

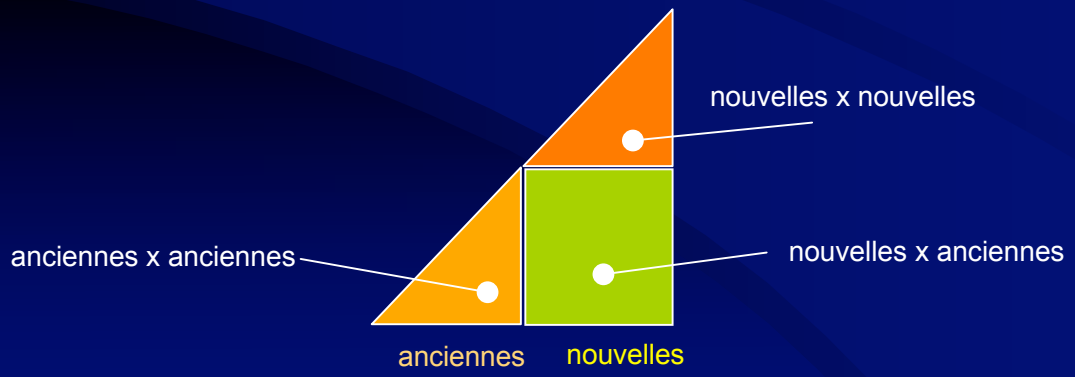
```
>1A1L1_BOVIN            502 RecName: Full=1-aminocyclopropane-1-carboxylate synthase-like protei
MFTLPQKEFRMTTACPGSDSIQDLPSNKGDGLERECSRKPDQKLLKFYGVGDPAAELSSS
SPYLSSRGSVIKWFWDSAEEGYRTYHMDEYDEDKNPSGIINLGTSENKLCFDLLSRRLSQ
SDMLQVEPALLQYPDWRGHLFLREEVARFLSFYCRSPAPLKPENVVVLNGCASLFSALAT
VLCEAGEAFLIPAPYYGAITQHVYLYGNVRLVCVYLDSEVTGLETRPFQLTVEKLEMALQ
GANSEGVKVKGLILINPQNPLGDIYSPGELQEYLEFAKRHELHVMVDEVYMLSVFEESAG
QYQMAQLLRDHDWINQVYLPENHARLKAAHTYVSEDLRALGIPFVSRGAGFFIWVDLRKY
LPEATFEEEVLLWRRFLENKVLLSFGKAFECKEPGWFRLVFSDKTHRLHLGMQRVRQVLE
GQPQLADGAPPHQIQEPQGPHR
>1A1L1_HUMAN            501 RecName: Full=1-aminocyclopropane-1-carboxylate synthase-like protei
MFTLPQKDFRAPTTCLGPTCMQDLGSSHGEDLEGECSRKLDQKLPELRGVGDPAMISSDT
SYLSSRGRMIKWFWDSAEEGYRTYHMDEYDEDKNPSGIINLGTSENKLCFDLLSWRLSQR
DMQRVEPSLLQYADWRGHLFLREEVAKFLSFYCKSPVPLRPENVVVLNGGASLFSALATV
LCEAGEAFLIPTPYYGAITQHVCLYGNIRLAYVYLDSEVTGLDTRPFQLTVEKLEMALRE
AHSEGVKVKGLILISPQNPLGDVYSPEELQEYLVFAKRHRLHVIVDEVYMLSVFEKSVGY
YQMAQLLRDRDWINQVYLPENHARLKAAHTYVSEELRALGIPFLSRGAGFFIWVDLRKYL
PKGTFEEEMLLWRRFLDNKVLLSFGKAFECKEPGWFRFVFSDQVHRLCLGMQRVQQVLAG
KSQVAEDPRPSQSQEPSDQRR
>1A1L1_MOUSE            502 RecName: Full=1-aminocyclopropane-1-carboxylate synthase-like protei
MFCLPQQESTAPTTCTGSASTQDMDSGYGDGLQGECLRKPDQTQPKLYGVGDPTATFSSD
SSCLSSRGRVIKWFWDSAEEGYRTYHMDEYDEDKNPSGIINLGTSENKLCFDLLSWRLTQ
GDMLHVEPSLLQYPDWRGHLFLREEVAKFLSFYCKSPAPLKPENVVVLNGCASLFSALAT
VLCEAGEALLIPTPYYGAITQHIYLYGNVRLAYVYLDSKVTGLNTRPFQLTVEKLEMVLQ
GVSSEGVKVKGLILINPQNPLGDVYSPEELQDFLRFAMRHKLHVIMDEVYMLSVFEESLG
QHQMAQLLRDHDWISQVYLPENHARLKAAHTYVSEELRALGIPFVSRGAGFFIWVDLRKY
LCKGTFEEEALLWRQFLDNKVLLSSGKTFECKEPGWFRVVFSDKENRLRLGMQRMRQVLE
GQSQVVEDASPCHAQEPQSQPR
>1A1L2_HUMAN            568 RecName: Full=1-aminocyclopropane-1-carboxylate synthase-like protei
MSHRSDTLPVPSGQRRGRVPRDHSIYTQLLEITLHLQQAMTEHFVQLTSRQGLSLEERRH
TEAICEHEALLSRLICRMINLLQSGAASGLELQVPLPSEDSRGDVRYGQRAQLSGQPDPV
PQLSDCEAAFVNRDLSIRGIDISVFYQSSFQDYNAYQKDKYHKDKNTLGFINLGTSENKL
CMDLMTERLQERDMNCIEDTLLQYPDWRGQPFLREEVARFLTYYCRAPTRLDPENVVVLN
......
```

Calculs BLAST incrémentiels

A chaque release, on classe les séquences en « nouvelles » et « anciennes »
On calcule :

- la similarité des séquences *nouvelles entre elles*
  (BLAST nouvelles x nouvelles)
- la similarité entre les séquences nouvelles et les séquences anciennes
  (BLAST nouvelles x anciennes)

BLAST n+1 : BLAST nouvelles x nouvelles + BLAST nouvelles x anciennes + BLAST anciennes x anciennes
(BLAST anciennes x anciennes = BLAST n)



Données utilisées

1ère Release BGENR (2009-2010) 7,775,267 séquences

Calcul de similarité sur toutes les séquences de protéine :

BLAST de environ 8,000,000 x 8,000,000 séquences TIDRA

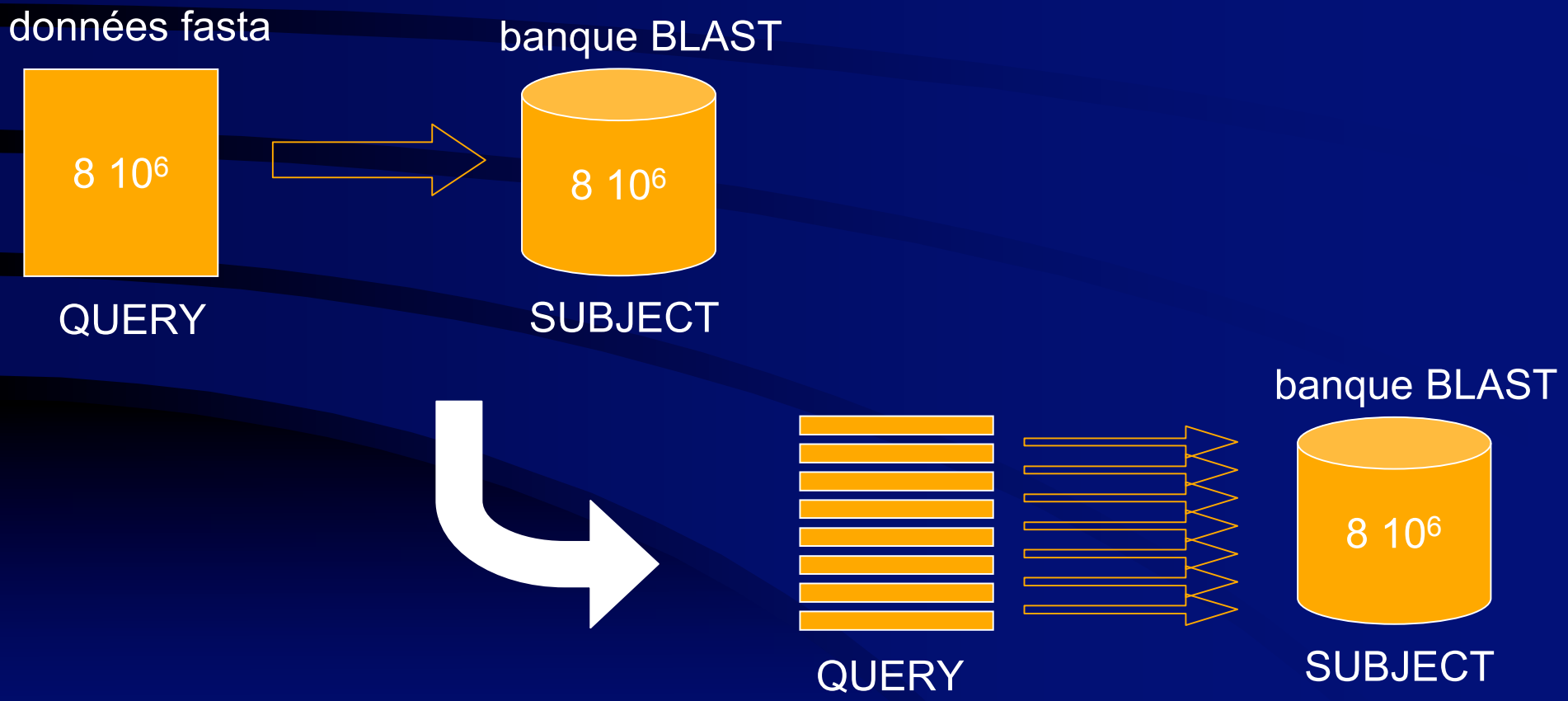
2ème Release BGENR(2011) 12,914,469 séquences

5,000,000 nouvelles BLAST x 5,000,000 nouvelles LOCAL

5,000,000 nouvelles BLAST x 7,500,000 anciennes TIDRA/GRISBI

5,000,000 nouvelles BLAST x 150,000 disparues GRISBI

Parallélisation par les « *queries* »
(pas de MPI BLAST)



Technologie grille et services associés sur TIDRA (Grille Rhône-Alpes)

- 7000 cœurs (cpu)
- 300 To de stockage
- 5 Sites
  - LAPP (Annecy)
  - LPSC (Grenoble)
  - IPNL (Lyon)
  - IBCP (Lyon)
  - CC-IN2P3 (Lyon)

Technologie grille et services associés TIDRA RAGRID (Grille Rhône-Alpes)

- *Middleware* :
  - *Job management* : gLite, LRMS
  - Stockage : iRODS, SRM
  - Utilisateur : JSAGA implementation SAGA

vo.rhone-alpes.idgrilles.fr

Mise en place sur TIDRA 1ère release (2009-2010)

1ère mouture de BGENR : séquences Uniprot (9 millions)

8 millions de séquences non redondantes à comparer.

Historique :

Mise en place de l'outil avec 3 membres du CC - **Y.Cardenas**, P. Calvat, J.Y. Nief

1er contact avec la grille		Novembre 2008
Premiers tests de charge et développements	blast intensifs	Février 2009
Arrivée de iRODS	Gros soulagement!!!	Juin 2009
Début de la production + développement		Juillet 2009
Fin de la production		Janvier 2010

Contraintes dues à la mémoire

Mémoire minimum des machines : 2 Go

La solidité d'une chaîne dépend du maillon le plus faible :
on veut éviter un dépassement de mémoire sur les machines les moins puissantes

La "tâche unitaire" compatible avec une mémoire de 2 Go est :

traitement d'un fichier de 30 séquences

8 millions de séquences → 250 000 fichiers

Optimisation du temps passé en calcul

Le calcul d'1 tâche unitaire est très court : env. 15 minutes

Bien inférieur au temps disponible dans un job :
variable selon les machines
quelques heures - quelques jours

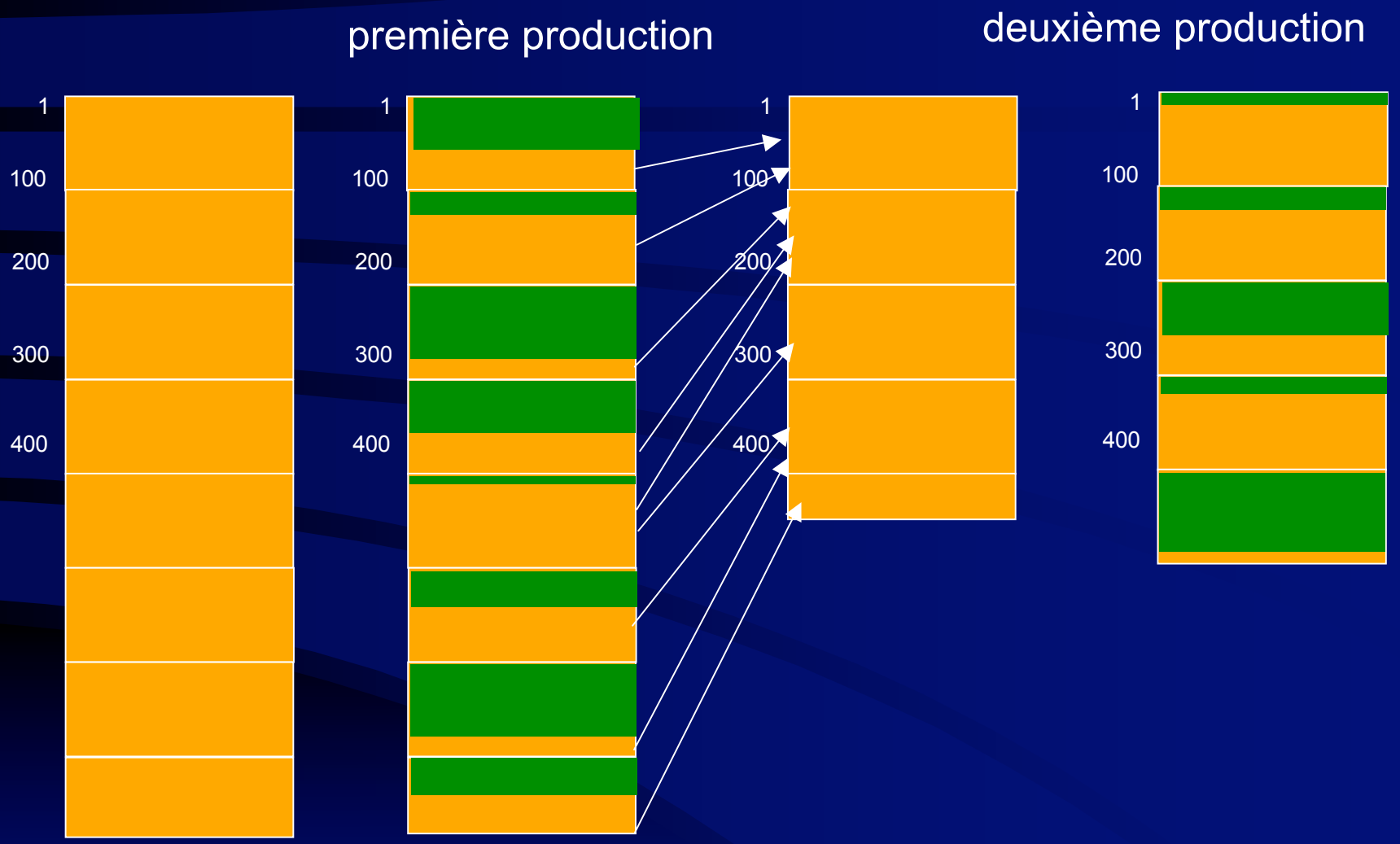
Chaque job doit exécuter le plus grand nombre possible de tâches unitaires

Stratégie adoptée
Plusieurs tâches par job

- 1 - Liste de tâches à effectuer (250,000 fichiers de 30 séquences)
- 2 - Chaque job N tente de traiter les 100 tâches à partir de la tâche numéro N x 100
- 3 - Une fois tous les jobs terminés, génération d'une nouvelle liste de tâches à traiter
- 4 - Retour au point 1

On utilise des jobs paramètriques, le paramètre est N

Stratégie


première production
deuxième production
1
100
200
300
400
1
100
200
300
400
1
100
200
300
400
1
100
200
300
400

Description du JDL

N est le paramètre du *job* paramétrique

- Déroulement d'un *job* numéro N :
  - récupération de différents outils via lcg-cp :
    - outils iRODS
    - outils pour l'estimation du temps de calcul
    - outils pour la gestion des *proxies*
  - Renouvellement du *proxy*
  - Lancement de l'application :
    - Copie des programmes BLAST en local via iRODS
    - Copie des banques BLAST en local via iRODS
    - Copie de la liste de fichiers à traiter
    - Copie des 100 fichiers à traiter : fichiers numéros *(N -1)x 100 +1* à *N x 100*
    - Boucle: pour i variant de (N -1)x 100 +1 à N x 100
      - traite le fichier i, copie le résultat via iRODS
      - tant que 95% du temps maximum n'est pas atteint, passe au fichier suivant
  - Post-traitement: envoi de mail, copie des *logs* via iRODS

Paramètres à ajuster:

- 1 - Nombre de séquences par fichiers 30
- 2 - Nombre max de fichiers par *job* 100
- 3 - Nombre de *jobs* par *job* paramétrique 40
- 4 - Délai entre la soumission des *jobs* paramétriques 20 mn

Lancement des jobs

Launcher :

script jdl

```
source launcher 1 1000 40 20 generic_par_irods_lst_new.jdl

premier job
dernier job
nombre de jobs par job paramétrique
délai de soumission entre les jobs
```

➔Le *launcher* propose une liste de noeuds.

Les *jobs* paramétriques seront répartis sur les noeuds choisis.

Ici on a lancé 1000 jobs (de 1 à 1000) par paquets de 40 :
25 jobs paramétriques, temps total soumission = 8h

Résultats

Rappel : 250,000 fichiers à traiter → 250,000 fichiers résultats

- 2,000,000,000 *hits* blast
- concaténation en 200 fichiers de 5 Go (1 To)
- moyenne de 50 fichiers par *job*
- environ 5000 *jobs* (plusieurs séries)
  - moyenne : 125 *jobs* x 40 paramétriques x 50 fichiers = 250 000
  - moyenne : *jobs* de 15 heures
- Calcul en 1 semaine au lieu de 8 ans

Données utilisées

1ère Release BGENR (2009-2010) 7,775,267 séquences

Calcul de similarité sur toutes les séquences de protéine :

BLAST de environ 8,000,000 x 8,000,000 séquences TIDRA

2ème Release BGENR(2011) 12,914,469 séquences

5,000,000 nouvelles BLAST x 5,000,000 nouvelles LOCAL

5,000,000 nouvelles BLAST x 7,500,000 anciennes TIDRA/GRISBI

5,000,000 nouvelles BLAST x 150,000 disparues GRISBI

Mise en place sur TIDRA 2ème release (2011)

2ère mouture de BGENR : séquences Uniprot + Ensembl + Autres
(33 millions de séquences, 12 millions non redondantes, 5 milions nouvelles)

5 millions de séquences,
soit 170,000 fichiers de 30 séquences

Nouveaux développements :

Outil DTM - Outil JJS - Acces iRODs

Mise en place sur TIDRA 2ème release

- Prototype du DTM (« Distributed Task Manager ») Yonny Cardenas
  - gLite + iRODS
  - système de base de données pour la gestion des *jobs* (runing, ended, canceled,etc.).
  - destination des *jobs* : à la fois en local, BQS et Grille
  - gestion automatisée de la production (gestion des erreurs, etc.)
  - l'utilisateur fournit seulement une liste de tâches, DTM s'occupe des *jobs* et de la production

Pour l'instant fonction uniquement pour BQS, adaptation a la grille en cours

Mise en place sur TIDRA 2ème release

- JJS Java Job Submission (Pascal Calvat)
  - commande « jjs-* » pour simplifier l'utilisation de la grille :
    - soumission
    - monitoring
    - gestion des proxy
  - gestion automatique de la répartition des jobs sur les noeuds de la grille, analyse de l'efficacité des différents noeuds
  - fonctionne via des jdl crées a partir d'un template avec une commande jjs
  - Notion de « production » c-à-d d'un ensemble de job

Mise en place sur TIDRA 2ème release

- Migration totale de nos données sur iRODS:
  - accès direct aux données : icd,ils,ipwd,iput,iget,*etc.*
  - programme de recherche des *hits* blast via iRODS (utilisation d'une API C pour iRODS)
  - utilisation des meta-données et des micro-services (à développer)

Mise en place sur TIDRA 2ème release

- Approche similaire à Release 1, mais :
  - *irods* intégré à la grille (plus besoin de récupérer les utilitaire irods)
  - gestion des proxies intégrée (plus besoin de déclarer les proxies dans le script)
  - Utilisation des commandes jjs (plus de jobs paramétriques)

⇨ Plus simple : 1 jdl + 1 script

Description du JDL avec JJS

N est le paramètre du *job* paramétrique

- Déroulement d'un *job* numéro N :
  - récupération de différents outils via lcg-cp X:
    - outils iRODS X
    - outils pour l'estimation du temps de calcul X
    - outils pour la gestion des *proxies* X
  - Renouvellement du *proxy* X
  - Lancement de l'application :
    - Copie des programmes BLAST en local via iRODS
    - Copie des banques BLAST en local via iRODS
    - Copie de la liste de fichiers à traiter
    - Copie des 100 fichiers à traiter : fichiers numéros *(N -1)x 100 +1* à *N x 100*
    - Boucle: pour i variant de (N -1)x 100 +1 à N x 100
      - traite le fichier i, copie le résultat via iRODS
      - tant que 95% du temps maximum n'est pas atteint, passe au fichier suivant X
  - Post-traitement: envoi de mail, copie des *logs* via iRODS

Mise en place sur TIDRA 2ème release

- Résultats
  - Production par « bloc » de 1000 jobs paramétriques, avec 20 fichiers par job soit 20 0000 fichiers traités.
    9 « blocs » de 1000 job à lancer pour traiter les 170 000 fichiers
- Qualité:
  - un proxy de 5 jours permet de perdre moins de jobs

Production	jobs	jobs récuperés	jobs OK	nb fichiers traités
Production 7	1000	734	411	10,535
Production 8	1000	881	676	17,174
Production 9	1000	982	786	19,835

Conclusion TIDRA
BGENR Release 1 et 2

- travail pionnier dans l'utilisation en production de iRODS avec gLite
  - 2 technologies/*middleware* integrées de manière transparente
    - gLite (EGEE) pour le calcul
    - iRODS pour le stockage
  - Après ajustement iRODS semble bien supporter la charge (à hauteur de 500 jobs en parallèle)

Intérêt d'une grille régionale

- Pas trop de nœuds (pas l'embaras du choix, connaissance des capacités des nœuds)
- En cas de problème sur un nœud, forte réactivité, facile de contacter un responsable

Perspectives TIDRA

- Fusion DTM et JJS

Technologie grille et services associés sur GRISBI

- 856 cœurs (cpu)
- 25 To de stockage
- 7 Sites/Plateformes bioinformatiques
  - IBCP (Lyon)
  - LBBE (Lyon)
  - ABiMS (Roscoff)
  - GenOuest (Rennes)
  - CBiB (Bordeaux)
  - Migale (Jouy-en-Josas)
  - GenoTool (Toulouse)
- Outils bioinformatiques
- Bases de données biologiques
- XtreemFS

- Collaboration avec 3 mésocentres pluri-disciplinaires à
  - Bordeaux
  - Lille
  - Strasbourg.

Mise en place sur GRISBI

- Outils disponibles
  - Tous les outils nécessaires à la bioinformatique sont disponible (tags)
  - Les bases de données biologiques sont disponible (tags)
  - Logiciel de statistique R installé (tags)
  - Systême de fichiers XtreemFS :
    - répertoire partagé en réseau par tous les jobs de grille
    - lié au proxy, totalement transparent
    - *grmount* : montage temporaire du répertoire
    - *grmount -u* : démontage

Stratégie « 1 job = 1 tâche » (sans XtreemFS)

- On utilise les outils disponibles sur la grille (blast)
- On dépose sur la grille la base de données blast à traiter (LFC)
- Fichier d'entrée en local, définis par le jdl (SandBox)
- Fichier de sortie en local, définis par le jdl (SandBox)

Description du JDL GRISBI « 1 job = 1 tâche »

InputSandBox: le fichier à traiter

OutputSandBox: le résultat

N est le paramètre du *job* paramétrique,

- Déroulement d'un *job* numéro N :
  - Copie des banques BLAST en local à partir de la grille (lcg-cp)
  - Traitement du fichier numéro N

- 170,000 fichiers à traiter : 170,000 jobs paramétriques
  - **ici 1 job = 1 fichier**
  - Essais sur une production de 1000 jobs paramétriques
  - 200 à 400 jobs en run simultanés

Stratégie « 1 job = 1 tâche » (sans XtreemFS)

Ca marche, mais:

- Très grand nombre de jobs
- On utilise la SandBox pour les sorties : Taille des sorties importantes, risque de limitation!
- Pas très efficace (temps de calcul court, on récupère la base blast pour chaque tâche, etc.)

On souhaiterais utiliser une approche similaire à celle utilisée dans TIDRA avec une liste de tâches comme argument du jdl:
Pas possible avec la SandBox

Stratégie « 1 job = n tâches » (avec XtreemFS)

- On utilise les outils disponibles sur la grille (blast)
- On dépose sur la grille la base de données blast à traiter (LFC)
- On dépose sur la grille tous les fichier d'entrées (XtreemFS)
- Chaque jobs traite plusieurs fichiers d'une liste (fournie par la *SandBox*)
- On stocke sur la grille les fichiers de sortie via XtreemFS

Description du JDL GRISBI/XtreemFS

InputSandBox: liste des fichiers à traiter

N est le paramètre du *job* paramétrique,

m est le nb de fichiers à traiter par job

m fichiers à traiter : fichiers numéros *(N -1)x m +1* à *N x m*

- Déroulement d'un *job* numéro N :
  - Copie des banques BLAST en local (lcg-cp)
  - Boucle: pour i variant de (N -1)x m +1 à N x m
    - Montage du répertoire grille
    - Vérifie si le fichier i a été calculé: si oui, saute à i +1
    - Copie du fichier i
    - Démontage du répertoire grille
    - Traite le fichier i avec blast,
    - Montage du répertoire grille
    - Copie du résultat
    - Démontage du répertoire grille
  - Fin de boucle

- 170,000 fichiers à traiter
  - **ici 1 job = 30 tâches**
    30 fichiers soit 90 séquences
  - Production : 6 soumissions par paquets de 1000 jobs paramétriques
  - on atteint 850 jobs en run simultanés
- Entrées et sorties sur XtreemFS

Conclusion GRISBI

Travail pionnier dans l'utilisation en production de XtreemFS avec gLite

- 2 technologies/*middleware* integrées
  - gLite (EGEE) pour le calcul
  - XtreemFS pour le stockage
- Après ajustement XtreemFS semble bien supporter la charge (à hauteur de 500 jobs en parallèle)

Temps de calcul GRISBI et TIDRA

TIDRA 1ère production

- 22 248 jobs
- 115 751 heures (13 ans) sur 1 processeur Intel Xeon.

TIDRA 2 ème production

- 15 797 jobs (répartis sur une dizaine de soumissions)
- 63 433 heures (7 ans) sur 1 processeur Intel Xeon.
- pics de 650 jobs en parallèle

GRISBI avec et sans XtreemFS

- 47 713 job
- 8 810 heures (1 an) sur 1 processeur Intel Xeon
- pics de 830 jobs en parallèle

CONCLUSIONS TIDRA vs GRISBI

Différences

TIDRA	GRISBI
Généraliste	Bioinformatique
Régionale	Nationale
Pas de softs/BDD installés	Logiciels et BDD bioinfo et biostats disponibles
Gestion des données avec iRODS	Gestion des données XtreemFS
Commandes jjs*	Commandes gri*
DTM+JJS (en devéloppement)	
Gestion des tâches (job « agent »)	
Adapté à des grandes productions	Adapté a des calculs de bioinfo/biostats classiques

Points communs

- Pas trop de nœuds (pas l'embaras du choix, connaissance des capacités des nœuds)
- En cas de problème sur un nœud, forte réactivité, facile de contacter un responsable
- Commandes « grilles » simplifiées